

# Asciidoctor User Manual

Sarah White, Dan Allen

# Table of Contents

Introduction to Asciidoctor .....	2
1. What is Asciidoctor? .....	3
1.1. The Big Picture .....	3
1.2. Asciidoctor on the JVM .....	3
1.3. Asciidoctor.js .....	3
1.4. Asciidoctor's Most Notable Benefits .....	3
1.5. Compared to Markdown .....	4
Quick Starts .....	11
2. Using the Command Line Interface .....	12
3. Using the Ruby API .....	13
Getting Started .....	14
4. System Requirements .....	15
5. Installing the Asciidoctor Ruby Gem .....	16
5.1. Install using <code>gem</code> .....	16
5.2. Install using Bundler .....	16
5.3. Install on Fedora .....	17
5.4. Install on Debian or Ubuntu .....	17
5.5. Install on Alpine Linux .....	17
6. Upgrading the Asciidoctor Ruby Gem .....	18
7. Extensions and Integrations .....	19
Terms and Concepts .....	20
8. Elements .....	21
9. Formatting Marks .....	23
9.1. Constrained quotes .....	23
9.2. Unconstrained quotes .....	23
9.3. When should I use unconstrained quotes? .....	23
9.4. Unconstrained formatting edge cases .....	24
9.5. Escaping unconstrained quotes .....	26
10. Attributes .....	28
10.1. Attribute Restrictions .....	28
10.2. Attribute Assignment Precedence .....	28
10.3. Using Attributes: Set, Assign, and Reference .....	30
10.4. Setting Attributes on a Document .....	31
10.5. Setting Attributes on an Element .....	37
10.6. Assigning Document Attributes Inline .....	42
Building a Document .....	46
11. Text Editor .....	47
12. Document Types .....	48

12.1. Inline doctype .....	48
13. Basic Document Anatomy .....	49
14. Header .....	50
14.1. Document Title .....	50
14.2. Author and Email .....	53
14.3. Revision Number, Date and Remark .....	58
14.4. Subtitle Partitioning .....	60
14.5. Metadata .....	61
14.6. Header Summary .....	63
15. Preamble .....	65
16. Sections .....	67
16.1. Titles as HTML Headings .....	68
16.2. Auto-generated IDs .....	69
16.3. Custom IDs .....	70
16.4. Multiple Anchors .....	71
16.5. Links .....	71
16.6. Anchors .....	72
16.7. Numbering .....	72
16.8. Discrete Headings (aka Floating Titles) .....	74
16.9. Section Styles .....	75
16.10. Sections Summary .....	75
17. Blocks .....	77
17.1. Title .....	77
17.2. Metadata .....	77
17.3. Delimited blocks .....	78
17.4. Built-in blocks summary .....	79
18. Paragraph .....	82
18.1. Alignment .....	82
18.2. Line Breaks .....	82
18.3. Lead Style .....	83
19. Text Formatting .....	85
19.1. Bold and Italic .....	85
19.2. Quotation Marks and Apostrophes .....	86
19.3. Subscript and Superscript .....	88
19.4. Monospace .....	88
19.5. Custom Styling With Attributes .....	90
20. Unordered Lists .....	91
20.1. Nested .....	92
20.2. Complex List Content .....	93
20.3. Custom Markers .....	99
20.4. Checklist .....	100

21. Ordered Lists .....	102
21.1. Nested.....	103
21.2. Numbering Styles .....	105
22. Description List .....	107
22.1. Question and Answer Style List.....	110
23. Tables .....	111
23.1. Columns.....	114
23.2. Column Formatting.....	115
23.3. Cell Formatting.....	122
23.4. Header Row.....	127
23.5. Footer Row .....	129
23.6. Table Width.....	129
23.7. Table Borders .....	131
23.8. Striping.....	134
23.9. Orientation .....	136
23.10. Nested Tables .....	136
23.11. Table Caption .....	137
23.12. Escaping the Cell Separator.....	140
23.13. Delimiter-Separated Values .....	141
23.14. Summary .....	145
24. Horizontal Rules .....	149
24.1. Markdown-style horizontal rules .....	149
25. Page Break .....	150
26. URLs .....	151
26.1. Link to Relative Files.....	154
26.2. Summary .....	155
27. Cross References .....	156
27.1. Automatic Anchors .....	156
27.2. Defining an Anchor .....	156
27.3. Internal Cross References .....	158
27.4. Validating Internal Cross References .....	159
27.5. Customizing the Cross Reference Text .....	160
27.6. Inter-document Cross References .....	162
28. Include Directive .....	165
28.1. Anatomy .....	165
28.2. Processing .....	166
28.3. File resolution .....	166
28.4. Partitioning large documents and using leveloffset .....	167
28.5. AsciiDoc vs non-AsciiDoc files .....	169
28.6. Select Portions of a Document to Include .....	169
28.7. Normalize Block Indentation .....	173

28.8. Include Content from a URI .....	174
28.9. Caching URI Content .....	175
28.10. Include a File Multiple Times in the Same Document .....	175
28.11. Using an Include in a List Item .....	176
29. Images .....	178
29.1. Setting the Location of Images .....	179
29.2. Putting Images in Their Place .....	181
29.3. Sizing Images .....	184
29.4. Taming SVGs .....	187
29.5. Summary .....	189
30. Video .....	191
30.1. YouTube and Vimeo videos .....	192
30.2. Supported Attributes .....	192
31. Audio .....	194
32. Admonition .....	195
33. Sidebar .....	197
34. Example .....	198
35. Prose Excerpts, Quotes and Verses .....	199
35.1. Quote .....	199
35.2. Verse .....	202
36. Comments .....	204
Controlling Your Content .....	205
37. Text Substitutions .....	206
37.1. Special Characters .....	207
37.2. Quotes .....	207
37.3. Attributes .....	209
37.4. Replacements .....	209
37.5. Macros .....	212
37.6. Post Replacements .....	213
37.7. Applying Substitutions .....	214
37.8. Incremental Substitutions .....	215
37.9. Preventing Substitutions .....	217
38. Literal Text and Blocks .....	218
39. Listing Blocks .....	220
39.1. To Wrap or to Scroll .....	222
40. Passthroughs .....	224
40.1. Passthrough Macros .....	224
40.2. Passthrough Blocks .....	227
41. Open Blocks .....	228
Enriching Your Content .....	230
42. Equations and Formulas .....	231

43. Activating stem support .....	232
43.1. Inline Stem Content .....	232
43.2. Block Stem Content .....	233
43.3. Using Multiple Stem Interpreters .....	233
43.4. Enabling STEM expressions in the DocBook Toolchain .....	234
44. User Interface Macros .....	236
44.1. Keyboard shortcuts .....	236
44.2. Menu selections .....	237
44.3. UI buttons .....	237
45. Icons .....	238
45.1. Admonition Icons .....	238
45.2. Inline Icons .....	241
45.3. Favicon .....	244
46. Syntax Highlighting Source Code .....	246
46.1. Enabling Source Highlighting .....	246
46.2. Available Source Highlighters .....	246
46.3. Applying Source Highlighting .....	247
46.4. Rouge .....	250
46.5. Pygments .....	253
46.6. CodeRay .....	256
46.7. highlight.js .....	259
47. Callouts .....	260
47.1. Copy and Paste Friendly Callouts .....	260
47.2. Callout Icons .....	262
48. Conditional Preprocessor Directives .....	263
48.1. Processing .....	263
48.2. ifdef Directive .....	263
48.3. ifndef Directive .....	265
48.4. Checking multiple attributes (ifdef and ifndef only) .....	265
48.5. ifeval directive .....	266
49. Docinfo Files .....	269
49.1. Head docinfo files .....	269
49.2. Footer docinfo files .....	270
49.3. Naming docinfo files .....	271
49.4. Locating docinfo files .....	272
49.5. Attribute substitution in docinfo files .....	272
50. Counters .....	274
Structuring, Navigating, and Referencing Your Content .....	276
51. Colophon .....	277
52. Table of Contents .....	278
52.1. In-Document Placement .....	279

52.2. Side Column Placement .....	280
52.3. Title .....	281
52.4. Levels .....	282
52.5. Using a TOC with Embeddable HTML .....	284
52.6. Table of Contents Summary .....	284
53. Abstract .....	286
54. Preface .....	288
55. Dedication .....	290
56. Book Parts and Chapters .....	291
57. Appendix .....	293
58. Glossary .....	296
59. Bibliography .....	297
60. Index .....	299
60.1. Index Terms .....	299
60.2. Index Catalog .....	300
61. Footnotes .....	301
61.1. Externalizing a Footnote .....	301
61.2. Footnotes in Headings .....	302
Processing Your Content .....	303
62. Selecting an Output Format .....	304
63. HTML .....	305
63.1. Using the Command Line .....	305
63.2. Using the Ruby API .....	306
63.3. Styling the HTML with CSS .....	306
63.4. Managing Images .....	308
63.5. CodeRay and Pygments Stylesheets .....	310
64. XHTML .....	311
65. DocBook .....	312
66. Man Pages .....	315
67. PDFs .....	319
68. Preview Your Content .....	320
68.1. Guard/Live Viewer .....	320
69. CLI Inputs and Outputs .....	321
69.1. Process Multiple Source Files from the CLI .....	321
69.2. Specifying an Output File .....	322
69.3. Piping Content Through the CLI .....	323
70. Running Asciidoctor Securely .....	325
70.1. Set the Safe Mode in the CLI .....	326
70.2. Set the Safe Mode in the API .....	326
70.3. Set Attributes Based on the Safe Mode .....	326
Customizing Your Output .....	328

71. Applying a Theme .....	329
72. Stylesheet Factory .....	333
72.1. Setting up the Factory .....	333
72.2. Creating a Theme .....	334
72.3. Applying a Stylesheet .....	334
72.4. Generate an HTML Document .....	335
72.5. External Preview .....	336
73. Slide Decks .....	337
74. Custom Backends .....	338
74.1. Storing Multiple Templates .....	338
75. Using Asciidoctor with Other Languages .....	339
75.1. Translating built-in labels .....	339
75.2. Translation .....	341
Publishing Your Content .....	342
76. Static Website Generators .....	343
76.1. Front Matter Added for Static Site Generators .....	343
Using Asciidoctor's API .....	345
77. Require the Library .....	346
78. Load and Convert Files .....	347
79. Load and Convert Strings .....	348
79.1. Embeddable output .....	348
79.2. Convert inline markup only .....	349
79.3. Convert to DocBook .....	349
80. Generate an HTML TOC .....	350
81. Provide Custom Templates .....	351
Extensions .....	352
82. Extension Points .....	353
83. Example Extensions .....	355
83.1. Preprocessor Example .....	355
83.2. Tree Processor Example .....	356
83.3. Postprocessor Example .....	358
83.4. Docinfo Processor Example .....	358
83.5. Block Processor Example .....	359
83.6. Compound Block Processor Example .....	360
83.7. Block Macro Processor Example .....	361
83.8. Inline Macro Processor Example .....	362
83.9. Include Processor Example .....	363
Build Integrations and Implementations .....	365
84. Java .....	366
85. Gradle .....	367
86. Maven .....	368

87. Apache Ant .....	369
88. JavaDoc .....	370
89. JavaScript .....	371
Conversions and Migrations .....	372
90. Migrating from AsciiDoc Python .....	373
90.1. Command Line Interface .....	373
90.2. Changed Syntax .....	375
90.3. Deleted and Deprecated Syntax and Attributes .....	376
90.4. Default HTML Stylesheet .....	377
90.5. Mathematical Expressions .....	377
90.6. AsciiDoc Python Extensions .....	378
90.7. Custom Extensions .....	378
90.8. Features Introduced by Asciidoctor .....	378
91. Convert DocBook XML to AsciiDoc .....	380
92. Convert Markdown to AsciiDoc .....	381
93. Convert Confluence XHTML to AsciiDoc .....	382
94. Convert MS Word to AsciiDoc .....	383
Resources .....	384
95. Copyright and License .....	385
96. Authors .....	386
97. Troubleshooting .....	387
Glossary .....	389
Appendix A: Catalog of Document Attributes .....	391
A.1. Environment Attributes .....	391
A.2. Built-in Attributes .....	392
A.3. Predefined Attributes for Character Replacements .....	407
Appendix B: CLI Options .....	410
B.1. Security Settings .....	410
B.2. Document Settings .....	410
B.3. Document Conversion .....	411
B.4. Processing Information .....	412
B.5. Program Information .....	412
Appendix C: Ruby API Options .....	413
Appendix D: Application Messages .....	418



If you find errors or omissions in this document, please don't hesitate to [submit an issue](#) or [open a pull request](#) with a fix. We also encourage you to ask questions and discuss any aspects of the project on the [mailing list](#) or in the [chat room](#). New contributors are always welcome!

This manual assumes you are using Asciidoctor to produce and convert your document. Asciidoctor implements more syntax, attributes and functions than the legacy AsciiDoc.py processor. [Migrating from AsciiDoc Python](#) lists which features are available to the Asciidoctor and AsciiDoc processors.

# Introduction to Asciidoctor

# Chapter 1. What is Asciidoctor?

Asciidoctor is a *fast* text processor and publishing toolchain for converting AsciiDoc content to HTML5, EPUB3, PDF, DocBook 5 (or 4.5) slidedecks and other formats. Asciidoctor is written in Ruby, packaged as a RubyGem and published to [RubyGems.org](#). The gem is also packaged in several Linux distributions, including Fedora, Debian and Ubuntu. Asciidoctor is open source, [hosted on GitHub](#), and released under the MIT license.

## 1.1. The Big Picture

Asciidoctor reads content written in plain text, as shown in the panel on the left in the image below, and converts it to HTML5, as shown rendered in the right panel. Asciidoctor adds a default stylesheet to the HTML5 document, as shown, to provide a pleasant out-of-the-box experience.

<pre>= AsciiDoc is Writing Zen Doc Writer &lt;doc.writer@example.com&gt; :icons: font  _Zen_ in the *art* of writing `plain text` with http://asciidoc.org[AsciiDoc].  [TIP] Use http://asciidoctor.org[Asciidoctor] for the best AsciiDoc experience.footnote:[Not to mention the best looking output!] Then icon[twitter[role=aqua] about it!  == Sample Section  [square] * item 1 * item 2  [source,ruby] ----- puts "Hello, World!" -----</pre>	<p>AsciiDoc is Writing Zen Doc Writer – <a href="mailto:doc.writer@example.com">doc.writer@example.com</a></p> <p>Zen in the <b>art</b> of writing <b>plain text</b> with <a href="#">AsciiDoc</a>.</p> <p><b>[TIP]</b> Use <a href="#">Asciidoctor</a> for the best AsciiDoc experience.<sup>[1]</sup> Then <a href="#">Twitter</a> about it!</p> <p><b>Sample Section</b></p> <ul style="list-style-type: none"><li>■ item 1</li><li>■ item 2</li></ul> <p><code>puts "Hello, World!"</code></p> <p><small>1. Not to mention the best looking output!</small></p>
AsciiDoc source	Rendered HTML

## 1.2. Asciidoctor on the JVM

You can run Asciidoctor on the JVM using JRuby. You can also use [AsciidoctorJ](#) to invoke Asciidoctor's APIs from Java and other JVM languages.

## 1.3. Asciidoctor.js

Asciidoctor can be used in JavaScript. [Opal](#) is used to transcompile the code from Ruby to JavaScript to make [Asciidoctor.js](#), which can be used wherever JavaScript runs, such as in a web browser or on Node.js.

## 1.4. Asciidoctor's Most Notable Benefits

While Asciidoctor aims to offer full compliance with the AsciiDoc syntax, it's more than just a clone.

### *Built-in and custom templates*

Asciidoctor uses a set of built-in ERB templates to generate HTML 5 and DocBook output that is structurally equivalent to what AsciiDoc produces. Any of these templates can be replaced by a custom template written in any template language available in the Ruby ecosystem. Custom

template rendering is handled by the [Tilt](#) template abstraction library. Tilt is one of the most popular gems in the Ruby ecosystem.

### *Parser and object model*

Leveraging the Ruby stack isn't the only benefit of Asciidoctor. Unlike the AsciiDoc Python implementation, Asciidoctor parses and converts the source document in discrete steps. This makes conversion optional and gives Ruby programs the opportunity to extract, add or replace information in the document by interacting with the document object model Asciidoctor assembles. Developers can use the full power of the Ruby programming language to play with the content in the document.

### *Performance and security*

No coverage of Asciidoctor would be complete without mention of its *speed*. Despite not being an original goal of the project, Asciidoctor has proven startlingly fast. It loads, parses, and converts documents a **100 times as fast** as the Python implementation. That's good news for developer productivity and good news for GitHub or any server-side application that needs to render AsciiDoc markup. Asciidoctor also offers several levels of security, further justifying its suitability for server-side deployments.

### *Beyond Ruby*

Asciidoctor's usage is not limited to the Ruby community. Thanks to [JRuby](#), a port of Ruby to the JVM, Asciidoctor can be used inside Java applications as well. Plugins are available for [Apache Maven](#), [Gradle](#), and [Rewrite](#). These plugins are based on the [AsciidoctorJ](#) for Asciidoctor.

Asciidoctor also ships with a command line interface (CLI). The Asciidoctor CLI, [asciidoctor](#), is a drop-in replacement for the [asciidoc.py](#) script from the AsciiDoc Python distribution.

#### **1.4.1. AsciiDoc Syntax Processing**

Asciidoctor reads and parses text written in the AsciiDoc syntax, then feeds the parse tree into a set of built-in templates to produce HTML5, PDF, DocBook 5, etc. You have the option of writing your own converter or providing [Tilt](#)-supported templates to customize the generated output or produce alternative formats.



Asciidoctor is a drop-in replacement for the original AsciiDoc Python processor ([asciidoc.py](#)). The Asciidoctor test suite has > 1,500 tests to ensure compatibility with the AsciiDoc syntax.

In addition to the standard AsciiDoc syntax, Asciidoctor recognizes additional markup and formatting options, such as font-based icons (e.g., `icon:fire[]`) and UI elements (e.g., `btn:[Save]`). Asciidoctor also offers a modern, responsive theme based on [Foundation](#) to style the HTML5 output.

## **1.5. Compared to Markdown**

The most compelling reason to choose a lightweight markup language for writing is to minimize the number of technical concepts an author must grasp in order to be immediately productive. In other words, the goal is to be able to *write without friction*.

### 1.5.1. Getting your start with Markdown

The defacto lightweight markup language is Markdown. (At least, that's what you call it at first). The main advantage of Markdown lies in its primitive syntax: its manual and cheatsheet are one and the same. But this advantage is also its greatest weakness.

As soon as authors need something slightly more complex than basic prose (e.g., tables, cross references, footnotes, embedded YouTube videos, etc.), they find themselves resorting to embedded HTML or seeking out more feature-rich implementations. Markdown has become a maze of different implementations, termed “flavors”, which make a universal definition evasive.



The IETF has declared “there is no such thing as “invalid” Markdown.” See [This Is Markdown! Or: Markup and Its Discontents](#).

Here's how the story inevitably goes. You start out with Markdown. Then it's Markdown + X. Then Markdown + X + Y. And down the rabbit hole you go. What's worse, X and Y often require you to sprinkle in HTML, unnecessarily coupling content with presentation and wrecking portability. Your instinct to choose Markdown is good. There are just better options.

### 1.5.2. Graduating to AsciiDoc

AsciiDoc presents a more sound alternative. The AsciiDoc syntax is more concise than (or at least as concise as) Markdown. At the same time, AsciiDoc offers power and flexibility without requiring the use of HTML or “flavors” for essential syntax such as tables, description lists, admonitions (tips, notes, warnings, etc.) and table of contents.

It's important to understand that AsciiDoc was initially designed as a plain-text alternative to the DocBook XML schema. AsciiDoc isn't stuck in a game of whack-a-mole trying to satisfy publishing needs like Markdown. Rather, the AsciiDoc syntax was explicitly designed with the needs of publishing in mind, both print and web. If the need arises, you can make full use of the huge choice of tools available for a DocBook workflow using Asciidoctor's DocBook converter. That's why mapping to an enterprise documentation format like DocBook remains a key use case for AsciiDoc.

And yet, AsciiDoc is simple enough to stand in as a better flavor of Markdown. But what truly makes AsciiDoc the right investment is that its syntax was designed to be extended as a core feature. This extensibility not only means that AsciiDoc has a lot more to offer, with room to grow, it also fulfills the objective of ensuring your content is maximally reusable.

### 1.5.3. Comparison by example

The following table shows the AsciiDoc syntax as it compares to Markdown. Since AsciiDoc supports a broader range of syntax than Markdown, this side-by-side comparison focuses mainly on areas where the syntax overlaps.

*A selection of AsciiDoc language features compared to Markdown*

Language Feature	Markdown	AsciiDoc
Bold (constrained )	<b>**bold**</b>	*bold*
Bold (unconstrained)	<b>**b**old</b>	<b>**b**old</b>
Italic (constrained )	<i>*italic*</i>	<u>_italic_</u>
Italic (unconstrained)	n/a	<u>--i--italic</u>
Monospace (constrained )	`monospace`	'monospace'
Monospace (unconstrained)	`m`onospace	``m``onospace
Link with label	[AsciiDoc](http://asciidoc.org)	http://asciidoc.org[AsciiDoc]
Relative link	[user guide](user-guide.html)	link:user-guide.html[user guide] xref:user-guide.adoc[user guide]
File link	[get the PDF]({% raw %}{{ site.url }}{% endraw %}/assets/mydoc.pdf)	link:{site-url}/assets/mydoc.pdf[get the PDF]
Cross reference	See link:#_usage[Usage].  <h2 id="_usage">Usage</h2>	See <<_usage>>.  == Usage
Block ID / anchor	<h2 id="usage">Usage</h2>	[#usage] == Usage

Language Feature	Markdown	AsciiDoc
Inline anchor	n/a	. [[step-1]]Download the software
Inline image w/ alt text	![Logo](/images/logo.png)	image:logo.png[Logo]
Block image w/ alt text	n/a	image::logo.png[Logo]
Section heading*	## Heading 2	== Heading 2
Blockquote*	> Quoted text. > > Another paragraph in quote.	---- Quoted text.  Another paragraph in quote. ----
Literal block	\$ gem install asciidoc	<p><i>Indented (by 1 or more spaces)</i></p> <p>\$ gem install asciidoc</p> <p><i>Delimited</i></p> <p>.... \$ gem install asciidoc ....</p>
Code block*	<pre>```java public class Person {     private String name;     public Person(String name) {         this.name = name;     } } ``` </pre>	[source,java] ---- public class Person {     private String name;     public Person(String name) {         this.name = name;     } } ----

Language Feature	Markdown	AsciiDoc
Unordered list	<ul style="list-style-type: none"> <li>* apples</li> <li>* orange</li> <li>* temple</li> <li>* navel</li> <li>* bananas</li> </ul>	<ul style="list-style-type: none"> <li>* apples</li> <li>* oranges</li> <li>** temple</li> <li>** navel</li> <li>* bananas</li> </ul>
Ordered list	<ol style="list-style-type: none"> <li>1. first</li> <li>2. second</li> <li>3. third</li> </ol>	<ol style="list-style-type: none"> <li>. first</li> <li>. second</li> <li>. third</li> </ol>
Thematic break (aka horizontal rule)*	<pre>*** * * *  ---  - - -  ---  - - -</pre>	<pre>...  ...</pre>
Typographic quotes (aka “smart quotes”)	Enabled through an extension switch, but offer little control in how they are applied.	<p>The '90s popularized a new form of music known as "grunge" rock. Its influence extended well beyond music.</p>
Document header	<p><i>Slapped on as “front matter”</i></p> <pre>--- layout: docs title: Writing posts prev_section: defining-frontmatter next_section: creating-pages permalink: /docs/writing-posts/ ---</pre>	<p><i>Native support!</i></p> <pre>= Writing posts :awestruct-layout: base :showtitle: :prev_section: defining-frontmatter :next_section: creating-pages</pre>

Language Feature	Markdown	AsciiDoc
Admonitions	<i>n/a</i>	<p>TIP: You can add line numbers to source listings by adding the word 'numbered' in the attribute list after the language name.</p>
Sidebars	<i>n/a</i>	<p>.Lightweight Markup **** Writing languages that let you type less and express more. ****</p>
Block titles	<i>n/a</i>	<p>.Grocery list * Milk * Eggs * Bread</p>
Includes	<i>n/a</i>	<pre>include::intro.adoc[]</pre>
URI reference	<p>Go [Home][home].</p> <p>[home]: https://example.org</p>	<p>:home: https://example.org</p> <p>Go {home}[Home].</p>
Custom CSS classes	<i>n/a</i>	<pre>[.path]_Gemfile_</pre>

\* Asciidoc also supports the Markdown syntax for this language feature.

You can see that AsciiDoc has the following advantages over Markdown:

- AsciiDoc uses the same number of markup characters or less when compared to Markdown in nearly all cases.
- AsciiDoc uses a consistent formatting scheme (i.e., it has consistent patterns).
- AsciiDoc can handle all permutations of nested inline (and block) formatting, whereas Markdown often falls down.
- AsciiDoc handles cases that Markdown doesn't, such as a proper approach to inner-word markup, source code blocks and block-level images.



Certain Markdown flavors, such as Markdown Extra, support additional features such as tables and description lists. However, since these features don't appear in "plain" Markdown, they're not included in the comparison table. But they're supported natively by AsciiDoc.

Asciidoctor, which is used for converting AsciiDoc on GitHub and GitLab, emulates "the good parts" of the Markdown syntax, like headings, blockquotes and fenced code blocks, making migration from Markdown to AsciiDoc fairly simple. For details about migration, see [Markdown Compatibility](#).

To read more about the shortcomings of Markdown, see these opinion pieces:

- [Why You Shouldn't Use "Markdown" for Documentation](#)
- [Markdown Considered Harmful](#)
- [Sundown on Markdown?](#)

# Quick Starts

# Chapter 2. Using the Command Line Interface

Asciidoctor's command line interface (CLI) is a drop-in replacement for the `asciidoc.py` command from the Python implementation.

If the Asciidoctor gem installed successfully, the `asciidoctor` command line interface (CLI) will be available on your PATH. To confirm that Asciidoctor is available, execute:

```
$ asciidoctor --version
```

The following information should be output in your terminal:

```
Asciidoctor 1.5.6.2 [https://asciidoctor.org]
```

To invoke Asciidoctor from the CLI and convert an `.adoc` file, execute:

```
$ asciidoctor <asciidoc_file>
```

This will use the built-in defaults for options and create a new file in the same directory as the input file, with the same base name, but with the `.html` extension.

There are many other options available, listed in [CLI Options](#).

Full help is provided in the [man page](#) or via:

```
$ asciidoctor --help
```

There is also an `asciidoctor-safe` command, which turns on safe mode by default, preventing access to files outside the parent directory of the source file. This mode is very similar to the safe mode of `asciidoc.py`.

# Chapter 3. Using the Ruby API

In addition to the command line interface, Asciidoctor provides a Ruby API. The API is intended for integration with other software projects and is suitable for server-side applications, such as Rails, Sinatra and GitHub.

Asciidoctor also has a Java API that mirrors the Ruby API. The Java API calls through to the Ruby API using an embedded JRuby runtime. See the [AsciidoctorJ project](#) for more information.

To use Asciidoctor in your application, you first need to require the gem:

```
require 'asciidoctor'
```

This statement makes all of the [public APIs in Asciidoctor](#) available to your script or application. You are now ready to start processing AsciiDoc documents.

The main entry points in the Asciidoctor API are the static methods to load or convert AsciiDoc documents, which we'll cover the next two chapters.

To parse a file into an `Asciidoctor::Document` object:

```
doc = Asciidoctor.load_file 'mysample.adoc'
```

You can get information about the document:

```
puts doc.title  
puts doc.attributes
```

More than likely, you will want to convert the document. To convert a file containing AsciiDoc markup to HTML 5, use:

```
Asciidoctor.convert_file 'mysample.adoc'
```

The command will output to the file *mysample.html* in the same directory.

You can convert the file to DocBook 5.0 by setting the `:backend` option to '`docbook`':

```
Asciidoctor.convert_file 'mysample.adoc', backend: 'docbook'
```

The command will output to the file *mysample.xml* in the same directory. If you're on Linux, you can view the file using [Yelp](#).

You can also use the API to [convert strings](#) and [load custom templates](#).

# Getting Started

# Chapter 4. System Requirements

Asciidoctor works on Linux, macOS and Windows.

Asciidoctor requires one of the following implementations of Ruby:

- Ruby 1.8.7
- Ruby 1.9.3
- Ruby 2 (2.0.0 or better)
- JRuby 1.7 (Ruby 1.8 and 1.9 modes)
- JRuby 9000
- Rubinius 2.0 (Ruby 1.8 and 1.9 modes)
- Opal (Javascript)

We expect Asciidoctor to work with other versions of Ruby as well. We welcome your help testing those versions if you are interested in seeing them supported.

# Chapter 5. Installing the Asciidoctor Ruby Gem

Asciidoctor can be installed using the `gem` command, Bundler or a Linux package manager.

## 5.1. Install using `gem`

To install Asciidoctor using the `gem` command:

1. Open a terminal
2. Type the following `gem` command

```
$ gem install asciidoctor
```

If the Asciidoctor gem installed successfully, the `asciidoctor` command line interface (CLI) will be available on your PATH. To confirm that Asciidoctor is available, execute:

```
$ asciidoctor --version
```

The following output should appear in your terminal:

```
Asciidoctor 1.5.6.2 [https://asciidoctor.org]
Runtime Environment (ruby 2.3.0p0 [x86_64-linux]) (lc=UTF-8 fs=UTF-8 in):- ex=UTF-8)
```

## 5.2. Install using Bundler

To install Asciidoctor for a project using Bundler:

1. Open your project's Gemfile
2. Add the `asciidoctor` gem using:

```
gem 'asciidoctor'
```

3. Save the Gemfile
4. Open a terminal
5. Install the gem using the `bundle` command:

```
$ bundle
```

## 5.3. Install on Fedora

To install Asciidoctor on Fedora (or RHEL via EPEL) using the [rubygem-asciidoctor](#) package:

1. Open a terminal
2. Run the installation command on Fedora:

```
$ sudo dnf install asciidoctor
```

The benefit of installing the gem using this method is that the package manager will also install Ruby and RubyGems if not already on your machine.

## 5.4. Install on Debian or Ubuntu

To install Asciidoctor on Debian or Ubuntu:

1. Open a terminal
2. Type the following `apt-get` command using sudo:

```
$ sudo apt-get install asciidoctor
```

The benefit of installing the gem via `apt-get` is that the package manager will also install Ruby and RubyGems if not already on your machine.

## 5.5. Install on Alpine Linux

To install Asciidoctor on Alpine Linux using the [asciidoctor](#) package:

1. Open a terminal
2. Type the following `apk` command using sudo:

```
$ sudo apk add asciidoctor
```

The benefit of installing the gem via `apk` is that the package manager will also install Ruby and RubyGems if not already on your machine.

# Chapter 6. Upgrading the Asciidoctor Ruby Gem

If you have an earlier version of Asciidoctor installed, you can update the gem using the `gem` command:

```
$ gem update asciidoctor
```

 If you accidentally use `gem install` instead of `gem update`, then you'll end up with both versions installed. To remove the older version, use the following `gem` command:

```
$ gem cleanup asciidoctor
```

On Fedora, you can update the package using:

```
$ sudo dnf update asciidoctor
```



Your Fedora system may be configured to automatically update packages, in which case no further action is required by you. Refer to the [Fedora docs](#) if you are unsure.

On Debian or Ubuntu, you can update the package using:

```
$ sudo apt-get upgrade asciidoctor
```

On Alpine Linux, you can update the package using:

```
$ sudo apk add --upgrade asciidoctor
```



The Linux packages may not be available right away after a release of the gem. It may take several weeks for the packages to be updated. If you need to upgrade to the latest version immediately, use the `gem` install option documented above.

# Chapter 7. Extensions and Integrations

See [Extensions](#).

# Terms and Concepts

All of the content in an Asciidoc document, including lines of text, predefined styles, and processing commands, is classified as either a block or an inline element. Within each of these elements are an array of styles, options, and functions that can be applied to your content.

This section will provide you with an overview of what each of these elements and sub-elements are and the basic syntax and rules for using them.

# Chapter 8. Elements

One or more lines of text in a document are defined as a block element. Block elements can be nested within block elements.

A document can include the following block elements:

- Header
- Title
- Author Info
- First Name
- Middle Name
- Last Name
- Email Address
- Revision Info
- Revision Number
- Revision Date
- Revision Remark
- Attribute Entry
- Preamble
- Section Title
- Section Body
- BlockId
- Block Title
- Block Macro
- Block
- Paragraph
- Delimited Block
- Table
- List
- Bulleted List
- Numbered List
- Description List
- Callout List
- List Entry
- List Label

- List Item
- Item Text
- List Paragraph
- List Continuation

An inline element performs an operation on a subset of the content within a block element.

Inline elements include:

- Quotes
- Replacements
- Special characters
- Special words
- Attribute references
- Inline macros

# Chapter 9. Formatting Marks

There are two categories of formatting marks for applying styles (i.e., formatting) to text, *constrained* and *unconstrained*. These formatting marks are referred to as *quotes* in the AsciiDoc syntax. This section covers their purpose, their differences and how to apply them.

## 9.1. Constrained quotes

In short, “constrained” means **around** a word or sequence of words.

Constrained quotes are single characters (often symbols) placed around a word. The “around” is defined by the fact that word characters do not appear immediately outside the enclosing marks.

You use this form to format a word that stands alone,

*When the word stands alone*

That is \*strong\* stuff!

to format a sequence of words,

*When there are multiple words*

That is \*really strong\* stuff!

or to format a word adjacent to punctuation, like an exclamation mark.

*When the word is adjacent to punctuation*

This stuff sure is \*strong\*!

## 9.2. Unconstrained quotes

In short, “unconstrained” means anywhere, including **within** a word.

Unconstrained quotes are repeated characters (often symbols) placed anywhere in the text, including within a word. The “within” is defined by the fact that a word character may appear directly outside one of the enclosing marks.

*Unconstrained formatting*

She spells her name with an ``h``, as in Sara\*\*h\*\*.

## 9.3. When should I use unconstrained quotes?

Consider the following questions:

- Is there a letter, number, underscore directly outside the formatting marks (on either side)?
- Is there a colon, semi-colon, or closing curly bracket directly before the starting formatting mark?
- Is there a space directly inside of the formatting mark?

If you answered “yes” to any of these questions, you need to switch to unconstrained (double formatting) quotes.

To help you determine whether a particular syntax pattern requires unconstrained quotes, consider the following scenarios:

#### *Constrained or Unconstrained?*

AsciiDoc	Result	Quote type	Reason
Sara__h__	Sarah	Unconstrained	The <b>a</b> is directly adjacent to (the left of) a formatting mark.
**B**old	<b>Bold</b>	Unconstrained	The <b>o</b> is directly adjacent to (the right of) a formatting mark.
&ndash;**2016**	&ndash;<strong>2016</strong>	Unconstrained	; is directly adjacent to (the left of) a formatting mark.
** bold **	<b>bold</b>	Unconstrained	There are spaces directly inside the formatting marks.
*2016*&ndash;	<strong>2016</strong> &ndash;	Constrained	The adjacent <b>&amp;</b> is not a letter, number, underscore, colon, or semi-colon.
*9*-to-*5*	9-to-5	Constrained	The adjacent hyphen is not a letter, number, underscore, colon, or semi-colon.

## 9.4. Unconstrained formatting edge cases

There are cases when it might seem logical to use constrained quotes, however unconstrained quotes are required. This happens because of the way the Asciidocparser parser (and the AsciiDoc Python parser) currently handles substitutions.

Substitutions may be applied by the parser before getting to the formatting marks, in which case the characters adjacent to those marks may not be what you see in the original source.

One such example is enclosing a monospaced phrase (i.e., codespan) inside typographic quotation marks, such as “**endpoints**”. Here’s how you would enter that:

*A monospaced phrase inside typographic quotes*

```
"```\endpoints```"
```

You might start with the following syntax:

```
"`endpoints`"
```

That only gives you “endpoints”, thought. The backticks contribute to making the typographic quotes.

Adding another set of backticks isn’t enough because the phrase now calls for unconstrained formatting marks. As a result, the parser ignores the inner set of backticks and instead interprets them as literal characters.

```
"``endpoints```"
```

So you have to unconstrained monospace inside the typographic quotes (three sets of backticks in total) to coerce the parser into formatting the phrase as monospace.

```
"```\endpoints```"
```

Although more rare, if what you’re after is to surround the monospaced phrase with normal double quotes, such as “`endpoints`”, then you need to interrupt the typographic quote syntax by applying a role to monospaced phrase or escaping the typographic quote. For example:

*A monospaced phase inside normal quotes*

```
"[.code]``endpoints``" or \"```\endpoints```"
```

Another example is a possessive, monospaced phrase that ends in an “s”. In this case, you must switch the monospaced phrase to unconstrained formatting.

```
The ``class`` static methods make it easy to operate on files and directories.
```

*Rendered possessive, monospaced phrase*

```
The class' static methods make it easy to operate on files and directories.
```

Alternately, you could encode the typographic apostrophe directly in the AsciiDoc source to get the same result without the need to use unconstrained formatting.

The `class` static methods make it easy to operate on files and directories.



This situation may improve in the future when Asciidoctor is switched to using a parsing expression grammar for inline formatting instead of the current regular expression-based strategy. For details, follow [issue #61](#).

## 9.5. Escaping unconstrained quotes

Unconstrained quotes are meant to match anywhere in the text, context free. However, that means you catch them formatting when you don't intend them to. Admittedly, these symbols are a bit tricky to type literally when the content calls for it. But being able to do so is just a matter of knowing the tricks, which this section will cover.

Let's assume you are typing the following two lines:

```
The __kernel qualifier can be used with the __attribute__ keyword...
```

```
#`CB###2`# and #'CB###3`#
```

In the first sentence, you aren't looking for any text formatting, but you're certainly going to get it. Double underscore is an unconstrained formatting mark. In the second sentence, you might expect **CB###2** and **CB###3** to be formatted in monospace and highlighted. However, what you get is a scrambled mess. The mix of constrained and unconstrained formatting marks in the line is ambiguous.

There are two (reliable) solutions for escaping unconstrained formatting marks:

- Use an attribute reference to insert the unconstrained formatting mark verbatim
- Wrap the text you don't want formatted in an inline passthrough

The attribute reference is preferred because it's the easiest to read:

```
:dbl_: __  
:3H: ###
```

```
The {dbl}_kernel qualifier can be used with the {dbl}_attribute{dbl}_ keyword...
```

```
#`CB{3H}2`# and #'CB{3H}3`#
```

This works because attribute expansion is performed *after* text formatting (i.e., quotes substitution) under normal substitution order. (Recall that backticks around text format the text in monospace but permit the use of attribute references).

Here's how you'd write these lines using the inline passthrough to escape the unconstrained formatting marks instead:

The `+_kernel+` qualifier can be used with the `+_attribute_+` keyword...

```
#`+CB###2+`# and #`+CB###3+`#
```

Notice the addition of the plus symbols. That's the closest thing to a text formatting escape. Everything between the plus symbols is escaped from interpolation (attribute references, text formatting, etc). However, the text still receives proper output escaping for HTML (e.g., `<` becomes `&lt;`).

The enclosure `'+TEXT+'` (text enclosed in pluses surrounded by backticks) is a special formatting combination in Asciidoc. It means to format TEXT as monospace, but don't interpolate formatting marks or attribute references in TEXT. It's roughly equivalent to Markdown's backticks. Since AsciiDoc offers more advanced formatting, the double enclosure is necessary.

The more brute-force solution to the inline passthrough approach is to use the `pass:c[]` macro, which is a more verbose (and flexible) version of the plus formatting marks.

The `pass:c[_kernel]` qualifier can be used with the `pass:c[_attribute_]` keyword...

```
#`pass:c[CB###2]`# and #`pass:c[CB###3]`#
```

As you can see, however, the macro is not quite as elegant or concise. In case you're wondering, the `c` in the target slot of the `pass:c[]` macro applies output escaping for HTML. Though not always required, it's best to include this flag so you don't forget to when it is needed.

Backslashes for escaping aren't very reliable in AsciiDoc. While they can be used, they have to be placed so strategically that they are rather finicky.

# Chapter 10. Attributes

Attributes are one of the features that sets Asciidoc apart from other lightweight markup languages. Attributes can activate features (behaviors, styles, integrations, etc) or hold replacement (i.e., variable) content.

In Asciidoc, attributes are classified as:

- [Environment attributes](#)
- [Built-in attributes](#)
- [Predefined attributes](#)
- [User-defined attributes](#)
- [API and Command Line Attributes](#)
- [Element Attributes](#)

## 10.1. Attribute Restrictions

All attributes have a name and a value (though the value may be implicit).

The attribute name:

- must be at least one character long,
- must begin with a word character (A-Z, a-z, 0-9 or \_) and
- must only contain word characters and hyphens.

In other words, the name cannot contain dots or spaces.

Although uppercase characters are permitted in an attribute entry (the place where an attribute is defined), the attribute name is converted to lowercase before being stored. The attribute name in an attribute reference is also converted to lowercase before the attribute is resolved. For example, [URI](#), [Uri](#) and [uRI](#) are all treated as [uri](#). (See [issue #509](#) for a proposed change to this restriction). A best practice is to only use lowercase for letters in the name and avoid starting the name with a number.

The attribute value:

- can be any inline content and
- can only contain line breaks if an explicit line continuation is used.

Certain attributes have a restricted range of allowable values. See the entries in the [Catalog of Document Attributes](#) for details.

## 10.2. Attribute Assignment Precedence

The attribute assignment precedence, listed from highest to lowest, is as follows:

- An attribute defined using the API or CLI
- An attribute defined in the document
- The default value of the attribute, if applicable

Let's use the `imagesdir` attribute to show how precedence works.

The default value for the `imagesdir` attribute is an empty string. Therefore, if the `imagesdir` attribute is not assigned a value (either in the document, API, or CLI), the processor will assign it the default value of empty string. If the `imagesdir` attribute is set in the document (meaning assigned a new value, such as `images`), that value will override the default value. Finally, if a value is assigned to the `imagesdir` attribute via the API or CLI, that value will override both the default value and the value assigned in the document.

It's possible to alter this order of precedence using a modifier, covered in the next section.

### 10.2.1. Altering the Attribute Assignment Precedence

You can allow the document to reassign an attribute that is defined via the API or CLI by adding the `@` precedence modifier to the end of the attribute value (or, since 1.5.7, the end of the attribute name). Adding this modifier lowers the precedence so that an assignment in the document still wins out. We sometimes refer to this as “soft setting” the attribute. This feature can be useful for assigning default values for attribute, but still letting the document control its own fate.



The `@` modifier is removed before the assignment is made.

Here's an example that shows how to set the `imagesdir` from the CLI with a lower precedence:

```
$ asciidoctor -a imagesdir=images@ doc.adoc
```

Since 1.5.7, you can place the modifier at the end of the attribute name:

```
$ asciidoctor -a imagesdir@=images doc.adoc
```

It's now possible to override the value of the `imagesdir` attribute from within the document:

```
= Document Title
:imagesdir: new/path/to/images
```

Let's update the attribute assignment precedence list defined earlier to reflect this additional rule:

- An attribute passed to the API or CLI
- An attribute defined in the document
- An attribute passed to the API or CLI whose value (or, since 1.5.7, name) ends in `@`
- The default value of the attribute, if applicable

Regardless of whether the precedence modifier is applied, an attribute assignment always overrides the default value.

## 10.3. Using Attributes: Set, Assign, and Reference

Before you can use an attribute in your document, it must be set. (Sometimes referred to as “toggling on” the attribute).

Some attributes are automatically set when Asciidoctor processes a document. You can also set (or override) an attribute for a document by declaring an attribute entry. For example:

```
:sectnums:
```

Many attributes can be assigned a value at the same time:

```
:leveloffset: 3
```

The value may be empty, a string (of characters) or a number. A string value may include references to other attributes.

Attributes can be unset using the bang symbol (!). The ! can be placed either before or after the attribute’s name.

For example, both:

```
:sectnums!:
```

and

```
:!sectnums:
```

mean unset the `sectnums` attribute. In this case, it tells Asciidoctor to not number the sections.

To soft unset an attribute from the CLI or API, you can use the following syntax:

```
!name=@
```

The leading ! unsets the attribute while the @ lowers the precedence of the assignment. This assignment is almost always used to unset a default value while still allowing the document to assign a new one. One such example is `sectids`, which is enabled by default. `!sectids=@` switches the setting off.

An *attribute reference* is an inline element composed of the name of the attribute enclosed in curly brackets. For example:

The value of `leveloffset` is `{leveloffset}`.

The attribute reference is replaced by the attribute's value when Asciidoctor processes the document. Referencing an attribute that is not set is considered an error and is handled specially by the processor.

The following sections will show you how to use attributes on your whole document, individual blocks, and inline elements.

## 10.4. Setting Attributes on a Document

An *attribute entry* is the primary mechanism for defining a document attribute in an AsciiDoc document. You can think of an attribute entry as a global variable assignment for AsciiDoc. The document attribute it creates becomes available from that point forward in the document. Attribute entries are also frequently used to toggle features.

An attribute entry consists of two parts: an attribute name and an attribute value. The attribute name comes first. It must be at the start of the line and must be enclosed in colons (e.g., `:name:`). If present, the attribute value is offset from the name part by at least one space (e.g., `:name: value`). Be aware that substitutions automatically get applied to the value by default, as described in [Substitutions in an attribute entry](#).

*Anatomy of an attribute entry*

```
:name: value
```

The attribute value is optional. A blank value is often used to set (i.e., activate) a boolean attribute (thus making a blank value implicitly true).

*Anatomy of an attribute entry to set a boolean attribute*

```
:name:
```

An exclamation point (!) before (or after) the attribute name unsets the attribute. In this case, the value is ignored.

*Anatomy of an attribute entry to unset an attribute*

```
:!name:
```

Attribute entries have the following characteristics:

### Attributes entries can:

- be assigned to a document:
  - through the CLI or API
  - in the document's header

- in the document’s body
- be unset (turned off) with a leading (or trailing) ! added to the name
- have default values (in the case of a built-in attribute)
- have alternate values (in the case of a built-in attribute)
- span multiple, contiguous lines
- include inline AsciiDoc content

#### **Attribute entries can not:**

- override locked attributes assigned from the command line
- include AsciiDoc block content (such as, bulleted lists or other types of whitespace-dependent markup)

Attributes are typically defined in the document header, though they may also be defined in the body of the document. Once set, an attribute (and its value) are available for use for the remainder of the document. Unless locked by the API, attributes may be reassigned at any subsequent point in the document.

#### **10.4.1. Attribute entry use cases**

Attributes entries serve three main purposes:

1. Toggle or configure built-in features
2. Set built-in asset locations
3. Content reuse

#### **Setting built-in attributes**

Numerous attribute are reserved for special purposes. These built-in attributes can be used to toggle behavior, such as the table of contents, or control the generated output, such as selecting or configuring a converter. Many built-in attributes only take effect when defined in the document header.

For example, to enable the built-in table of contents, you can define (i.e., set) the `toc` attribute using an attribute entry in the document header as follows:

```
:toc:
```

When the value following an attribute is left empty, as it is in the example above, the default value will be assigned (if any). The default value for `toc` is `auto`. Therefore, the table of contents will be placed in the default location (below the document’s title) when the document is converted.

If you want the table of contents to be placed on the right side of the document, you must assign the attribute a new value.

```
:toc: right
```

The `right` value will override the default value of `auto`. The value assigned to an attribute in the document header replaces the intrinsic value (assuming the attribute is not locked).

## Setting asset locations

You can also use attributes to set the base path to images (default: `empty`), icons (default: `./images/icons`), and stylesheets (default: `./stylesheets`).

*Base asset path configuration example*

```
:imagesdir: ./images  
:iconsdir: ./icons  
:stylesdir: ./styles
```

## Content reuse

If you're familiar with writing in XML, you might recognize a document attribute as a user-defined entity. When you find yourself typing the same text repeatedly, or text that often needs to be updated, consider assigning it to a document attribute and use that instead.

A prime use case for attribute entries is to promote frequently used URLs and links to the top of the document.

*URL attribute entry*

```
:url-fedpkg: https://apps.fedoraproject.org/packages/rubygem-asciidoc
```

Now you can refer to this attribute entry anywhere in the document (where attribute substitution is performed) by surrounding its name in curly braces.

Instead of having to type the URL out longhand in the link macro, as follows:

*A case for using an attribute reference*

Did you know there's an `https://apps.fedoraproject.org/packages/rubygem-asciidoc`[Asciidoc package for Fedora]?

We can replace the target side of the link macro with a reference to our attribute.

*url-fedpkg attribute usage example*

Did you know there's an `{url-fedpkg}`[Asciidoc package for Fedora]?

To save even more typing, you can store the whole link in an attribute value.

## *Link attribute entry*

```
:link-fedpkg: https://apps.fedoraproject.org/packages/rubygem-asciidoc[Asciidoc package for Fedora]
```

Now you insert this link anywhere in the document using an attribute reference.

### *link-fedpkg attribute usage example*

Did you know there's an {link-fedpkg}?

Note that the link substitution occurs *after* the attribute reference is resolved. This works thanks to the default order of substitutions on a paragraph. If you want the link macro to be resolved eagerly at the time the attribute is assigned, you need to enclose it in a pass macro.

### *Link attribute entry resolved eagerly*

```
:link-fedpkg: pass:m[https://apps.fedoraproject.org/packages/rubygem-asciidoc[Asciidoc package for Fedora]]
```

Now you can use this link in a section title (where the order of substitutions is different). Let's dive deeper into which substitutions are applied to an attribute entry and how to alter them.

#### **10.4.2. Substitutions in an attribute entry**

The AsciiDoc processor automatically applies substitutions from the header substitution group to the value of an attribute entry prior to the assignment (regardless of where the attribute entry is declared in the document). The header substitution group replaces [special characters](#) and [attribute references](#). This is the same group that gets applied to metadata lines (author and revision information) in the document header. To learn about how these substitutions work, refer to the [Substitutions](#) chapter.

#### **10.4.3. Altering attribute entry substitutions**

If you want the value of an attribute entry to be used **as is** (not subject to substitutions), or you want to alter the substitutions that are applied, you can enclose the value in the [inline pass macro](#) (i.e., `pass:[]`). The inline pass macro accepts a list of zero or more substitutions in the target slot, which can be used to control which substitutions are applied to the value. If no substitutions are specified, no substitutions will be applied.

In order for the inline macro to work in this context, it must completely surround the attribute value. If it's used anywhere else in the value, it is ignored.

Here's how to prevent substitutions from being applied to the value of an attribute entry:

```
:cols: pass:[.>2,.>4]
```

This might be useful if you're referencing the attribute in a place that depends on the unaltered text, such as the value of the `cols` attribute on a table.

Here's how to apply the [quotes substitution](#) to the value of an attribute entry:

```
:app-name: pass:quotes[MyApp^2^]
```

Internally, the value is stored as `MyApp2`. You can inspect the value stored in an attribute using this trick:

```
[subs=attributes+]
-----
{app-name}
-----
```

You can also specify the substitution using the single-character alias, `q`.

```
:app-name: pass:q[MyApp^2^]
```

The inline pass macro kind of works like an attribute value preprocessor. If the processor detects that an inline pass macro completely surrounds the attribute value, it:

1. reads the list of substitutions from the target slot of the macro
2. unwraps the value from the macro
3. applies the substitutions to the value

If the macro is absent, the value is processed with the header substitution group.

You can also change the substitutions that are applied to an attribute at the time it is resolved. This is done by manipulating the substitutions applied to the text where it is referenced. For example, here's how we could get the processor to apply quote substitutions to the value of an attribute:

```
:app-name: MyApp^2^
[subs="specialchars,attributes,quotes,replacements,macros,post_replacements"]
The application is called {app-name}.
```

Notice that we've swapped the order of the `attributes` and `quotes` substitutions. This strategy is akin to postprocessing the attribute value.

#### 10.4.4. Splitting attribute values over multiple lines

When an attribute value is very long, it's possible to split it (i.e., soft-wrap) across multiple lines.

Let's assume we are working with the following attribute entry:

### *A long, single-line attribute*

```
:long-value: If you have a very long line of text that you need to substitute regularly in a document, you may find it easier to split it neatly in the header so it remains readable to the next person reading your docs code.
```

You can split the value over multiple lines to make it more readable by inserting a space followed by a backslash (i.e., \) at the end of each continuing line.

### *A long, multiline attribute (soft wrapped)*

```
:long-value: If you have a very long line of text \
that you need to substitute regularly in a document, \
you may find it easier to split it neatly in the header \
so it remains readable to folks reading your docs code.
```

The backslash and the newline that follows will be removed from the attribute value when the attribute entry is parsed, making this second example effectively the same as the first. The space before the backslash is preserved, so you have to use this technique at a natural break point in the content.

You can force an attribute value to hard wrap by adding a plus surrounded by spaces before the backslash.

### *An attribute value with hard line breaks*

```
:haiku: Write your docs in text, + \
AsciiDoc makes it easy, + \
Now get back to work!
```

This syntax ensures that the newlines are preserved in the output document as hard line breaks.

## **10.4.5. Attribute limitations**

Attributes let you do a surprising amount of formatting for what is fundamentally a text replacement tool.

It may be tempting to try and extend attributes to be used for complex replaceable markup.

### **Supported**

Basic in-line AsciiDoc markup is permitted in attribute values, such as:

- attribute references
- text formatting (usually wrapped in a pass macro)
- inline macros (usually wrapped in a pass macro)

### **Unsupported**

Complex AsciiDoc markup is not permitted in attribute values, such as:

- lists
- multiple paragraphs
- other whitespace-dependent markup types

## 10.5. Setting Attributes on an Element

An attribute list can apply to blocks, inline quotes text, and macros. The attributes and their values contained in the list will take precedence over attribute entries.

*Anatomy of an attribute list*

```
[positional_attribute1, positional_attribute2, name_attribute1="value"]
```

Attribute lists:

1. apply to blocks as well as macros and inline quoted text
2. can contain positional and named attributes
3. take precedence over global attributes

### 10.5.1. Positional Attribute

Positional attributes are un-named values at the start of the attribute list. The attribute that they are assigned to depends on the type of the element:

- `icon`: (1) size
- `image`: and `image::`: (1) alt text, (2) width, (3) height
- Delimited blocks: (1) block name (aka style)
- Other inline quoted text: (1) role

For example, the following two image macros are equivalent.

```
image::sunset.jpg[Sunset,300,400]
```

```
image::sunset.jpg[alt=Sunset,width=300,height=400]
```

The second macro is just a duplicate of the first macro written out longhand.

### 10.5.2. Named Attribute

A named attribute consists of a name and a value separated by an `=` character (e.g., `name=value`).

If the value contains a space, comma, or quote character, it must be enclosed in double or single quotes (e.g., `name="value with space"`). In all other cases, the surrounding quotes are optional. If present, the enclosing quotes are dropped from the parsed value.

To undefine a named attribute, set the value to **None** (case sensitive).

### 10.5.3. Attribute List Substitutions

Attribute references are expanded before the block attribute list is processed. Therefore, it's not necessary to force substitutions to be applied if you simply want to use a document attribute reference in a block attributes.

If the attribute name (for a positional attribute) or value (for a named attribute) is enclosed in single quotes (e.g., `title='Processed by https://asciidoc.org[Asciidoctor]'`), **normal substitutions** are applied to the value at assignment time (with some exceptions). No special processing is performed if the value is not enclosed in quotes or is enclosed in double quotes.

If the attribute value contains the same quote character being used to enclose the value, escape the quote character in the value by prefixing it with a backslash (e.g., `title="\\"Dark Horse\\" is a song by George Harrison"`).

### 10.5.4. Style

The style attribute is the first positional attribute in an attribute list. It specifies a predefined set of characteristics that should apply to a block element or macro.

For example, a paragraph block can be assigned one of the following built-in style attributes:

- normal (default, so does not need to be set)
- literal
- verse
- quote
- listing
- TIP
- NOTE
- IMPORTANT
- WARNING
- CAUTION
- abstract
- partintro
- comment
- example
- sidebar
- source

## 10.5.5. Id

The id attribute specifies a unique name for an element. That name can only be used once in a document.

An id has two purposes:

1. to provide an internal link or cross reference anchor for the element
2. to reference a style or script used by the output processor

### Block Assignment

In an attribute list, there are two ways to assign an id attribute to a block element.

1. Prefixing the name with a hash (#).
2. Specifying the name with `id=<name>`.

```
[#goals]
* Goal 1
* Goal 2
```

Let's say you want to create a blockquote from an open block and assign it an ID and role. You add `quote` (the block style) in front of the # (the ID) in the first attribute position, as this example shows:

```
[quote#roads, Dr. Emmett Brown]
-----
Roads? Where we're going, we don't need roads.
-----
```



The order of ID and role values in the shorthand syntax does not matter.



If the ID contains a ., you must define it using either a longhand assignment (e.g., `id=classname.propertyname`) or the anchor shorthand (e.g., `[[classname.propertyname]]`). This is necessary since the . character in the shorthand syntax is the delimiter for a role, and thus gets misinterpreted as such.

### Inline Assignment

The id (#) shorthand can be used on inline quoted text.

*Quoted text block with id assignment using Asciidoctor shorthand*

```
[#free_the_world]*free the world*
```

## 10.5.6. Role

An element can be assigned numerous roles.

### Block Assignment

In an attribute list, there are two ways to assign a role attribute to a block element.

1. Prefixing the name with a dot (.).
2. Specifying the name with `role=<name>`.

```
[.summary]
```

```
* Review 1
```

```
* Review 2
```

```
[role="summary"]
```

```
* Review 1
```

```
* Review 2
```

To specify multiple roles using the shorthand syntax, separate them by dots.

```
[.summary.incremental]
```

```
* Review 1
```

```
* Review 2
```

```
[role="summary,incremental"]
```

```
* Review 1
```

```
* Review 2
```

### Inline Assignment

The role (.) shorthand can be used on inline quoted text.

*Quoted text with role assignments using traditional AsciiDoc syntax*

```
[big goal]*free the world*
```

*Quoted text with role assignments using Asciidoc doctor shorthand*

```
[.big.goal]*free the world*
```



The attribute list preceding formatted text can be escaped using a backslash (e.g., `\[role]*bold*`). In this case, the text will still be formatted, but the attribute list will be unescaped and output verbatim.

### *Role-playing for text enclosed in backticks*

To align with other formatted (i.e., quoted) text in AsciiDoc, roles can now be assigned to text enclosed in backticks.

Given:

```
[.rolename]`monospace text`
```

the following HTML is produced:

```
<code class="rolename">monospace text</code>
```

Using the shorthand notation, an id (i.e., anchor) can also be specified:

```
[#idname.rolename]`monospace text`
```

which produces:

```
<a id="idname"></a><code class="rolename">monospace text</code>
```

## 10.5.7. Options

The options attribute is a versatile named attribute that can contain a comma separated list of values.

It can also be defined globally with an attribute entry.

### Block Assignment

In an attribute list, there are three ways to assign an options attribute to a block element.

1. Prefixing the value with a percent sign (%).
2. Specifying the value with `opts=<name>`
3. Specifying the value with `options=<name>`.

Consider a table block with the three option values `header`, `footer`, and `autowidth`.

Here's how the options are assigned to the table using the shorthand notation (%).

### *Shorthand Asciidoctor syntax*

```
[%header%footer%autowidth]
|===
| Cell A | Cell B
|===
```

Here's how the options are assigned to the table using [options](#).

### *Traditional AsciiDoc syntax*

```
[options="header,footer,autowidth"]
|===
| Cell A | Cell B
|===
```

Let's consider the options when combined with other attributes.

### *Shorthand Asciidoctor block syntax*

```
[horizontal.properties%step]
property 1:: does stuff
property 2:: does different stuff
```

### *Traditional AsciiDoc block syntax*

```
[horizontal, role="properties", options="step"]
property 1:: does stuff
property 2:: does different stuff
```

## 10.6. Assigning Document Attributes Inline

Document attributes can be assigned using the following syntax:

```
{set:<attrname>[!][:<value>]}
```

For example:

```
{set:sourcedir:src/main/java}
```

is effectively the same as:

```
:sourcedir: src/main/java
```

This is important for being able to assign document attributes in places where attribute entry lines are not normally processed, such as in a table cell.

### 10.6.1. Handle Missing or Undefined Attributes

As a result of a misconfigured document or inadvertent substitution, an attribute reference may point to a non-existent attribute (e.g., `{does-not-exist}`). It could be that the attribute reference itself undefines the attribute (e.g., `{set:attribute-no-more!}`). You'll want to think about how you want the processor to handle these situations and configure it accordingly.

AsciiDoc Python simply drops any line that contains a reference to a missing attribute. This “feature” was designed with AsciiDoc Python’s own template language in mind, which is also based on the AsciiDoc syntax. However, this behavior was never really intended for use in regular AsciiDoc documents. The behavior is frustrating for the author, editor, or reader because it’s not immediately obvious when a line goes missing. Discovering the absence of certain line often requires a painstaking read-through of the document or section, if it’s even noticed at all.

Asciidoctor offers two attributes to alleviate this inconvenience: `attribute-missing` and `attribute-undefined`.

#### Missing attribute

The `attribute-missing` attribute controls how missing (i.e., unresolved) references are handled. By default, missing references are left intact so the integrity of the document is preserved (`skip`). However, that mode doesn’t help the author track down these references.

To help with that task, Asciidoctor can be configured to warn when a missing reference is encountered (`warn`). Asciidoctor can also emulate the behavior of AsciiDoc Python (`drop-line`), or offer something in between (`drop`).

Here are the four possible values of the `attribute-missing` attribute:

##### `skip`

leaves the reference intact without issuing a warning (default setting)

##### `drop`

drops the reference, but not the whole line

##### `drop-line`

drops the whole line on which the reference occurs (matches behavior of AsciiDoc Python)

##### `warn`

leaves the reference intact, but also prints a warning about the missing attribute (recommended)

Consider the following line of AsciiDoc:

```
Hello, {name}!
```

Here's how the line is handled in each case, assuming the `name` attribute is not defined:

### skip

```
Hello, {name}!
```

### drop

```
Hello, !
```

### drop-line

### warn

```
asciidoc: WARNING: skipping reference to missing attribute: name
```

If you want the processor to fail when the document contains a missing attribute, set the `attribute-missing` attribute to `warn` and pass the `--failure-level=WARN` option to the processor.

```
$ asciidoc -a attribute-missing=warn --failure-level=WARN doc.adoc
```

When using the API, you can consult the logger for the max severity of all messages reported or look for specific messages in the stack.

There are several exceptions when the `attribute-missing` attribute is not strictly honored. One of those cases is the include directive. If a missing attribute is found in the target of an include directive, the processor will issue a warning about the missing attribute and drop the include directive. This behavior was chosen because showing the unresolved include directive to the reader is messy.

Another case is the block image macro. If a missing attribute is found in the target of an include directive, the processor will issue a warning about the missing attribute, but leave the image macro unresolved so as to show it as alt text.

A missing attribute reference can safely be used in an ifeval clause without any side effects (i.e., `drop`) since often the purpose of that statement is to determine whether an attribute resolves to a value.

## Undefined attribute

The attribute `attribute-undefined` controls how expressions that undefine an attribute are handled. By default, the line is dropped since the expression is a statement, not content.

This attribute has two possible values:

### drop

substitute the expression with an empty string after processing it

## **drop-line**

drop the line that contains this expression (default setting; matches behavior of AsciiDoc Python)

The option `skip` doesn't make sense here since the statement is not intended to produce content.

Consider the following declaration:

```
{set:name!}
```

Depending on whether attribute-undefined is `drop` or `drop-line`, either the statement or the line that contains it will be discarded. It's reasonable to stick with the compliant behavior, `drop-line`, in this case.



We recommend putting any statement that undefines an attribute on a line by itself.

# Building a Document

# Chapter 11. Text Editor

Since AsciiDoc syntax is just **plain text**, you can write an AsciiDoc document using *any* text editor. You don't need complex word processing programs like Microsoft Word, OpenOffice Writer or Google Docs. In fact, you shouldn't use these programs because they add cruft to your document that you can't see that makes conversion tedious.

While it's true any text editor will do, an editor that supports syntax highlighting for AsciiDoc may be more helpful. The **color** brings contrast to the text, making it easier to read. The highlighting also confirms when you've entered the correct syntax for an inline or block element.

The most popular application for editing plain text on macOS is **TextMate**. A similar choice on Linux is **GEdit**. On Windows, stay away from Notepad and Wordpad because they produce plain text which is not cross-platform friendly. Opt instead for a competent text editor like **NotePad++**. If you're a programmer (or a writer with an inner geek), you'll likely prefer **Vim**, **Emacs**, or **Sublime Text**, all of which are available cross-platform. For those that work on multiple platforms, **Atom** is a consistent choice with many add-on packages for working with AsciiDoc files. The key feature all these editors share is [syntax highlighting for AsciiDoc](#).



Previewing the output of the document while editing can be helpful. To learn how to setup instant preview, check out the [Editing AsciiDoc with Live Preview](#) tutorial.

# Chapter 12. Document Types

## Article (keyword: `article`)

The default doctype. In DocBook, includes the appendix, abstract, bibliography, glossary, and index sections.

## Book (keyword: `book`)

Builds on the article doctype with the additional ability to use a top-level title as part titles, includes the appendix, dedication, preface, bibliography, glossary, index, and colophon. There's also the concept of a multi-part book, but the distinction from a regular book is determined by the content. A book only has chapters and special sections, whereas a multi-part book is divided by parts that each contain one or more chapters or special sections.

## Man page (keyword: `manpage`)

Used for producing a roff or HTML-formatted [man page](#) (short for manual page) for Unix and Unix-like operating systems. This doctype instructs the parser to recognize a special document header and section naming conventions for organizing the AsciiDoc content as a manual page. Refer to [Man Pages](#) for details on how to compose AsciiDoc for this purpose.

## Inline (keyword: `inline`)

Asciidoc only. There may be cases when you only want to apply inline AsciiDoc formatting to input text without wrapping it in a block element. For example, in the [Asciidoclet project](#) (AsciiDoc in Javadoc), only the inline formatting is needed for the text in Javadoc tags.

## 12.1. Inline doctype

The rules for the inline doctype are as follows:

- Only a single paragraph is read from the AsciiDoc source.
- Inline formatting is applied.
- The output is not wrapped in the normal paragraph tags.

Given the following input:

```
http://asciidoc.org[Asciidoc] is a _lightweight_ markup language...
```

Processing it with the options `doctype=inline` and `backend=html5` produces:

```
<a href="http://asciidoc.org">Asciidoc</a> is a <em>lightweight</em> markup language&#8230;
```

The inline doctype allows the Asciidoc processor to cover the full range of applications, from unstructured (inline) text to full, standalone documents!

# **Chapter 13. Basic Document Anatomy**

# Chapter 14. Header

The document header is a special set of contiguous lines at the start of the document that encapsulates the document title, author and revision information, and document-wide attributes (either built-in or user-defined).

The header typically begins with a document title, though this element is optional. If a document title is specified, it may be immediately followed by two optional lines of text to set the author and revision information. Finally, the header may declare document-wide attributes (built-in or user-defined) using attribute entries. Attribute entries can be placed anywhere in the header, including above the document title, though the preferred placement is below the document title, if present. Since the document title is optional, it's possible for the header to only consist of attribute entries.

The first block content (e.g., a paragraph) or a blank line in the document marks the end of the header. Any attributes defined below the document header will *not* be scoped to the document.

## **The document header must *not* contain any blank lines or block content!**

Line comments may be used in the header, but only if those lines are directly adjacent to other lines in the header.

The header is *optional* when the `doctype` is `article` or `book`. However, a header is required when the document type is `manpage`. The requirements for a manual page (man page) are described in the [man pages](#) section.

The header (document title, author, and revision information) is included by default when converting to a standalone document. If you do not want the header of a document to be displayed, set the `noheader` attribute in the document's header (or set the attribute using the API or CLI).

### **Front matter**

Many static site generators, such as Jekyll and Middleman, rely on front matter added to the top of the document to determine how to convert the content. Asciidoctor has a number of attributes available to correctly handle front matter. See the [static website generators](#) section to learn how Asciidoctor integrates with static website generators.

Now let's explore the document title in detail.

## **14.1. Document Title**

The document title resembles a level-0 section title, which is written using a single equal sign followed by at least one space (i.e., `=` ), then the text of the title. The document title must be the first level-0 section title in the document. The only content permitted above the document title are blank lines, comment lines and document-wide attribute entries.

Here's an example of a document title followed by a short paragraph. Notice the blank line between the document title and the first line of prose. That blank line is what offsets the document header

from the body.

*Document with a title*

= The Dangerous and Thrilling Documentation Chronicles

This journey begins on a bleary Monday morning.

*Result: Rendered document title*

# The Dangerous and Thrilling Documentation Chronicles

This journey begins on a bleary Monday morning.

When the `doctype` is `article` or `manpage`, the document can only have one level-0 section title. In contrast, the `book` document type permits multiple level-0 section titles. When the `doctype` is `book`, the first level-0 section title, located in the header, is the document's title and subsequent level-0 section titles are the part titles.

## 14.1.1. doctitle attribute

A document's title is assigned to the built-in `doctitle` attribute. The `doctitle` attribute can be referenced anywhere in a document and resolves to the document's title when displayed.

*Referencing the doctitle attribute*

= The Dangerous and Thrilling Documentation Chronicles

{doctitle} begins on a bleary Monday morning.

*Result: doctitle output*

# The Dangerous and Thrilling Documentation Chronicles

The Dangerous and Thrilling Documentation Chronicles begins on a bleary Monday morning.

The `doctitle` attribute can also be used to set the document title instead of using a level-0 section title. However, the attribute must still be set in the document header.

## 14.1.2. Document subtitle

Asciidoctor recognizes a subtitle in the primary level-0 heading. If the primary title contains at least one colon followed by a space (i.e. : ), Asciidoctor treats the text after the final colon-space sequence as the subtitle.



The subtitle is not distinguished from the main title in the `html5` output. It's only distinguished from the main title when using the `docbook`, `epub3`, and `pdf` converters.

*Document with a subtitle*

```
= The Dangerous and Thrilling Documentation Chronicles: A Tale of Caffeine and Words  
It began on a bleary Monday morning.
```

In this example, the following is true:

### Main title

The Dangerous and Thrilling Documentation Chronicles

### Subtitle

A Tale of Caffeine and Words

*Document with a subtitle and multiple colons*

```
= A Cautionary Tale: The Dangerous and Thrilling Documentation Chronicles: A Tale of  
Caffeine and Words  
It began on a bleary Monday morning.
```

In this example, the following is true:

### Main title

A Cautionary Tale: The Dangerous and Thrilling Documentation Chronicles

### Subtitle

A Tale of Caffeine and Words

Instead of using a colon followed by a space as the separator characters between the main title and the subtitle, you can specify a custom separator using the `title-separator` attribute.

*Document with a subtitle using a custom separator*

```
:title-separator: {sp}|  
= The Dangerous and Thrilling Documentation Chronicles | A Tale of Caffeine and Words  
It began on a bleary Monday morning.
```

Note that a space is always appended to the value of the `title-separator` (making the default value of the `title-separator` effectively a single colon).

Asciidoctor also provides an API for extracting the title and subtitle. See the API docs for the [Document::Title](#) for more information. Support for subtitle functionality for other sections is being considered. Refer to [issue #1493](#).

### 14.1.3. Document title visibility

You can control whether or not the document title appears in the converted document using the `showtitle` attribute.

When converting a standalone document, the document title is shown by default. If you don't want the title to be shown in this case, unset the `showtitle` attribute using `showtitle!` in the document header or via the CLI or API.

When converted to an embeddable document, the document title is *not* shown by default. If you want the title to be shown, set the `showtitle` attribute in the document header or via the CLI or API. The author and revision information is not shown below the document title in the embeddable version of the document like it is in the standalone document, even when the `showtitle` attribute is set.

Let's look at how to add additional metadata to the document header, including an author and her email address.

## 14.2. Author and Email

The author of a document is listed on the line beneath the document's title. An optional email address or URL can follow an author's name inside angle brackets.

Let's add an author with her email address to the document below.

```
= The Dangerous and Thrilling Documentation Chronicles  
Kismet Rainbow Chameleon <kismet@asciidoctor.org>
```

This journey begins...

== About the Author

You can contact {author} at {email}.  
{firstname} loves to hear from other chroniclers.

P.S. And yes, my middle name really is {middlename}.  
What else would you expect from a member of the Rocky Mountain {lastname} Clan?

{authorinitials}

*Result: Rendered author and email information displayed on the byline and referenced in the document's body*

# The Dangerous and Thrilling Documentation Chronicles

Kismet Rainbow Chameleon – [kismet@asciidoctor.org](mailto:kismet@asciidoctor.org)

This journey begins...

## About the Author

You can contact Kismet Rainbow Chameleon at [kismet@asciidoctor.org](mailto:kismet@asciidoctor.org). Kismet loves to hear from other chroniclers.

P.S. And yes, my middle name really is Rainbow. What else would you expect from a member of the Rocky Mountain Chameleon Clan?

KRC

As you can see in the example above, Asciidoctor uses the author's name and email to assign values to a number of built-in attributes that can be used throughout the document's body. These attributes include:

### **author**

The author's full name, which includes all of the characters or words prior to a semicolon (;), angle bracket (<) or the end of the line.

### **firstname**

The first word in the author attribute.

### **lastname**

The last word in the author attribute.

### **middlename**

If a firstname and lastname are present, any remaining words or characters found between these attributes are assigned to the middlename attribute.

### **authorinitials**

The first character of the firstname, middlename, and lastname attributes.

### **email**

An email address, delimited by angle brackets (<>).

If one or more of the author's names consists of more than one word, use an underscore (\_) between the words you want to adjoin. For example, the author of the following document has a compound last name.

= The Unbearable Lightness of Nomenclature

Jan Hendrik van\_den\_Berg

My first name is {firstname}.

My middle name is {middlename}.

My last name is {lastname}.

My initials are {authorinitials}.

*Result: Rendered author information when author has a compound name*

# The Unbearable Lightness of Nomenclature

Jan Hendrik van den Berg

My first name is Jan.

My middle name is Hendrik.

My last name is van den Berg.

My initials are JHv.

Alternatively, the author and email attributes can be set explicitly in the header.

= The Dangerous and Thrilling Documentation Chronicles

:author: Kismet Rainbow Chameleon

:email: kismet@asciidoc.org

This journey begins...

== About the Author

You can contact {author} at {email}.

{firstname} loves to hear from other chroniclers.

P.S. And yes, my middle name really is {middlename}.

What else would you expect from a member of the Rocky Mountain {lastname} Clan?

{authorinitials}

# The Dangerous and Thrilling Documentation Chronicles

Kismet Rainbow Chameleon – [kismet@asciidocdoctor.org](mailto:kismet@asciidocdoctor.org)

This journey begins...

## About the Author

You can contact Kismet Rainbow Chameleon at [kismet@asciidocdoctor.org](mailto:kismet@asciidocdoctor.org). Kismet loves to hear from other chroniclers.

P.S. And yes, my middle name really is Rainbow. What else would you expect from a member of the Rocky Mountain Chameleon Clan?

KRC

The `html5` and `docbook` converters can convert documents with multiple authors. Multiple authors and their emails are separated by semicolons (`;`) when they're listed on the same line.

```
= The Dangerous and Thrilling Documentation Chronicles
Kismet Rainbow Chameleon <kismet@asciidocdoctor.org>; Lazarus het_Draeke
<lazarus@asciidocdoctor.org>
```

This journey begins...

-- About the Authors

You can contact {author} at {email}.  
{firstname} loves to hear from other chroniclers.

{author\_2} specializes in burning down automation obstacles. ①  
Email {lastname\_2} at {email\_2}.

Until our next adventure!

{authorinitials} & {authorinitials\_2}

① To reference the additional authors in the document body, the author attributes are appended with an underscore (`_`) followed by the position of the author in the author information list (i.e. Lazarus het Draeke is the second author in the list so his author attributes are appended with a 2).

# The Dangerous and Thrilling Documentation Chronicles

Kismet Rainbow Chameleon – [kismet@asciidocdoctor.org](mailto:kismet@asciidocdoctor.org), Lazarus het Draeke – [lazarus@asciidocdoctor.org](mailto:lazarus@asciidocdoctor.org)

This journey begins...

## About the Authors

You can contact Kismet Rainbow Chameleon at [kismet@asciidocdoctor.org](mailto:kismet@asciidocdoctor.org). Kismet loves to hear from other chroniclers.

Lazarus het Draeke specializes in burning down automation obstacles. Email Draeke at [lazarus@asciidocdoctor.org](mailto:lazarus@asciidocdoctor.org).

Until our next adventure!

KRC & LhD

### 14.2.1. Attribute references in the author line

The implicit author line was not intended to support arbitrary placement of attribute references. While attribute references are replaced in the author line (as part of the header substitution group), they aren't substituted until *after* the line is parsed. This ordering can sometimes produce undesirable or surprising results. It's best to use the author line strictly as a shorthand for defining a fixed author and email.

If you do need to use attribute references in the author or email value, you should revert to defining the attributes explicitly using attribute entries.

*Using attribute references to set author and email*

```
= Document Title  
:author_name: ACME Industries  
:author_email: info@acme.com  
:author: {author_name}  
:email: {author_email}
```

Just remember that the author line is for static text. Once you graduate beyond static text, you should switch to using attribute entries to define the built-in author attributes, which will give you much more power.

## 14.3. Revision Number, Date and Remark

The revision information is read from the third (non-attribute) line of the header, beneath the author information line. A document's revision information has slots for three implicit attributes.

### revnumber

The document's version number. In order to be recognized, the version number must contain at least one numeric character. Any letters or symbols preceding the numeric character are ignored. If the implicit `revnumber` is on a line by itself, it must begin with the "v" character (e.g., `v1.0`) or be followed by a comma (e.g., `1.0,`) If the implicit `revdate` is present, it must be separated from the `revnumber` by a comma (e.g., `2020-10-10, v78.1`).

### revdate

The date the document version was completed. When the `revnumber` or `revremark` attributes are set, but `revdate` is not, then `revdate` will be assigned the `docdate` value.

### revremark

Information about this version of the document.

Here's an example of a revision line.

```
= The Dangerous and Thrilling Documentation Chronicles  
Kismet Chameleon <kismet@asciidoc.org>  
v1.0, October 2, 2013: First incarnation // <1> ② ③
```

This journey begins...

-- Colophon

Version: {revnumber}

Version Date: {revdate}

Version Notes: {revremark}

① revnumber and revdate must be separated by a comma (,).

② revdate can contain words, letters, numbers, and symbols.

③ The revremark attribute must be preceded by a colon (:), regardless of whether revnumber or revdate are set.

*Result: Rendered revision information displayed on the byline and referenced in the document's body*

# The Dangerous and Thrilling Documentation Chronicles

Kismet Chameleon – [kismet@asciidocdoctor.org](mailto:kismet@asciidocdoctor.org) – Version 1.0, October 2, 2013 – First incarnation

This journey begins...

## Colophon

Version: 1.0

Version Date: October 2, 2013

Version Notes: First incarnation

The revnumber in the byline is prefixed by the word *Version*; however, when referenced in the body of the document, only the numerical value is displayed. The **version-label** attribute controls the version number label in the byline. The revision information attributes can also be explicitly set in the header.

```
= The Dangerous and Thrilling Documentation Chronicles
Kismet Chameleon <kismet@asciidocdoctor.org>
:revnumber: 1.0 ①
:revdate: 10-02-2013
:revremark: The first incarnation of {doctitle} ②
:version-label!: ③
```

This journey begins...

-- Colophon

Version: {revnumber}

Version Date: {revdate}

Version Notes: {revremark}

① When explicitly set, any characters preceding the version number are **not** dropped.

② The revremark can contain attribute references.

③ The version-label attribute is unset so that the word *Version* does not precede the revnumber in the byline.

# The Dangerous and Thrilling Documentation Chronicles

Kismet Chameleon – [kismet@asciidocdoctor.org](mailto:kismet@asciidocdoctor.org) – V1.0, 10-02-2013 – The first incarnation of The Dangerous and Thrilling Documentation Chronicles

This journey begins...

## Colophon

Version: v1.0

Version Date: 10-02-2013

Version Notes: The first incarnation of The Dangerous and Thrilling Documentation Chronicles

In the converted document, notice that the V preceding the revnumber is capitalized in the byline but not when the attribute is referenced in the body of the document.



Revision extraction information and an extraction example are pending.

## 14.4. Subtitle Partitioning

By default, the document title is separated into a main title and subtitle using the industry standard, a colon followed by a space.



As of Asciidoctor 1.5.2, subtitle partitioning is not implemented in the HTML 5 backend.

*A document title that contains a subtitle*

= Main Title: Subtitle

The separator is searched from the end of the text. Therefore, only the last occurrence of the separator is used for partitioning the title.

*A document title that contains a subtitle and more than one separator*

= Main Title: Main Title Continued: Subtitle

You can modify the title separator by specifying the **separator** block attribute explicitly above the document title (since Asciidoctor 1.5.3). Note that a space will automatically be appended to the separator value.

### *A document title with an explicit title separator*

```
[separator=:::]  
= Main Title:: Subtitle
```

You can also set the separator using a document attribute, either in the document:

### *A document title with an explicit title separator*

```
= Main Title:: Subtitle  
:title-separator: ::
```

or from the API or CLI (shown here):

```
$ asciidoctor -a title-separator=::: document.adoc
```

You can partition the title from the API when calling the `doctitle` method on Document:

#### *Retrieving a partitioned document title*

```
title_parts = document.doctitle partition: true  
puts title_parts.title  
puts title_parts.subtitle
```

You can partition the title in an arbitrary way by passing the separator as a value to the partition option. In this case, the partition option both activates subtitle partitioning and passes in a custom separator.

#### *Retrieving a partitioned document title with a custom separator*

```
title_parts = document.doctitle partition: '::'  
puts title_parts.title  
puts title_parts.subtitle
```

## 14.5. Metadata

Document metadata, such as a description of the document, keywords, and the title, can be assigned to attributes in the header. When converted to HTML, the values of these attributes will correspond to tags contained in the `<head>` section of an HTML document.

### 14.5.1. Description

You can include a description of the document using the `description` attribute.

= The Dangerous and Thrilling Documentation Chronicles  
Kismet Rainbow Chameleon <kismet@asciidoc.org>; Lazarus het\_Draeke  
<lazarus@asciidoc.org>  
:description: A story chronicling the inexplicable hazards and vicious beasts a \ ① documentation team must surmount and vanquish on their journey to finding an \ open source project's true power.

This journey begins on a bleary Monday morning.

- ① If the document's description is long, you can break the attribute's value across several lines by ending each line with a backslash \ that is preceded by a space.

When converted to HTML, the document description value is assigned to the HTML <meta> tag.

*HTML output*

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<meta name="generator" content="Asciidoctor 1.5.6.2">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="A story chronicling the inexplicable hazards and vicious beasts a documentation team must surmount and vanquish on their journey to finding an open source project's true power.">
<title>The Dangerous and Thrilling Documentation Chronicles</title>
<style>
```

## 14.5.2. Keywords

The **keywords** attribute contains a list of comma separated values that are assigned to the HTML <meta> tag.

= The Dangerous and Thrilling Documentation Chronicles  
Kismet Rainbow Chameleon <kismet@asciidoc.org>; Lazarus het\_Draeke  
<lazarus@asciidoc.org>  
:keywords: documentation, team, obstacles, journey, victory

This journey begins on a bleary Monday morning.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<meta name="generator" content="Asciidoctor 1.5.6.1">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="keywords" content="documentation, team, obstacles, journey, victory">
<title>The Dangerous and Thrilling Documentation Chronicles</title>
<style>
```

### 14.5.3. Alternate Title

By default, Asciidoctor uses the text of the document title as the value of the HTML `<title>` tag. You can override this behavior by setting the `title` attribute in the document header.

### 14.5.4. Custom Metadata, Styles and Functions

You can add content, such as custom metadata, stylesheet, and script information, to the header of the output document using docinfo (“document information”) files. The [docinfo file](#) section details what these files can contain and how to use them.

## 14.6. Header Summary

*Header attributes and values*

Attribute	Values	Description	Notes	Converters
<code>author</code>		Author's full name		all
<code>authorinitials</code>		First character of each word in the <code>author</code> attribute		all
<code>description</code>		Text describing the document		html
<code>docinfo</code> , <code>docinfo1</code> , <code>docinfo2</code>		Adds content from a docinfo file to header		html, docbook
<code>doctitle</code>	Text entered by user	Title of document	Identical to the value returned by <code>Document#doctitle</code>	all
<code>email</code>		Email address		all
<code>firstname</code>		First word of <code>author</code> attribute		all

Attribute	Values	Description	Notes	Converters
<code>keywords</code>		A list of comma-separated values that describe the document		html
<code>lastname</code>		Last word of <code>author</code> attribute		all
<code>middlename</code>		Middle word of <code>author</code> attribute		all
<code>no-header-footer, -s</code>		Generate an embeddable document; exclude the document frame		all
<code>noheader</code>		Suppresses the output of the header		all
<code>nofooter</code>		Suppresses the output of the footer		all
<code>notitle</code>		Toggles the display of a document's title		all
<code>revdate</code>		Date of document version		all
<code>revnumber</code>		Version number of the document		all
<code>revremark</code>		Version comments		all
<code>showtitle</code>		Toggles the display of an embedded document's title		all
<code>title</code>		Alternative title of the document		html
<code>version-label</code>	<code>Version</code> , User defined	The label preceding the <code>revnumber</code> in a converted document's byline		html

# Chapter 15. Preamble

Content between the document header and the first section title is called the preamble. The preamble is optional.

## *Preamble*

```
= The Dangerous and Thrilling Documentation Chronicles  
Kismet Chameleon
```

This journey begins on a bleary Monday morning.

Our intrepid team is in desperate need of double shot mochas, but the milk expired eight days ago.

A trip to the dairy was out of the question.

On Friday night, a mutant, script-injecting warlock had infected the Shetland cattle herd with a ravenous craving for tags and annotations.

The security wolves were at a trust building retreat in Katchanga, and no one in the village could locate their defensive operations manual.

Weak daylight trickled across the stripped pasture, chased by distant bovine screams...

```
-- Cavern Glow
```

The river rages through the cavern, rattling its content...

When using the default Asciidoctor stylesheet, the `lead` attribute is applied to the first paragraph of the preamble.

# The Dangerous and Thrilling Documentation Chronicles

Kismet Chameleon

This journey begins on a bleary Monday morning. Our intrepid team is in desperate need of double shot mochas, but the milk expired eight days ago. A trip to the dairy was out of the question. On Friday night, a mutant, script-injecting warlock had infected the Shetland cattle herd with a ravenous craving for tags and annotations. The security wolves were at a trust building retreat in Katchanga, and no one in the village could locate their defensive operations manual.

Weak daylight trickled across the stripped pasture, chased by distant bovine screams...

## Cavern Glow

The river rages through the cavern, rattling its content...

# Chapter 16. Sections

Sections partition the document into a content hierarchy. In AsciiDoc, sections are defined using section titles.

A section title represents the heading for a section. Section title levels are specified by two to six equal signs. The number of equal signs in front of the title represents the nesting level (using a 0-based index) of the section.

*Section titles available in an article doctype*

```
= Document Title (Level 0)
```

```
== Level 1 Section Title
```

```
==== Level 2 Section Title
```

```
===== Level 3 Section Title
```

```
===== Level 4 Section Title
```

```
===== Level 5 Section Title
```

```
== Another Level 1 Section Title
```

*Render section titles*

# Document Title (Level 0)

## Level 1 Section Title

### Level 2 Section Title

#### Level 3 Section Title

#### Level 4 Section Title

#### Level 5 Section Title

## Another Level 1 Section Title



In addition to the equals sign marker used for defining single-line section titles, Asciidoctor recognizes the hash symbol (#) from Markdown. That means the outline of a Markdown document will be converted just fine as an AsciiDoc document.

Section levels must be nested logically. There are two rules you must follow:

1. A document can only have multiple level 0 sections if the `doctype` is set to `book`.
  - The first level 0 section is the document title; subsequent level 0 sections represent parts.
2. Section levels cannot be skipped when nesting sections (e.g., you can't nest a level 5 section directly inside a level 3 section; an intermediary level 4 section is required).

For example, the following syntax is illegal:

```
= Document Title  
  
= Illegal Level 0 Section (violates rule #1)  
  
== First Section  
  
==== Illegal Nested Section (violates rule #2)
```

Content above the first section title is designated as the document's preamble. Once the first section title is reached, content is associated with the section it is nested in.

```
-- First Section  
  
Content of first section  
  
==== Nested Section  
  
Content of nested section  
  
-- Second Section  
  
Content of second section
```

## 16.1. Titles as HTML Headings

When the document is converted to HTML 5 (using the built-in `html5` backend), each section title becomes a heading element where the heading level matches the number of equal signs. For example, a level 1 section (`--`) maps to an `<h2>` element.

## 16.2. Auto-generated IDs

If you don't specify a custom ID for a section, the processor automatically generates an ID from the section's title and assigns it to the section by default.

The ID is derived from the section title as follows:

- Inline formatting is applied (in title substitution order).
- All characters are converted to lowercase.
- The value of the `idprefix` attribute (`_` by default) is prepended.
- Character references, HTML/XML tags, and invalid ID characters are removed.
  - Refer to the [NT-Name](#) section of the XML specification for a list of valid ID characters.
  - Prior to 1.5.7, HTML/XML tags were not removed and character references and invalid ID characters were replaced with the value of the `idseparator` attribute (`_` by default).
- Spaces, hyphens, and periods are replaced with the value of the `idseparator` attribute (`_` by default)
- Repeating separator characters are condensed.
- If necessary, a sequence number is appended until the ID is unique within the document.

For example, the section title `Wiley & Sons, Inc.` produces the ID `_wiley_sons_inc`. You can customize the replacement characters or toggle ID auto-generation using document attributes.

You can change the ID prefix by setting the `idprefix` attribute. The value of the `idprefix` attribute must begin with a valid ID start character and can have any number of additional valid ID characters.

```
:idprefix: id_
```

If you want to remove the prefix, set the attribute to an empty value.

```
:idprefix:
```



If you set the `idprefix` to blank, you could end up generating IDs that are invalid in DocBook output (e.g., an ID that begins with a number) or that match a built-in ID in the HTML output (e.g., `header`). In this case, we recommend either using a non-empty value of `idprefix` or assigning explicit IDs to your sections.

You can change the word separator using the `idseparator` attribute. Unless empty, the value of the `idseparator` must be *exactly one valid ID character*.

```
:idseparator: -
```

If you don't want to use a separator, set the attribute to an empty value.

```
:idseparator:
```



When a document is rendered on GitHub, the `idprefix` is set to an empty value and the `idseparator` is set to `-`. These settings are used to ensure that the IDs generated by GitHub match the IDs generated by Asciidoctor.

To disable the auto-generation of section IDs, unset the `sectids` attribute.

```
:sectids!:
```

Custom IDs, covered next, are used even if auto-generation of section IDs is disabled.



If you disable auto-generated section IDs, and you don't assign a custom ID to a section, you won't be able to link to (i.e., reference) that section.

## 16.3. Custom IDs

You can assign a custom ID and optional reference text (i.e., label) to a section (see [Defining an Anchor](#)). The custom ID is used in place of the auto-generated ID. This can be useful when you want to define a stable anchor for linking to a section using a cross reference. The reference text is used when referencing the ID without specifying explicit text. This is a good way to centrally manage the automatic reference text that is used to refer to a section.

Here's an example of a section with a custom ID:

```
[#tiger-subspecies]
===
Subspecies of Tiger
```

Here's an example of a section with a custom ID and reference text:

```
[[tiger-subspecies,Tigers]]
===
Subspecies of Tiger
```

The ID can be written using the following shorthand (though there's no shorthand yet for the reference text):

```
[#tiger-subspecies,reftext=Tigers]
===
Subspecies of Tiger
```

The value of the `reftext` attribute must be quoted if it contains spaces or commas.

Asciidoctor recognizes all valid UTF-8 characters in section IDs. When using the double square bracket form, the ID must conform to the [XML Name](#) rules, which means the ID must start with a letter, an underscore, or a colon. If you're generating a PDF from AsciiDoc using a2x and dblatex, see [Using UTF-8 titles with a2x](#) to learn about the required `latex.encoding=utf8` switch to activate this portability.



## 16.4. Multiple Anchors

While a section title can only have a single primary ID, as of Asciidoctor 1.5.7, it's possible to register additional anchor points on a section title using inline anchors. This feature works regardless of whether you assign a custom ID.

Here's how to register additional anchor points when using an auto-generated ID:

*Register additional leading anchor points*

```
== [[secondary-id]][[tertiary-id]]Section Title
```

*Register additional trailing anchor points*

```
== Section Title[[secondary-id]][[tertiary-id]]
```

You can place the anchors anywhere you want within the section title line. Where you place them is where the anchor will end up in the output, so it's best to put them either at the beginning or the end. The beginning is the preferred location.

These additional anchor points don't interfere with defining a custom ID.

*Register additional leading anchor points on a section title with a custom ID*

```
[#primary-id]
== [[secondary-id]][[tertiary-id]]Section Title
```

Alternately, you can move the primary ID inline by placing it after the section title, offset by a single space.

*Set the custom ID within the section title*

```
== Section Title[[secondary-id]][[tertiary-id]] [[primary-id]]
```

However, we don't recommend defining the primary ID inline as it's more difficult to read.

## 16.5. Links

To turn section titles into links, enable the `sectlinks` attribute. The default Asciidoctor stylesheet displays linked section titles with the same color and styles as unlinked section titles.

## 16.6. Anchors

When the `<code>sectanchors</code>` attribute is enabled on a document, an anchor (empty link) is added before the section title. The default Asciidoctor stylesheet renders the anchor as a section entity (`<code>&sect;</code>`) that floats to the left of the section title.

## 16.7. Numbering

Asciidoctor allows section numbering to be toggled on and off throughout a document. You can enable section numbers using the `sectnums` attribute.

```
:sectnums:
```



Asciidoctor still supports the attribute name `numbered` to number sections for backward compatibility with AsciiDoc Python, but the name `sectnums` is preferred.

You can also use this attribute entry above any section title in the document to toggle the auto-numbering setting. When you want to turn off the numbering, add an exclamation point to the end of the attribute name:

```
:sectnums!:
```

```
== Unnumbered Section
```

For regions of the document where section numbering is turned off, the section numbering will not be incremented.

Given:

```
= Document Title  
  
:sectnums!:  
  
== Colophon Section  
  
== Another Colophon Section  
  
== Last Colophon Section  
  
:sectnums:  
  
== Section One  
  
== Section Two  
  
== Section Three
```

The sections will be numbered as follows:

```
Colophon Section  
  
Another Colophon Section  
  
Last Colophon Section  
  
1. Section One  
  
2. Section Two  
  
3. Section Three
```

Asciidoctor will always curtail incrementing the section number in regions of the document where section numbers are off.

If `sectnums` is set on the command line (or API), that overrides the value set in the document header, but it does not prevent the document from toggling the value for regions of the document.

If `sectnums!` is set on the command line (or API), then the numbers are disabled regardless of the setting within the document.

#### *Flexible attributes*



The `sectnums` attribute is referred to as a “flexible attribute”, which means it can always be changed midstream in the document, even if it is enabled through the API or CLI. At the time of writing, `sectnums` is unique in this way, though other attributes may receive this special status in the future.

### 16.7.1. Numbering depth

You can restrict section numbering depth by defining the `sectnumlevels` (document header-only) attribute.

```
:sectnumlevels: 2 ①
```

- ① When the `sectnumlevels` attribute is assigned a value of `2`, section titles with levels 3 through 5 are not numbered (i.e., not prefixed with a number).

When the `doctype` is `book`, recall that level-1 sections become chapters. Therefore, a `sectnumlevels` of `4` translates to 3 levels of numbered sections inside each chapter.

Assigning `sectnumlevels` a value of `0` is effectively the same as disabling section numbering (i.e., `sectnums!`).

## 16.8. Discrete Headings (aka Floating Titles)

To make a discrete heading, you add the `discrete` attribute to any `section title`. A discrete heading is styled in a manner similar to a section title, but is not part of the section hierarchy (i.e., and thus excluded from section nesting requirements), it does not enclose subsequent blocks, and it is not included in the table of contents. The `discrete` style effectively demotes the section title to a normal heading.



Alternately, you may use the `float` attribute to identify a discrete heading. In this context, the term “float” does not imply layout, meaning it does not float to the left or right of other content blocks. It just means it’s not anchored to the section hierarchy.

Here's an example of a discrete heading in use.

```
**** ①
```

Discrete headings are useful for making headings inside of other blocks, like this sidebar.

```
[discrete] ②
```

```
== Discrete Heading ③
```

Discrete headings can be used where sections are not permitted.

```
****
```

① A delimiter line that indicates the start of a sidebar block.

② Set the `discrete` attribute above the section title to demote it to a discrete heading.

③ The discrete heading title is designated by one to six equal signs, just like a section title.

## 16.9. Section Styles

Asciidoctor provides styles for the frontmatter and backmatter sections commonly found in journal articles, academic papers, and books. The styles are:

- colophon
- abstract
- preface
- dedication
- part introduction
- appendix
- glossary
- bibliography
- index

These styles are available to the `article` and `book` document types, with the exception of the part introduction style which is exclusive to books.

In general, the section style attribute is set above a level 1 section title or block of text. For instance, the example below shows how to label a section as an abstract.

```
[abstract]
== Documentation Abstract

Documentation is a distillation of many long, squiggly adventures.
```

The structure and usage rules for each section style is explained in [Structuring, Navigating, and Referencing Your Content](#).

## 16.10. Sections Summary

*Section attributes and values*

Attribute	Value(s)	Example Syntax	Comments
<code>sectids</code>	n/a	<code>:sectids!: </code>	Set to auto-generate section IDs by default.
<code>idprefix</code>	<code>_</code> , or user defined text	<code>:idprefix: id_ </code>	Set to prepend string to generated section ID.
<code>idseparator</code>	<code>_</code> , or user defined character	<code>:idseparator: - </code>	Set to insert character between words in generated section ID.
<code>sectanchors</code>	n/a	<code>:sectanchors: </code>	Asciidoctor only
<code>sectlinks</code>	n/a	<code>:sectlinks: </code>	Asciidoctor only

Attribute	Value(s)	Example Syntax	Comments
<code>sectnums</code>	n/a	<code>:sectnums:</code>	Section numbering is off by default. Can be toggled on or off in the flow of the document.
<code>sectnumlevels</code>	0-5	<code>:sectnumlevels: 3</code>	Set to autogenerated section numbers up to level three by default. Setting a value of <code>0</code> has the same effect as setting <code>sectnums!</code> .

# Chapter 17. Blocks



Section Pending

## 17.1. Title

You can assign a title to any paragraph, list, delimited block, or block macro. In most cases, the title is displayed immediately above the content. If the content is a figure or image, the title is displayed below the content.

A block title is defined on a line above the element. The line must begin with a dot (.) and be followed immediately by the title text.

Here's an example of a list with a title:

```
.TODO list
- Learn the AsciiDoc syntax
- Install Asciidoctor
- Write my document
```

## 17.2. Metadata

In addition to a title, a block can be assigned additional metadata including:

- Id (i.e., anchor)
- Block name (first positional attribute)
- Block attributes

Here's an example of a quote block with metadata:

```
.Gettysburg Address ①
[#gettysburg] ②
[quote, Abraham Lincoln, Address delivered at the dedication of the Cemetery at
Gettysburg] // <3> ④ ⑤
```

----  
Four score and seven years ago our fathers brought forth  
on this continent a new nation...

Now we are engaged in a great civil war, testing whether  
that nation, or any nation so conceived and so dedicated,  
can long endure. ...

① Title: Gettysburg Address

② ID: gettysburg, see [Defining an Anchor](#)

- ③ Block name: quote
- ④ attribution: Abraham Lincoln (Named block attribute)
- ⑤ citetitle: Address delivered at the dedication of the Cemetery at Gettysburg (Named block attribute)



A block can have multiple block attribute lines. The attributes will be aggregated. If there is a name conflict, the last attribute defined wins.

Some metadata is used as supplementary content, such as the title, whereas other metadata controls how the block is converted, such as the block name.

## 17.3. Delimited blocks

The AsciiDoc syntax provides a set of components for including non-paragraph text, such as block quotes, source code listings, sidebars and tables, in your document. These components are referred to as *delimited blocks* because they are surrounded by delimiter lines (e.g., \*\*\*\*).

Here's an example of a sidebar block:

```
****  
sidebar block  
****
```

Within the boundaries of a delimited block, you can enter any content or blank lines. The block doesn't end until the ending delimiter is found. The delimiters around the block determine the type of block, how the content is processed and converted, and what elements are used to wrap the content in the output.

### 17.3.1. Delimiter lines

The boundaries of a delimited block *must be balanced*. In other words, the opening delimiter line must be the same length as the closing delimiter line.

For example, the following delimited block is not balanced and therefore invalid:

```
*****  
invalid sidebar block  
****
```

When the processor encounters the previous example, it will put the remainder of the content in the document inside the delimited block (without warning, currently). As far as the processor is concerned, the closing delimiter is just a line of content. If you want the closing delimiter to be matched, it must be the same length as the opening delimiter.

```
****  
valid sidebar block  
****
```

The AsciiDoc Python processor did not require the delimiters to be balanced, but also never documented that this was permissible. We view AsciiDoc Python's behavior of matching unbalanced delimited blocks to be a bug and therefore do not allow it in Asciidoc.

### 17.3.2. Optional delimiters

If the content is contiguous (not interrupted by blank lines), you can forgo the use of the block delimiters and instead use the block name above a paragraph to repurpose it as one of the delimited block types.

This format is often used for single-line listings:

```
[listing]  
sudo dnf install asciidoc
```

or single-line quotes:

```
[quote]  
Never do today what you can put off ''til tomorrow.
```

### 17.3.3. Masquerading blocks

Some blocks can masquerade as other blocks, a feature which is controlled by the block name.

### 17.3.4. Nesting blocks

You can nest blocks inside each other. To nest blocks of the same type, add a pair of extra symbols to the delimiter. For example:

```
=====  
This is an example  
======  
This is an example inside an example  
======  
====
```

## 17.4. Built-in blocks summary

The following table identifies the built-in block names and delimited blocks syntax, their purposes, and the substitutions performed on their contents.

Block	Block Name	Delimiter	Purpose	Substitution s
Admonition	[<LABEL>]	====	Aside content that demands special attention; often labeled with a tag or icon	Normal
Comment	N/A	///	Private notes that are not displayed in the output	None
Example	[example]	====	Designates example content or defines an admonition block	Normal
Fenced	N/A	```	Source code or keyboard input is displayed as entered	Verbatim
Listing	[listing]	----	Source code or keyboard input is displayed as entered	Verbatim
Literal	[literal]	....	Output text is displayed exactly as entered	Verbatim
Open	Most block names	--	Anonymous block that can act as any block except <i>passthrough</i> or <i>table</i> blocks	Varies
Passthrough	[pass]	****	Unprocessed content that is sent directly to the output	None
Quote	[quote]	----	A quotation with optional attribution	Normal
Sidebar	[sidebar]	****	Aside text and content displayed outside the flow of the document	Normal
Source	[source]	----	Source code or keyboard input to be displayed as entered	Verbatim
Stem	[stem]	****	Unprocessed content that is sent directly to an interpreter (such as AsciiMath or LaTeX math)	None
Table	N/A	==	Displays tabular content	Varies
Verse	[verse]	----	A verse with optional attribution	Normal

This table shows the substitutions performed by each substitution group referenced in the previous table.

#### Substitution groups

Group	Special characters	Quotes	Attributes	Replacements	Macros	Post replacements
Header	✓	✗	✓	✗	✗	✗
None	✗	✗	✗	✗	✗	✗
Normal	✓	✓	✓	✓	✓	✓
Pass	✗	✗	✗	✗	✗	✗

Group	Special characters	Quotes	Attributes	Replacements	Macros	Post replacements
Verbatim	✓	✗	✗	✗	✗	✗



Surround an attribute value with single quotes in order to apply normal substitutions.

# Chapter 18. Paragraph

The bulk of the content in a document is paragraph text. This is why Asciidoctor doesn't require any special markup or attributes to specify paragraph content. You can just start typing.

In Asciidoctor, adjacent or consecutive lines of text form a paragraph element. To start a new paragraph after another element, such as a section title or table, hit the **RETURN** key twice to insert a blank line, and then continue typing your content.

*Two paragraphs in an AsciiDoc document*

```
This journey begins one late Monday afternoon in Antwerp.  
Our team desperately needs coffee, but none of us dare open the office door.  
  
To leave means code dismemberment and certain death.
```

*The two paragraphs rendered using the default (html5) converter and stylesheet (asciidoc.css)*

```
This journey begins one late Monday afternoon in Antwerp. Our team desperately needs  
coffee, but none of us dare open the office door.  
  
To leave means code dismemberment and certain death.
```

## 18.1. Alignment

Asciidoctor provides built-in roles to align the text of a paragraph. The name of the role follows the pattern **text-<alignment>**, where **<alignment>** is one of left, center, right, or justify (e.g., **text-center**).

Here's an example of a paragraph with center-aligned text.

```
[.text-center]  
This text is centered.
```

These built-in text alignment roles are currently supported by the HTML5 and PDF converters.

## 18.2. Line Breaks

Since adjacent lines of text are combined into a single paragraph when Asciidoctor converts a document, that means you can wrap paragraph text or put each sentence or phrase on a separate line. The line breaks won't appear in the output.

However, if you want the line breaks in a paragraph to be preserved, you can either use a space followed by a plus sign (+) or set the **hardbreaks** option on the paragraph. This results in a visible line break (e.g., <br>) following each line.

*Line breaks preserved using a space followed by the plus sign (+)*

```
Rubies are red, +
Topazes are blue.
```

```
Rubies are red,
Topazes are blue.
```

*Line breaks preserved using the hardbreaks option*

```
[%hardbreaks]
Ruby is red.
Java is black.
```

```
Ruby is red.
Java is black.
```

To preserve line breaks throughout your whole document, add the `hardbreaks` attribute to the document's header.

*Line breaks preserved throughout the document using the hardbreaks attribute*

```
= Line Break Doc Title
:hardbreaks:
```

```
Rubies are red,
Topazes are blue.
```

You can also preserve line breaks using [literal blocks](#), [listing blocks](#), and [verses](#).

## 18.3. Lead Style

Apply the `lead` style to any paragraph, and it will render using a larger font size. The lead style is assigned to the `role` attribute. You can set `role` using the long- or shorthand method.

*Setting role using shorthand (.) and assigning it the lead value*

```
[.lead]
This is the ultimate paragraph.
```

*Rendered lead paragraph*

```
This is the ultimate paragraph.
```

When you convert a document to HTML using the default stylesheet, the first paragraph of the preamble is automatically styled as a lead paragraph. To disable this behavior, assign any role to the first paragraph.

*Disabling the automatic lead paragraph styling*

```
[ .normal]
```

```
This is a normal paragraph, regardless of its position in the document.
```

The presence of the custom role tells the CSS not to style it as a lead paragraph.

# Chapter 19. Text Formatting

Just as we emphasize certain words and phrases when we speak, we can emphasize them in text with formatting. This formatting, such as bold or monospace, is indicated by surrounding letters, words, or phrases with simple markup. When Asciidoctor processes text enclosed by formatting markup, the markup is replaced by the corresponding HTML or XML tags, depending on your backend, during the [quotes substitution phase](#).

## Text formatting versus quotes

The original AsciiDoc processor calls this markup and its related features *quotes*, *quotes substitutions*, and *quoted text*. However, in the Asciidoctor documentation, we refer to the text formatting markup as, you guessed it, text formatting. When you need to control when text formatting is applied to certain blocks, the substitution attribute is still named **quotes**, but we're considering changing this to a more semantic name in a future Asciidoctor version.

Continue reading to learn how to format letters, words or phrases with the following styles:

- bold
- italic
- curved (smart) quotation marks and apostrophes
- subscript and superscript
- monospace
- highlighted, built-in and custom CSS styles



You may not always want these symbols to indicate text formatting. In those cases, you'll need to use additional markup to [escape the text formatting markup](#).

## 19.1. Bold and Italic

The two most common ways of emphasizing a word is to format it as bold or italic. To apply bold styling to a word or phrase, place an asterisk (\*) at the beginning and end of the text you wish to format. To bold a letter or letters in a string of text, place two asterisks (\*\*) before and after the letter or letters.

Letters and words are italicized using one (\_) or two (\_\_) underscores. When you want to bold and italicize a letter or word, the bold markup must be the outermost markup.

*Bold and italic text formatting syntax*

```
_To tame_ the wild wolpertingers we needed to build a *charm*.  
But **u**ltimate victory could only be won if we divined the *_true name_* of the  
__war__ lock.
```

*Result: bold and italic text*

*To tame the wild wolpertingers we needed to build a **charm**. But ultimate victory could only be won if we divined the **true name** of the warlock.*

## 19.2. Quotation Marks and Apostrophes

Single and double quotation marks are **not** rendered as curved quotation marks (also known as smart, curly, typographic or book quotation marks). When entered using the `'` and `"` key, Asciidoctor outputs straight (dumb, vertical, typewriter) quotation marks. However, by creating a set of backticks (`) contained within a set of single quotes ('') or double quotes ("`), you can tell Asciidoctor where to output curved quotation marks.

*Single and double curved quotation marks syntax*

`"'What kind of charm?'" Lazarus asked. "'An odoriferous one or a mineral one?'" ①`

`Kizmet shrugged. "'The note from Olaf's desk says ''wormwood and licorice,' but these could be normal groceries for werewolves.'" ②`

- ① To output double curved quotes, enclose a word or phrase in a set of double quotes ("") and a set of backticks (`).
- ② To output single curved quotes, enclose a word or phrase in a set of single quotes ('') and a set of backticks (`). In this example, the phrase *wormwood and licorice* should be enclosed in curved single quotes when the document is rendered.

*Rendered curved quotation marks*

“What kind of charm?” Lazarus asked. “An odoriferous one or a mineral one?”

Kizmet shrugged. “The note from Olaf’s desk says ‘wormwood and licorice,’ but these could be normal groceries for werewolves.”

When entered with the `'` key, Asciidoctor renders an apostrophe that is directly preceded and followed by a character, such as in contractions and possessive singular forms, as a curved apostrophe. This inconsistent handling of apostrophes and quotation marks is a hold over from the original AsciiDoc processor. An apostrophe directly bounded by two characters is processed during the [replacements substitution phase](#), not the [quotes phase](#). This is why an apostrophe directly followed by white space, such as the possessive plural form, is not curved by default.

To render an apostrophe as curved when it is not bound by two characters, mark it as you would a single curved quote.

## *Curved apostrophe syntax*

```
Olaf had been with the company since the ``60s.  
His desk overflowed with heaps of paper, apple cores and squeaky toys.  
We couldn't find Olaf's keyboard.  
The state of his desk was replicated, in triplicate, across all of the werewolves' desks.
```

In the rendered output, note that the plural possessive apostrophe, seen trailing *werewolves*, is curved, as is the omission apostrophe before *60s*.

## *Rendered apostrophes*

```
Olaf had been with the company since the '60s. His desk overflowed with heaps of paper, apple  
cores and squeaky toys. We couldn't find Olaf's keyboard. The state of his desk was replicated,  
in triplicate, across all of the werewolves' desks.
```

If you don't want an apostrophe that is bound by two characters to be rendered as curved, escape it by preceding it with a backslash (\). The [preventing substitutions](#) and [passthrough](#) sections detail additional ways to prevent punctuation substitutions.

```
Olaf\'s desk ...
```

## *Rendered escaped apostrophe*

```
Olaf's desk ...
```

In order to make a possessive, monospaced phrase, you need to switch to unconstrained formatting and use an explicit typographic apostrophe.

```
``npm``'s job is to manage the dependencies for your application.
```

## *Rendered possessive, monospaced phrase*

```
npm's job is to manage the dependencies for your application.
```

The same goes if the monospaced phrase ends in an “s”.

```
This ``class``' static methods make it easy to operate on files and directories.
```

## *Rendered possessive, monospaced phrase that ends in an “s”*

```
This class' static methods make it easy to operate on files and directories.
```

Alternately, you could encode the typographic apostrophe directly in the AsciiDoc source to get the same result without the need to use unconstrained formatting.

The ‘class’ static methods make it easy to operate on files and directories.

## 19.3. Subscript and Superscript

Subscript and superscript text is common in mathematical expressions and chemical formulas. When rendered, the size of subscripted and superscripted text is reduced. Subscripted text is placed at the baseline and superscripted text above the baseline. The size and precise placement of the text depends on the font and other stylesheet parameters applied to the converted document.

Text is subscripted when you enclose it in a set of tildes (~) and superscripted with a set of carets (^)

### Subscript

One tilde (~) on either side of a word or phrase makes it subscript.

### Superscript

One caret (^) on either side of a word or phrase makes it superscript.

“Well the H<sub>2</sub>O formula written on their whiteboard could be part of a shopping list, but I don’t think the local bodega sells E=mc<sup>2</sup>,” Lazarus replied.

“Well the H<sub>2</sub>O formula written on their whiteboard could be part of a shopping list, but I don’t think the local bodega sells E=mc<sup>2</sup>,” Lazarus replied.



You can [write and render equations and formulas](#) in AsciiDoc documents using MathJax.

## 19.4. Monospace

In AsciiDoc, a span of text enclosed in a single set of backtick (`) characters is displayed using a fixed-width (i.e., monospaced) font. Monospace text formatting is typically used to represent text shown in computer terminals or code editors (often referred to as a codespan).

Monospaced text can be bold and italicized, as long as the markup sets are entered in the right order. The monospace markup must be the outermost set, then the bold set, and the italic markup must always be the innermost set.

As with other types of text formatting, if the text is bounded by characters on either side, it must be enclosed in a double set of backtick characters (` `) in order for the formatting to be applied.

Here’s an example:

```
"'Wait!'" Indigo plucked a small vial from her desk's top drawer and held it toward us.  
The vial's label read: 'E=mc^2^'; the '_E_' represents _energy_, but also pure  
_genius!_
```

#### Rendered monospace formatted text

"Wait!" Indigo plucked a small vial from her desk's top drawer and held it toward us. The vial's label read: E=mc<sup>2</sup>; the E represents *energy*, but also pure *genius*!

### 19.4.1. Literal Monospace

Unlike other markup languages, monospaced text in AsciiDoc is not synonymous with literal text. Instead, it gets interpreted just like normal text. In other words, it's subject to text substitutions.

This might be surprising at first. But there's good reason for this difference. In AsciiDoc, you can take advantage of attribute references and inline macros inside of monospaced text span. The drawback, of course, is that you have to be careful to escape these special characters if you intend to output them literally.

One way to prevent the processor from interpreting special characters in monospaced text is to escape them using backslash characters, just as you would with normal text. However, escaping individual occurrences that way can be tedious. That's why AsciiDoc offers a special type of monospace formatting called the literal monospace.

To make a true literal codespan in AsciiDoc, you must enclose the monospaced text in a passthrough. Rather than using a single set of backtick characters, you'll use the combination of the backtick and plus characters, where the plus characters fall on the inside of the backtick characters (e.g., `+text+`).

Here's an example that contains literal, monospaced text.

You can reference the value of a document attribute using the syntax `+{name}+`, where 'name' is the attribute name.

This shorthand syntax can accommodate most of the literal monospace cases. The main exception is when the text itself contains plus characters. To avoid confusing the processor, you'll need to switch to using the more formal passthrough macro to handle these cases.

Here's an example of literal, monospaced text that contains plus characters.

`pass:[++` is the increment operator in C.

Passthroughs are a general purpose utility in AsciiDoc. You can learn about the various passthrough options in [Passthrough Macros](#).

## 19.5. Custom Styling With Attributes

When text is enclosed in a set of single or double hash symbols (#), and no style is assigned to it, the text will be rendered as highlighted text (text wrapped in `<mark>`).

*Highlighted style syntax*

```
Werewolves are #allergic to cinnamon#.
```

*Highlighted text HTML output*

```
<mark>mark element</mark>
```

*Rendered highlighted text*

```
Werewolves are allergic to cinnamon.
```

Additionally, text marked with hash symbols can be assigned built-in styles, such as **big** and **green**.

*Built-in CSS class syntax*

```
Do werewolves believe in [small]#small print#? ①
```

```
[big]##0##nce upon an infinite loop.
```

① The first positional attribute is treated as a role. You can assign it a custom or built-in CSS class.

*Rendered CSS class styled text*

```
Do werewolves believe in small print?
```

```
Once upon an infinite loop.
```

You can format text with custom styles that you define as well.

*Custom CSS class syntax*

```
Type the word [.userinput]#asciidoc# into the search bar.
```

When converted to HTML, the word `asciidoc` is wrapped in `<span>` tags and the role `userinput` is used as the element's CSS class.

*HTML output*

```
<span class="userinput">asciidoc</span>
```

# Chapter 20. Unordered Lists

If you were to create a list in an e-mail, how would you do it? Chances are, you'd mark list items using the same characters that Asciidoc uses to find list items.

In the example below, each list item is marked using an asterisk (\*), the AsciiDoc syntax specifying an unordered list item.

- \* Edgar Allen Poe
- \* Sheri S. Tepper
- \* Bill Bryson

A list item's first line of text must be offset from the marker (\*) by at least one space. If you prefer, you can indent list items. Blank lines are required before and after a list. Additionally, blank lines are permitted, but not required, between list items.

*Rendered unordered list*

- Edgar Allen Poe
- Sheri S. Tepper
- Bill Bryson

You can add a title to a list by prefixing the title with a period (.).

- .Kizmet's Favorite Authors
- \* Edgar Allen Poe
  - \* Sheri S. Tepper
  - \* Bill Bryson

*Rendered unordered list with a title*

- Kizmet's Favorite Authors*
- Edgar Allen Poe
  - Sheri S. Tepper
  - Bill Bryson

Was your instinct to use a hyphen (-) instead of an asterisk to mark list items? Guess what? That works too!

- Edgar Allen Poe
- Sheri S. Tepper
- Bill Bryson

You should reserve the hyphen for lists that only have a single level because the hyphen marker (-) doesn't work for nested lists. Now that we've mentioned nested lists, let's go to the next section and learn how to create lists with multiple levels.

## Separating Lists

If you have adjacent lists, they have the tendency to want to fuse together. To force lists apart, insert a line comment (//) surrounded by blank lines between the two lists. Here's an example, where the - text in the line comment indicates the line serves as an "end of list" marker:

```
* Apples  
* Oranges  
  
//  
  
* Walnuts  
* Almonds
```

## 20.1. Nested

To nest an item, just add another asterisk (\*) to the marker, and another for each subsequent level.

```
.Possible DefOps manual locations  
* West wood maze  
** Maze heart  
*** Reflection pool  
** Secret exit  
* Untracked file in git repository
```

*Rendered nested, unordered list*

```
Possible DefOps manual locations  
• West wood maze  
    • Maze heart  
        • Reflection pool  
    • Secret exit  
• Untracked file in git repository
```

In Asciidoctor 1.5.7 and earlier you could only have up to six (6) levels of nesting (assuming one level uses the hyphen marker).

Since Asciidoctor 1.5.8, you can nest unordered lists to any depth. Keep in mind, however, that

some interfaces will begin flattening lists after a certain depth. GitHub starts flattening list after 10 levels of nesting.

```
* level 1
** level 2
*** level 3
**** level 4
***** level 5
* level 1
```

- level 1
  - level 2
    - level 3
      - level 4
        - level 5

While it would seem as though the number of asterisks represents the nesting level, that's not how depth is determined. A new level is created for each unique marker encountered. However, it's much more intuitive to follow the convention that the number of asterisks equals the level of nesting. After all, we're shooting for plain text markup that is readable *as is*.

## 20.2. Complex List Content

Aside from nested lists, all of the list items you've seen only have one line of text. But like with regular paragraph text, the text in a list item can wrap across any number of lines, as long as all the lines are adjacent. The wrapped lines can be indented and they will still be treated as normal paragraph text. For example:

- \* The header in AsciiDoc is optional, but if it is used it must start with a document title.
- \* Optional Author and Revision information immediately follows the header title.
- \* The document header must be separated from the remainder of the document by one or more blank lines and cannot contain blank lines.

- The header in AsciiDoc is optional, but if it is used it must start with a document title.
- Optional Author and Revision information immediately follows the header title.
- The document header must be separated from the remainder of the document by one or more blank lines and cannot contain blank lines.



When items contain more than one line of text, leave a blank line before the next item to make the list easier to read.

A list item may contain any type of AsciiDoc content, including paragraphs, delimited blocks, and block macros. You just need to *attach* them to the list item.

### 20.2.1. List continuation

To add additional paragraphs or other block elements to a list item, you must “attach” them (in a series) using a *list continuation*. A list continuation is a `+` symbol on a line by itself, immediately adjacent to the block being attached.

Here’s an example:

```
* The header in AsciiDoc must start with a document title.  
+  
The header is optional.
```

- The header in AsciiDoc must start with a document title.  
The header is optional.

Using the list continuation, you can attach any number of block elements to a list item. Each block must be preceded by a list continuation to form a chain of blocks.

Here’s an example that attaches both a listing block and an admonition paragraph to the first item:

```
* The header in AsciiDoc must start with a document title.  
+  
----  
= Document Title  
----  
+  
Keep in mind that the header is optional.  
  
* Optional Author and Revision information immediately follows the header title.  
+  
----  
= Document Title  
Doc Writer <doc.writer@asciidoc.org>  
v1.0, 2013-01-01  
----
```

Here's how the source is rendered:

*A list with complex content*

- The header in AsciiDoc must start with a document title.

```
= Document Title
```

Keep in mind that the header is optional.

- Optional Author and Revision information immediately follows the header title.

```
= Document Title  
Doc Writer <doc.writer@asciidoc.org>  
v1.0, 2013-01-01
```

If you're attaching more than one block to a list item, you're strongly encouraged to wrap the content inside an open block. That way, you only need a single list continuation line to attach the open block to the list item. Within the open block, you write like you normally would, no longer having to worry about adding list continuations between the blocks to keep them attached to the list item.

Here's an example of wrapping complex list content in an open block:

```
* The header in AsciiDoc must start with a document title.
```

```
+
```

```
--
```

```
Here's an example of a document title:
```

```
----
```

```
= Document Title
```

```
----
```

```
NOTE: The header is optional.
```

```
--
```

Here's how that content is rendered:

*A list with complex content wrapped in an open block*

- The header in AsciiDoc must start with a document title.

Here's an example of a document title:

```
= Document Title
```



The header is optional.

The open block wrapper is also useful if you're including content from a shared file into a list item. For example:

```
* list item
```

```
+
```

```
--
```

```
include::shared-content.adoc[]
```

```
--
```

By wrapping the include directive in an open block, the content can be used unmodified.

The only limitation of this technique is that the content itself may not contain an open block (since open blocks cannot be nested).

### 20.2.2. Dropping the principal text

If the principal text of a list item is blank, the node for the principal text is dropped. This is how you can get the first block (such as a listing block) to line up with the list marker. You can make the principal text blank by using the `{blank}` attribute reference.

Here's an example of a list that has items with *only* complex content.

```
. {blank}
+
-----
print("one")
-----
. {blank}
+
-----
print("one")
-----
```

Here's how the source is rendered:

*A list with complex content*

1. 

```
print("one")
```
2. 

```
print("one")
```

### 20.2.3. Attaching to an ancestor list

You may find that you need to attach block content to a parent list item instead of the current one. In other words, you want to attach the block content to the parent list item so it becomes a sibling of the child list. To do this, you add a blank line before the list continuation. The blank line signals to the list continuation to move out of the current list so it attaches the block to the last item of the parent list.

Here's an example of a paragraph that's attached to the parent list item, placing it adjacent to the child list (instead of inside of it).

```
* parent list item
** child list item

+
paragraph attached to parent list item
```

Here's how the source is rendered:

*A block attached to the parent list item*

- parent list item
  - child list item
- paragraph attached to parent list item

Each blank line that precedes the list continuation signals a move up one level of nesting. Here's an example that shows how to attach a paragraph to a grandparent list item using two leading blank lines:

```
* grandparent list item
** parent list item
*** child list item

+
paragraph attached to grandparent list item
```

Here's how the source is rendered:

*A block attached to the grandparent list item*

- grandparent list item
  - parent list item
    - child list item
- paragraph attached to grandparent list item

Using blank lines to back out of the nesting may feel fragile. A more robust way to accomplish the same thing is to enclose the nested lists in an open block. That way, it's clear where the nested list ends and the enclosing list continues.

```
* grandparent list item
+
--
** parent list item
*** child list item
--
+
paragraph attached to grandparent list item
```

Here's how the source is rendered:

*A nested block enclosed in an open block*

- grandparent list item
  - parent list item
    - child list item
- paragraph attached to grandparent list item

As you've learned in this section, the primary text of a list item can wrap to any number of adjacent lines. You can also attach any type of content to a list item using the list continuation. Combining those features with the open block makes it even easier to create lists with complex content.

## 20.3. Custom Markers

Asciidoctor offers numerous bullet styles for lists. The list marker (bullet) is set using the list's block style.

The unordered list marker can be set using any of the following block styles:

- square
- circle
- disc
- none or no-bullet (indented, but no bullet)
- unstyled (no indentation or bullet) (HTML only)



These styles are supported by the default Asciidoctor stylesheet.

When present, the style name is assigned to the unordered list element as follows:

### For HTML

the style name is assigned to the `class` attribute on the `<ul>` element.

### For DocBook

the style name is assigned to the `mark` attribute on the `<itemizedlist>` element.

Here's an unordered list that has square bullets:

```
[square]
* one
* two
* three
```

- one
- two
- three

## 20.4. Checklist

List items can be marked complete using checklists.

Checklists (i.e., task lists) are unordered lists that have items marked as checked ([\*] or [x]) or unchecked ([ ]). Here's an example:

*Checklist syntax*

```
* [*] checked
* [x] also checked
* [ ] not checked
*     normal list item
```

*Rendered checklist*

- checked
- also checked
- not checked
- normal list item



Not all items in the list have to be checklist items, as the previous example shows.

When checklists are converted to HTML, the checkbox markup is transformed into an HTML checkbox with the appropriate checked state. The `data-item-complete` attribute on the checkbox is set to 1 if the item is checked, 0 if not. The checkbox is used in place of the item's bullet.

Since HTML generated by Asciidoctor is usually static, the checkbox is set as disabled to make it appear as a simple mark. If you want to make the checkbox interactive (i.e., clickable), add the `interactive` option to the checklist (shown here using the shorthand syntax for [Options](#)):

*Checklist with interactive checkboxes*

```
[%interactive]
* [*] checked
* [x] also checked
* [ ] not checked
*     normal list item
```

*Rendered checklist with interactive checkboxes*

- checked
- also checked
- not checked
- normal list item

As a bonus, if you enable font-based icons, the checkbox markup (in non-interactive lists) is transformed into a font-based icon!

*Checklist with font-based checkboxes*

```
[%interactive]
* [*] checked
* [x] also checked
* [ ] not checked
*     normal list item
```

# Chapter 21. Ordered Lists

Sometimes, we need to number the items in a list. Instinct might tell you to prefix each item with a number, like in this next list:

1. Protons
2. Electrons
3. Neutrons

The above works, but since the numbering is obvious, the AsciiDoc processor will insert the numbers for you if you omit them:

- . Protons
- . Electrons
- . Neutrons

1. Protons
2. Electrons
3. Neutrons

If you decide to use number for your ordered list, you have to keep them sequential. This differs from other lightweight markup languages. It's one way to adjust the numbering offset of a list. For instance, you can type:

4. Step four
5. Step five
6. Step six

However, in general the best practice is to use the `start` attribute to configure this sort of thing:

```
[start=4]
. Step four
. Step five
. Step six
```

To present the items in reverse order, add the `reversed` option:

```
[%reversed]
.Parts of an atom
. Protons
. Electrons
. Neutrons
```

*Parts of an atom*

3. Protons
2. Electrons
1. Neutrons

You can give a list a title by prefixing the line with a dot immediately followed by the text (without leaving any space after the dot).

Here's an example of a list with a title:

```
.Parts of an atom
. Protons
. Electrons
. Neutrons
```

*Parts of an atom*

1. Protons
2. Electrons
3. Neutrons

## 21.1. Nested

You create a nested item by using one or more dots in front of each the item.

```
. Step 1
. Step 2
.. Step 2a
.. Step 2b
. Step 3
```

AsciiDoc selects a different number scheme for each level of nesting. Here's how the previous list renders:

### A nested ordered list

1. Step 1

2. Step 2

  a. Step 2a

  b. Step 2b

3. Step 3

Like with the asterisks in an unordered list, the number of dots in an ordered list doesn't represent the nesting level. However, it's much more intuitive to follow this convention:



# of dots = level of nesting

Again, we are shooting for plain text markup that is readable *as is*.

You can mix and match the three list types, ordered, [unordered](#), and [description](#), within a single hybrid list. Asciidoctor works hard to infer the relationships between the items that are most intuitive to us humans.

Here's an example of nesting an unordered list inside of an ordered list:

```
. Linux
* Fedora
* Ubuntu
* Slackware
. BSD
* FreeBSD
* NetBSD
```

1. Linux

- Fedora
- Ubuntu
- Slackware

2. BSD

- FreeBSD
- NetBSD

You can spread the items out and indent the nested lists if that makes it more readable for you:

- . Linux
  - \* Fedora
  - \* Ubuntu
  - \* Slackware
- . BSD
  - \* FreeBSD
  - \* NetBSD

The description list section demonstrates how to [combine all three list types](#).

## 21.2. Numbering Styles

For ordered lists, Asciidoctor supports the numeration styles such as lowergreek and decimal-leading-zero.

The full list of numeration styles that can be applied to an ordered list are as follows:

*Asciidoctor ordered list numeration styles*

Block style	CSS list-style-type
arabic	decimal
decimal <sup>&amp;ast;</sup>	decimal-leading-zero
loweralpha	lower-alpha
upperalpha	upper-alpha
lowerroman	lower-roman
upperroman	upper-roman
lowergreek <sup>&amp;ast;</sup>	lower-greek

<sup>&ast;</sup> These styles are only supported by the HTML converters.

Here are a few examples showing various numeration styles as defined by the block style shown in the header row:

[arabic] <sup>&amp;ast;</sup>	[decimal]	[loweralpha]	[lowergreek]
1. one	01. one	a. one	α. one
2. two	02. two	b. two	β. two
3. three	03. three	c. three	γ. three

<sup>&ast;</sup> Default numeration if block style is not specified



Custom enumeration styles can be implemented using a custom role. Define a new class selector (e.g., `.custom`) in your stylesheet that sets the `list-style-type` property to the value of your choice. Then, assign the name of that class as a role on any list to which you want that enumeration applied.

When the role shorthand (`.custom`) is used on an ordered list, the enumeration style is no longer omitted.

You can override the number scheme for any level by setting its style (the first positional entry in a block attribute list). You can also set the starting number using the `start` attribute:

```
["lowerroman", start=5]
. Five
. Six
[loweralpha]
.. a
.. b
.. c
. Seven
```

- v. Five
- vi. Six
  - a. a
  - b. b
  - c. c
- vii. Seven



The `start` attribute must be a number, even when using a different enumeration style. For instance, to start an alphabetic list at letter "c", set the enumeration style to `loweralpha` and the `start` attribute to 3.

# Chapter 22. Description List

A description list (often abbreviate as dlist) is useful when you need to include a description or supporting text for one or more terms. Each item in a description list consists of:

- one or more terms
- a separator following each term (typically a double colon, ::)
- at least one space or endline
- the supporting content (either text, attached blocks, or both)

Here's an example of a description list that identifies parts of a computer:

```
CPU:: The brain of the computer.  
Hard drive:: Permanent storage for operating system and/or user files.  
RAM:: Temporarily stores information the CPU uses during operation.  
Keyboard:: Used to enter text or control items on the screen.  
Mouse:: Used to point to and select items on your computer screen.  
Monitor:: Displays information in visual form using text and graphics.
```

By default, the content of each item is displayed below the description when rendered. Here's a preview of how this list is rendered:

*A basic description list*

## **CPU**

The brain of the computer.

## **Hard drive**

Permanent storage for operating system and/or user files.

## **RAM**

Temporarily stores information the CPU uses during operation.

## **Keyboard**

Used to enter text or control items on the screen.

## **Mouse**

Used to point to and select items on your computer screen.

## **Monitor**

Displays information in visual form using text and graphics.

If you want the description and content to appear on the same line, add the horizontal style to the list.

[horizontal]

CPU:: The brain of the computer.

Hard drive:: Permanent storage for operating system and/or user files.

RAM:: Temporarily stores information the CPU uses during operation.

## CPU

The brain of the computer.

## Hard drive

Permanent storage for operating system and/or user files.

## RAM

Temporarily stores information the CPU uses during operation.

The content of a description list can be any AsciiDoc element. For instance, we could rewrite the grocery list from above so that each aisle is a description rather than a parent outline list item.

Dairy::

\* Milk

\* Eggs

Bakery::

\* Bread

Produce::

\* Bananas

## Dairy

- Milk

- Eggs

## Bakery

- Bread

## Produce

- Bananas

Description lists are quite lenient about whitespace, so you can spread the items out and even indent the content if that makes it more readable for you:

Dairy::

- \* Milk
- \* Eggs

Bakery::

- \* Bread

Produce::

- \* Bananas

Finally, you can mix and match the three list types within a single hybrid list. Asciidoctor works hard to infer the relationships between the items that are most intuitive to us humans.

Here's a list that mixes description, ordered, and unordered list items:

Operating Systems::

Linux:::

- . Fedora
  - \* Desktop
- . Ubuntu
  - \* Desktop
  - \* Server

BSD:::

- . FreeBSD
- . NetBSD

Cloud Providers::

PaaS:::

- . OpenShift
- . CloudBees

IaaS:::

- . Amazon EC2
- . Rackspace

Here's how the list is rendered:

## Operating Systems

### Linux

1. Fedora
  - Desktop
2. Ubuntu
  - Desktop
  - Server

### BSD

1. FreeBSD
2. NetBSD

## Cloud Providers

### PaaS

1. OpenShift
2. CloudBees

### IaaS

1. Amazon EC2
2. Rackspace

You can include more complex content in a list item as well.

## 22.1. Question and Answer Style List

### Q&A

[qanda]

What is Asciidoctor?::

An implementation of the AsciiDoc processor in Ruby.

What is the answer to the Ultimate Question?:: 42

1. *What is Asciidoctor?*

An implementation of the AsciiDoc processor in Ruby.

2. *What is the answer to the Ultimate Question?*

42

# Chapter 23. Tables

Tables are one of the most intricate, yet refined areas of the AsciiDoc syntax. Armed with a bit of knowledge, you should discover that they are both easy to create and easy to read in raw form. Yet, under all that simplicity, they are remarkably sophisticated.

Tables are delimited by `|==` and made up of cells. The default table data format is PSV (Prefix Separated Values), which means that the processor creates a new cell each time it encounters a vertical bar (`|`). Cells are grouped into rows. Each row must share the same number of cells, taking into account any [column or row spans](#). Then, each consecutive cell in a row is placed in a separate column.

The simple table example below consists of two columns and three rows.

*Simple table*

```
|== ①  
②  
| Cell in column 1, row 1 | Cell in column 2, row 1 ③  
④  
| Cell in column 1, row 2 | Cell in column 2, row 2  
  
| Cell in column 1, row 3 | Cell in column 2, row 3  
  
|== ①
```

- ① The table's content boundaries are defined by a vertical bar followed by three equal signs (`|==`).
- ② Inserting a blank line before the first row is a trick to ensure the first row is not treated as the table header.
- ③ The new cell is marked by a vertical bar (`|`).
- ④ Rows can optionally be separated by any number of blank lines.

*Result: Rendered simple table*

Cell in column 1, row 1	Cell in column 2, row 1
Cell in column 1, row 2	Cell in column 2, row 2
Cell in column 1, row 3	Cell in column 2, row 3

Like with all blocks, you can add a role to a table using the `role` attribute. The `role` attribute becomes a CSS class when converted to HTML. The preferred shorthand for assigning the `role` attribute is to put the role name in the first position of the block attribute list prefixed with a `.` character, as shown here:

```
[.rolename]
|===
| Cell in column 1, row 1 | Cell in column 2, row 1 | Cell in column 3, row 1
| Cell in column 1, row 2 | Cell in column 2, row 2 | Cell in column 3, row 2
|===

```

Leading and trailing spaces around cell content is stripped and, therefore, don't affect the table's layout when rendered. The two examples below illustrate how leading and trailing spaces don't change the rendered table's layout.

*Cell content adjacent to the |*

```
|===
|Cell in column 1, row 1|Cell in column 2, row 1
|===

```

*Result: Rendered table when cell content was entered adjacent to the |*

Cell in column 1, row 1	Cell in column 2, row 1
-------------------------	-------------------------

*Cell content with varying leading and trailing spaces*

```
|===
| Cell in column 1, row 1 | Cell in column 2, row 1
|===

```

*Result: Rendered table when cell content was bounded by varying leading and trailing spaces*

Cell in column 1, row 1	Cell in column 2, row 1
-------------------------	-------------------------

There are multiple ways to group cells into a row. The cells in a row can be placed on:

- the same line
- consecutive, individual lines
- a combination of a and b

### Cells on the same line

```
|===
|Cell in column 1, row 1 |Cell in column 2, row 1 |Cell in column 3, row 1
|Cell in column 1, row 2 |Cell in column 2, row 2 |Cell in column 3, row 2
|===
```

*Result: Rendered table when multiple cells where entered on the same line*

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

When the cells of a row are individually entered on consecutive lines, the `cols` attribute is needed to specify the number of columns in the table. If the `cols` attribute is not set, the first non-blank line inside the block delimiter (`|==`) determines the number of columns.

### Cells on consecutive, individual lines

```
[cols="3*"] ①
|===
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

① The `cols` attribute states that this table has three columns. The `*` is a repeat operator which is explained in the [column specifiers](#) section.

*Result: Rendered table when cells where listed on consecutive, individual lines*

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

Rows can be formed from adjacent lines of individual cells and cells listed on the same line.

### *Cells on the same line and individual lines*

```
[cols="3*"]
|===
|Cell in column 1, row 1 |Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2 |Cell in column 3, row 2
|===
```

*Result: Cells on the same line and individual lines*

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

The next sections describe and demonstrate the variety of ways you can customize table cells, rows and columns.

## 23.1. Columns

The number of columns in a table is determined by the number of cells found in the first non-blank line after the table delimiter (|==>) or by the values assigned to the `cols` attribute.

For example, the syntax in the two examples below will both converted to a table with two columns.

====	
Cell in column 1, row 1	Cell in column 2, row 1
Cell in column 1, row 2	
Cell in column 2, row 2	
====	

*Result: Rendered table with two columns as defined by the number of cells in the first row*

Cell in column 1, row 1	Cell in column 2, row 1
Cell in column 1, row 2	Cell in column 2, row 2

```
[cols="2*"]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2

|===
```

*Result: Rendered table with two columns as defined by the `cols` attribute*

Cell in column 1, row 1	Cell in column 2, row 1
Cell in column 1, row 2	Cell in column 2, row 2

When a single number is assigned to the `cols` attribute, its value indicates the number of columns. Each column will be the same width. However, the number of columns can also be assigned as a comma delimited list. The number of entries in the list determines the number of columns.

The comma delimited list below creates a table with four columns of equal width.

```
[cols="1,1,1,1"]
```

This syntax provides that same result:

```
[cols="4*"]
```

Now, let's talk about that asterisk in the syntax above.

## 23.2. Column Formatting

The AsciiDoc syntax provides a variety of ways to control the size, style and layout of content within columns. These **specifiers** can be applied to whole columns.

To apply a specifier to a column, you must set the `cols` attribute and assign it a value. A column specifier can contain any of the following components:

- multiplier
- align
- width
- style

Each component is optional.

The multiplier operator (\*) is used when you want a specifier to apply to more than one consecutive column. If used, the multiplier must always be placed at the beginning of the specifier.

For example:

```
[cols="3*"] ①
|===
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

① The table will consist of three columns, as indicated by the 3. The \* operator ensures that the default layout and style will be applied to all of the columns.

*Result: Rendered table with multiplier applied*

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

The alignment component allows you to horizontally or vertically align a column's content. Content can be horizontally aligned left (<), center (^), or right (>).

To horizontally center the content in all of the columns, add the ^ operator after the multiplier.

```
[cols="3*^"]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

*Result: Rendered table with horizontal, center alignment applied to all columns*

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

What if you only want to center the content in the last column? Assign the default styles to the preceding columns, and ^ to the last column in a comma separated list.

```
[cols="2*,^"]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

*Result: Rendered table with horizontal, center alignment applied to last column*

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

Let's specify a different horizontal alignment for each column.

```
[cols("<,>")]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

*Result: Rendered table with a different horizontal alignment for each column*

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

You'll notice that the content in the examples above is only centered on the horizontal. It can also be aligned vertically when the alignment operator is prefixed with a dot (.). Content can be vertically aligned to the top (<), middle (^), or bottom (>) of a cell.

To vertically align the content to the middle of the cells in all of the columns, add a . and then the ^ operator after the multiplier.

```
[cols="3*.^"]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

*Result: Rendered table with vertical, middle alignment applied to all columns*

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

If you only want to align the content to the bottom of each cell in the last column, you'd assign the default styles to the preceding columns, and > to the last column in a comma separated list.

```
[cols="2*,.>"]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

*Result: Rendered table with vertical, bottom alignment applied to last column*

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

Let's specify a different vertical alignment for each column.

```
[cols=".<,.^,.>"]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

*Result: Rendered table with a different vertical alignment for each column*

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

Finally, we'll also horizontally center the content in the last column.

```
[cols=".<,.^,.>"]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

*Result: Rendered table with a different vertical alignment for each column and horizontal, center alignment in the last column*

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

When both a horizontal and vertical alignment is assigned to a column, the horizontal alignment operator must precede the vertical operator.

The width component sets the width of a column. Its value can be a proportional integer (the default is 1) or a percentage (1 to 99).

```
[cols="1,2,6"]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

*Result: Table rendered with column sizes adjusted by a proportional integer*

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
-------------------------	-------------------------	-------------------------

Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2
-------------------------	-------------------------	-------------------------

When assigning percentage values to the cols attribute, you do not need to include the percent sign (%).

```
[cols="50,20,30"]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

*Result: Table rendered with column sizes adjusted by a percentage*

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

Let's create a table with custom widths and alignments.

```
[cols=".<2,.^5,^.>3"]
|===
|Cell in column 1, row 1 with lots and lots and lots and lots of content
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2 and another bucket of content, and then a jelly roll of
content
|===
```

*Result: Rendered table with variable widths and alignments*

Cell in column 1, row 1 with lots and lots and lots and lots of content	Cell in column 2, row 1	Cell in column 3, row 1
---	-------------------------	-------------------------

Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2 and another bucket of content, and then a jelly roll of content
----------------------------	-------------------------	--

The style component must always be located at the end of the specifier. When no style name is provided column contents will be processed as regular inline text.

The column styles are described in the table below.

Style Name	Value	Description
AsciiDoc	a	Supports block-level elements (paragraphs, delimited blocks, and block macros). This style effectively creates a nested, standalone AsciiDoc document. Implicit attributes such as doctitle from the parent document will be shadowed. Custom attributes are inherited.
Emphasis	e	Text is italicized
Header	h	Header styles are applied to the column
Literal	l	Column content is treated as if it were inside a literal block
Monospaced	m	Text is rendered in monospaced font
None (default style)	d	Text is handled like a normal paragraph. Supports all markup (i.e., inline formatting, inline macros) that is permitted in a paragraph.
Strong	s	Text is bolded
Verse	v	Column content is treated as if it were inside a verse block

Let's apply the header style to the first column, the monospaced style to the second, the strong style to the third, and the emphasis style to the fourth.

```
[cols="h,m,s,e"]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1
|Cell in column 4, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|Cell in column 4, row 2
|===
```

*Result: Rendered table with a header, monospaced, and strong styled column*

<b>Cell in column 1, row 1</b>	<b>Cell in column 2, row 1</b>	<b>Cell in column 3, row 1</b>	<i>Cell in column 4, row 1</i>
<b>Cell in column 1, row 2</b>	<b>Cell in column 2, row 2</b>	<b>Cell in column 3, row 2</b>	<i>Cell in column 4, row 2</i>

Specifiers can also be applied to individual cells.

### 23.3. Cell Formatting

In addition to sharing many of the column specifier capabilities, cell specifiers allow cells to span rows and columns. Like a column specifier, a cell specifier is made up of components. These components, listed and defined below, are all optional.

- span
- align
- style

A cell specifier is prefixed directly to the cell delimiter (|) preceding the content you want to customize.

The span component can duplicate a cell or have it span multiple rows or columns.

To duplicate a cell in multiple, consecutive columns, prefix the | with the multiplication factor and the \* operator.

*Cell duplicated across three columns*

===   Cell in column 1, row 1  Cell in column 2, row 1  Cell in column 3, row 1  3* Same cell content in columns 1, 2, and 3   Cell in column 1, row 3  Cell in column 2, row 3  Cell in column 3, row 3   === 
--

*Result: Rendered table where cell was duplicated across three columns*

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Same cell content in columns 1, 2, and 3	Same cell content in columns 1, 2, and 3	Same cell content in columns 1, 2, and 3
Cell in column 1, row 3	Cell in column 2, row 3	Cell in column 3, row 3

To have a cell span multiple, consecutive columns, prefix the | with the span factor and the + operator.

#### *Cell spanning three columns*

```
|===
|Cell in column 1, row 1 |Cell in column 2, row 1 |Cell in column 3, row 1
3+|Content in a single cell that spans columns 1, 2, and 3
|Cell in column 1, row 3
|Cell in column 2, row 3
|Cell in column 3, row 3
|===
```

#### *Result: Rendered table where cell spans three columns*

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Content in a single cell that spans columns 1, 2, and 3		
Cell in column 1, row 3	Cell in column 2, row 3	Cell in column 3, row 3

If you want to have a cell span multiple, consecutive rows, prefix the span factor with a dot (.).

#### *Cell spanning two rows*

```
|===
|Cell in column 1, row 1 |Cell in column 2, row 1 |Cell in column 3, row 1
.2+|Content in a single cell that spans rows 2 and 3
|Cell in column 2, row 2
|Cell in column 3, row 2
|Cell in column 2, row 3
|Cell in column 3, row 3
|===
```

#### *Result: Rendered table where a cell spans two rows*

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Content in a single cell that spans rows 2 and 3	Cell in column 2, row 2	Cell in column 3, row 2
	Cell in column 2, row 3	Cell in column 3, row 3

Of course you can combine spanning over columns and rows. The number before the dot(.) is the number of columns to span and the number after the dot(.) is the number of rows to span.

## *Cell spanning columns and rows*

```
|===
|Column 1, row 1 |Column 2, row 1 |Column 3, row 1 |Column 4, row 1
|Column 1, row 2
2.3+|Content in a single cell that spans over rows and columns
|Column 4, row 2
|Column 1, row 3
|Column 4, row 3
|Column 1, row 4
|Column 4, row 4
|===
```

*Result: Rendered table where a cell spans over columns and rows*

Column 1, row 1	Column 2, row 1	Column 3, row 1	Column 4, row 1
Column 1, row 2	Content in a single cell that spans over rows and columns		Column 4, row 2
Column 1, row 3			Column 4, row 3
Column 1, row 4			Column 4, row 4

The alignment component for cells works the same as the [column specifier alignment component](#).

*Cells aligned horizontally, vertically, and across a span of three columns*

```
[cols="3"]
|===
^|Prefix the '{vbar}' with '{caret}' to center content horizontally
<|Prefix the '{vbar}' with '<' to align the content to the left horizontally
>|Prefix the '{vbar}' with '>' to align the content to the right horizontally

.^|Prefix the '{vbar}' with a '.' and '{caret}' to center the content in the cell
    vertically
.<|Prefix the '{vbar}' with a '.' and '<' to align the content to the top of the cell
.>|Prefix the '{vbar}' with a '.' and '>' to align the content to the bottom of the
    cell

3+^.^|This content spans three columns ('3{plus}') and is centered horizontally
    ('{caret}') and vertically ('.{caret}') within the cell.

|===
```

*Result: Rendered cells aligned horizontally, vertically, and across a span of three columns*

Prefix the   with ^ to center content horizontally	Prefix the   with < to align the content to the left horizontally	Prefix the   with > to align the content to the right horizontally
Prefix the   with a . and ^ to center the content in the cell vertically	Prefix the   with a . and < to align the content to the top of the cell	Prefix the   with a . and > to align the content to the bottom of the cell
This content spans three columns (3+) and is centered horizontally (^) and vertically (.^) within the cell.		



To use a pipe (|) within the content of a cell without creating a new cell, you must use the {vbar} attribute.

The `style component` can also be applied to individual cells. For example, you can apply the AsciiDoc element styles to an individual cell by prefixing the vertical bar with an `a`.

#### *Comparing cells using AsciiDoc styles and no AsciiDoc styles*

```
[cols="2"]
|===
a|This cell is prefixed with an 'a', so the processor interprets the following lines as an AsciiDoc list.

* List item 1
* List item 2
* List item 3
|This cell *is not* prefixed with an 'a', so the processor does not interpret the following lines as an AsciiDoc list.

* List item 1
* List item 2
* List item 3

a|This cell is prefixed with an 'a', so the processor honors the 'lead' style on the following paragraph.

[.lead]
I am a paragraph styled with the lead attribute.
|This cell *is not* prefixed with an 'a', so the processor does not honor the 'lead' style on the following paragraph.

[.lead]
I am a paragraph styled with the lead attribute.
|===
```

*Result: Rendered table comparing cells using AsciiDoc styles and no AsciiDoc styles*

This cell is prefixed with an <code>a</code> , so the processor interprets the following lines as an AsciiDoc list.	This cell <b>is not</b> prefixed with an <code>a</code> , so the processor does not interpret the following lines as an AsciiDoc list.  * List item 1 * List item 2 * List item 3
This cell is prefixed with an <code>a</code> , so the processor honors the <code>lead</code> style on the following paragraph.	This cell <b>is not</b> prefixed with an <code>a</code> , so the processor does not honor the <code>lead</code> style on the following paragraph.

Source code listing can be placed inside cells by using the listing syntax.

#### *Cells with source code listing*

```
|===
|Source Code 1 |Source Code 2

a|
[source,python]
-----
import os
print "%s" %(os.uname())
-----

a|
[source,python]
-----
import os
print ("%s" %(os.uname()))
-----
|===
```

*Result: Rendered table with cells containing source code listing*

Source Code 1	Source Code 2
<code>import os print "%s" %(os.uname())</code>	<code>import os print ("%s" %(os.uname()))</code>

For the grand finale, let's apply a variety of specifiers to individual cells.

|==

2\*>m|This content is duplicated across two columns.

It is aligned right horizontally.

And it is monospaced.

.3+^.>s|This cell spans 3 rows. The content is centered horizontally, aligned to the bottom of the cell, and strong.

e|This content is emphasized.

.^l|This content is aligned to the top of the cell and literal.

v|This cell contains a verse  
that may one day expound on the  
wonders of tables in an  
epic sonnet.

|==

*Result: Rendered table featuring a variety of cell specifiers*

This content is duplicated across two columns.	This content is duplicated across two columns.
It is aligned right horizontally.	It is aligned right horizontally.
And it is monospaced.	And it is monospaced.
<b>This cell spans 3 rows. The content is centered horizontally, aligned to the bottom of the cell, and strong.</b>	<i>This content is emphasized.</i>  <i>This content is aligned to the top of the cell and literal.</i>  This cell contains a verse that may one day expound on the wonders of tables in an epic sonnet.

## 23.4. Header Row

The first row of a table is promoted to a header row if the `header` option is set (either explicitly or implicitly).

You can enable the `header` option by adding the `header` keyword to the comma-separated list of values in the `options` attribute on the table. You can also enable the `header` option by adding the `%header` directive to the table style (the first positional attribute).

In the example below, the table has an attribute list containing an `options` attribute that includes the `header` option.

*Table with the header value assigned to the options attribute*

```
[cols=2*,options="header"]  
|====  
|Name of Column 1  
|Name of Column 2  
  
|Cell in column 1, row 1  
|Cell in column 2, row 1  
  
|Cell in column 1, row 2  
|Cell in column 2, row 2  
|====
```

*Result: Rendered table when the header value is assigned to the options attribute*

Name of Column 1	Name of Column 2
Cell in column 1, row 1	Cell in column 2, row 1
Cell in column 1, row 2	Cell in column 2, row 2

Alternately, you can define the header row based on how you layout the table. Asciidoctor uses the following conventions to determine when the first row should become the header row:

1. The first line of content inside the table block delimiters is not empty.
2. The second line of content inside the table block delimiters is empty.
3. The **options** attribute has not been specified in the block attributes (prior to 1.5.5).

As seen in the result of the example below, if all of these rules hold, then the first row of the table is treated as a header.

*Table that has an implicit header row*

```
|====  
|Name of Column 1 |Name of Column 2  
  
|Cell in column 1, row 1  
|Cell in column 2, row 1  
  
|Cell in column 1, row 2  
|Cell in column 2, row 2  
|====
```

*Result: Rendered table when the header row was assigned implicitly*

Name of Column 1	Name of Column 2
Cell in column 1, row 1	Cell in column 2, row 1
Cell in column 1, row 2	Cell in column 2, row 2

If you want to suppress this implicit behavior of promoting the first row to a header row, set the option value `noheader` (or add the `%noheader` directive to the table style).

Notice that when the implicit method of assigning the header row is used, it's not necessary to set the `cols` attribute. That's because the number of columns are determined by the number of cells in the first line if the `cols` attribute is absent.



We're considering using a similar convention for enabling the footer in the future. Thus, if you rely on this convention to enable the header row, it's advised that you not put all the cells in the last row on the same line unless you intend on making it the footer row.

## 23.5. Footer Row

The last row of a table can be styled as a footer by adding the `footer` keyword to the `options` attribute.

```
[options="footer"]
|===
|Name of Column 1 |Name of Column 2

|Cell in column 1, row 1
|Cell in column 2, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2

|Footer in column 1, row 3
|Footer in column 2, row 3
|===
```

*Result: Table rendered with a footer*

Name of Column 1	Name of Column 2
Cell in column 1, row 1	Cell in column 2, row 1
Cell in column 1, row 2	Cell in column 2, row 2
Footer in column 1, row 3	Footer in column 2, row 3

## 23.6. Table Width

By default, a table will span the width of the content area. To constrain the width of the table to a fixed value, set the `width` attribute in the table's attribute list. The width is an integer percentage value ranging from 1 to 100. The `%` sign is optional.

*Table with width set to 75%*

```
[width=75%]
|===
|Name of Column 1 |Name of Column 2 |Name of Column 3
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1
|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
|==
```

*Result: Table rendered with a width of 75%*

Name of Column 1	Name of Column 2	Name of Column 3
Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

Alternately, you can make the width fit the content by setting the `autowidth` option. The columns inherit this setting, so individual columns will also be sized according to the content.

*Table using autowidth*

```
%autowidth
|===
|Name of Column 1 |Name of Column 2 |Name of Column 3
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1
|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
|==
```

*Result: The width of the table is sized to accomodate the automatic column widths*

Name of Column 1	Name of Column 2	Name of Column 3
Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

If you want each column to have an automatic width, but want the table to span the width of the content area, add the `stretch` role to the table or set the `width` attribute to `100%`.

## Full-width table with autowidth columns

```
[%autowidth.stretch]
|===
|Name of Column 1 |Name of Column 2 |Name of Column 3
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1
|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
|==
```

*Result: Columns are sized to the content, but table spans the width of the page*

Name of Column 1	Name of Column 2	Name of Column 3
Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2



The `autowidth` option is not recognized by the DocBook converter.

If you want to apply autowidth only to certain columns, use the special value `~` as the width of the column. In this case, width values are assumed to be a percentage value (i.e., 100-based).

## Table with both fixed and autowidth columns

```
[cols="25h,~,~"]
|===
|small |as big as the column needs to be |the rest
|===
|==
```

# 23.7. Table Borders

The borders on a table are controlled using the `frame` and `grid` attributes. You can combine these two attributes to achieve a variety of border styles for your tables.

## 23.7.1. Frame

The border around a table is controlled using the `frame` attribute. By default, the frame attribute is assigned the `all` value, which draws a border on each side of the table. If you set the frame attribute, you can override the default value with the values `topbot`, `sides` or `none`.

The `topbot` value draws a border on the top and bottom of the table.

```
[frame=topbot]
|===
|Name of Column 1 |Name of Column 2 |Name of Column 3

|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

*Result:* Table rendered with frame=topbot

Name of Column 1	Name of Column 2	Name of Column 3
Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

The `sides` value draws a border on the right and left side of the table.

```
[frame=sides]
|===
|Name of Column 1 |Name of Column 2 |Name of Column 3

|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

*Result:* Table rendered with frame=sides

Name of Column 1	Name of Column 2	Name of Column 3
Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

The `none` value removes the borders around the table.

```
[frame=none]
|===
|Name of Column 1 |Name of Column 2 |Name of Column 3

|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

*Result: Table rendered with frame=none*

Name of Column 1	Name of Column 2	Name of Column 3
Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

### 23.7.2. Grid

The borders between the cells in a table are controlled using the `grid` attribute. By default, the `grid` attribute is assigned the `all` value, which draws a border between all cells. If you set the `grid` attribute, you can override the default value with the values `rows`, `cols` or `none`.

The `rows` value draws a border between the rows of the table.

```
[grid=rows]
|===
|Name of Column 1 |Name of Column 2 |Name of Column 3

|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

*Result: Table rendered with grid=rows*

Name of Column 1	Name of Column 2	Name of Column 3
Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

The `cols` value draws borders between the columns.

```
[grid=cols]
|===
|Name of Column 1 |Name of Column 2 |Name of Column 3

|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

*Result: Table rendered with grid=cols*

Name of Column 1	Name of Column 2	Name of Column 3
Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

The `none` value removes all borders between the cells.

```
[grid=none]
|===
|Name of Column 1 |Name of Column 2 |Name of Column 3

|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

*Result: Table rendered with grid=none*

Name of Column 1	Name of Column 2	Name of Column 3
Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

## 23.8. Striping

Starting with Asciidoctor 2.0, the rows in the body of a table are not shaded. You can configure the rows to be striped (as in zebra stripes) using an attribute. Use the `stripes` table attribute to configure the stripes for a single table. Use the `table-stripes` document attribute to configure stripes for all tables.

Striping is done by adding a background color to the specified rows. The `stripes` attribute accepts the following values:

- `none` - no rows are shaded (default in Asciidoctor >= 2.0)
- `even` - even rows are shaded (default in Asciidoctor < 2.0)
- `odd` - odd rows are shaded
- `all` - all rows are shaded
- `hover` - the row under the mouse cursor is shaded (HTML only)

In the following example, the stripes are enabled for even rows in the table body (the row that contains A2, B2, and C2).

```
[cols=3*, stripes=even]
|===
| A1
| B1
| C1

| A2
| B2
| C2

| A3
| B3
| C3
|===
```

Under the covers, the `stripes` attribute applies the CSS class `stripes-<value>` (e.g., `stripes-none`) to the table tag. As a shorthand, you can simply apply the CSS class to the table directly using a role.

```
[.stripes-even,cols=3*]
|===
| A1
| B1
| C1

| A2
| B2
| C2

| A3
| B3
| C3
|===
```

If you want to apply stripes to all tables in the document, set the `table-stripes` attribute in the document header. You can still override this setting per table.

```
:table-stripes: even
```

```
[cols=3*]
```

```
|==
```

```
| A1
```

```
| B1
```

```
| C1
```

```
| A2
```

```
| B2
```

```
| C2
```

```
| A3
```

```
| B3
```

```
| C3
```

```
|==
```

In the HTML output, table striping is done using CSS and thus depends on the stylesheet to supply the necessary styles. The default stylesheet for Asciidoctor includes these styles.

## 23.9. Orientation

A table can be displayed in landscape (rotated 90 degrees counterclockwise) using the `rotate` option (preferred):

```
[%rotate]
```

```
|==
```

```
| a | b
```

```
| c | d
```

```
|==
```

or the `orientation` attribute:

```
[orientation=landscape]
```

```
|==
```

```
| a | b
```

```
| c | d
```

```
|==
```

Currently, this is only implemented by the DocBook backend (it sets the attribute `orient="land"`).

## 23.10. Nested Tables

Table cells marked with the AsciiDoc table style (`a`) support nested tables in addition to normal block content. To distinguish the inner table from the enclosing one, you need to use `!==` as the table delimiter and a cell separator that differs from that used for the enclosing table. The default

cell separator for a nested table is `!`, though you can choose another character by defining the `separator` attribute on the table.



Although nested tables are not technically valid in DocBook 5.0, the DocBook toolchain processes them anyway.

The following example contains a nested table in the last cell. Notice the nested table has its own format, independent of that of the outer table:

```
[cols="1,2a"]
|===
| Col 1 | Col 2
|
| Cell 1.1
| Cell 1.2
|
| Cell 2.1
| Cell 2.2

[cols="2,1"]
!===
! Col1 ! Col2
!
! C11
! C12
!
! ===
|===
```

*Result: A nested table*

Col 1	Col 2				
Cell 1.1	Cell 1.2				
Cell 2.1	Cell 2.2				
	<table border="1"><thead><tr><th>Col1</th><th>Col2</th></tr></thead><tbody><tr><td>C11</td><td>C12</td></tr></tbody></table>	Col1	Col2	C11	C12
Col1	Col2				
C11	C12				

We recommend using nested tables sparingly. There's usually a better way to present the information.

## 23.11. Table Caption

If you specify a title on a table, the title will be used as part of the table's caption. Adding a title also designates the table as a *formal table*.

By default, the title of a formal table is prefixed with the label **Table <n>**, followed by a space, where **<n>** is the 1-based index of all formal tables in the document.

```
.A formal table
|===
|Name of Column 1 |Name of Column 2

|Cell in column 1, row 1
|Cell in column 2, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|===
```

*Table 1. A table with a title*

Name of Column 1	Name of Column 2
Cell in column 1, row 1	Cell in column 2, row 1
Cell in column 1, row 2	Cell in column 2, row 2

You can customize the caption label by specifying the **caption** attribute. (Don't let the name of the attribute mislead you. The caption attribute only sets the caption's label, not the whole caption line).



If you want a space between the label and the title, you must add a trailing space to the value of the caption attribute.

```
[caption="Table A. "]
.A formal table
|===
|Name of Column 1 |Name of Column 2

|Cell in column 1, row 1
|Cell in column 2, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|===
```

*Table A. A formal table*

Name of Column 1	Name of Column 2
Cell in column 1, row 1	Cell in column 2, row 1
Cell in column 1, row 2	Cell in column 2, row 2

If you want the caption of the table to only consist of the caption label, use the following syntax:

```
[caption=,title='{table-caption} {counter:table-number}']
|===
|Name of Column 1 |Name of Column 2

|Cell in column 1, row 1
|Cell in column 2, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|===
```

Alternately, you can write is as follows:

```
.{blank}
[caption='{table-caption} {counter:table-number}']
|===
|Name of Column 1 |Name of Column 2

|Cell in column 1, row 1
|Cell in column 2, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|===
```

If you want to exclude the caption label altogether, simply assign a blank value to the `caption` attribute.

```
[caption=]
.A formal table
|===
|Name of Column 1 |Name of Column 2

|Cell in column 1, row 1
|Cell in column 2, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|===
```

*A formal table*

Name of Column 1	Name of Column 2
Cell in column 1, row 1	Cell in column 2, row 1
Cell in column 1, row 2	Cell in column 2, row 2

Alternatively, you can disable the caption label for tables globally by undefining the `table-caption`

document attribute.

```
:table-caption!:
```

## 23.12. Escaping the Cell Separator

The parser scans for the cell separator to partition cells *before* it processes the cell text. So even if you try to hide the cell separator using an inline passthrough, the parser will see it. If the cell contain contains the cell separator, you must escape that character. There are three ways to escape it:

- Prefix the character with a leading backslash (i.e., `\|`), which will be removed from the output.
- Use the `{vbar}` attribute reference as a substitute.
- Change the cell separator used by the table.

Unless you do one of these things, the cell separator will be interpreted as a cell boundary.

Consider the following example, which escapes the cell separator using a leading backslash:

```
[cols=2*]
=====
|The default separator in PSV tables is the \| character.
|The \| character is often referred to as a "pipe".
=====
```

This table will render as follows:

*Result: Converted PSV table that contains pipe characters*

The default separator in PSV tables is the   character.	The   character is often referred to as a “pipe”.
---	---

Notice that the pipe character appears without the leading backslash (i.e., unescaped) in the rendered result.

An alternative is to use the `{vbar}` attribute reference as a substitute. This approach produces the same result as the previous example.

```
[cols=2*]
=====
|The default separator in PSV tables is the {vbar} character.
|The {vbar} character is often referred to as a "pipe".
=====
```

Escaping each cell separator character that appears in the content of a cell can be tedious. There are also times when you can't or don't want to modify the cell content (perhaps because it is being

included from another file). To address these cases, AsciiDoc allows you to override the cell separator.

The cell separator is controlled using the `separator` attribute on the table block. You'll want to select a character that will never be used for content. A good candidate is the broken bar, or `|`.

Here's the previous example rewritten using a custom separator.

```
[cols=2*,separator='|']
=====
|The default separator in PSV tables is the | character.
|The | character is often referred to as a "pipe".
=====
```

Notice that it's no longer necessary to escape the pipe character in the content of the table cells. You can safely use the original cell separator in the cell content and not worry about it being interpreted as the boundary of a cell.

## 23.13. Delimiter-Separated Values

Tables can also be populated from data formatted as delimiter-separated values (i.e., data tables). In contrast with the PSV format, in which the delimiter is placed in front of each cell value, the delimiter in a delimiter-separated format (CSV, TSV, DSV) is placed between the cell values (called a `separator`) and does not accept a cell formatting spec. Each line of data is assumed to represent a single row, though you'll learn that's not a strict rule. How the table data gets interpreted is controlled by the `format` and `separator` attributes on the table.

### What the delimiter?

Aren't comma-separated values a subset of [delimiter-separated values](#)? It really depends on who you consult.

The term “delimiter-separated values” in this text refers to the family of data formats that use a delimiter, including comma-separated values (CSV), tab-separated values (TSV) and delimited data (DSV), all of which are supported in AsciiDoc tables. CSV is the data format used most often.

“Comma-separated values” is really a misleading term since CSV can use delimiters other than `,` as the field separator (which, in this context, separates cells). What we're really talking about is how the data is interpreted.

CSV and TSV both use a delimiter and an optional enclosing character, loosely based on [RFC 4180](#). DSV (i.e., delimited data) only uses a delimiter, which can be escaped using a backslash; an enclosing character is not recognized. These parsing rules are described in detail in [Data Table Formats](#).

Let's consider an example of using comma-separated values (CSV) to populate an AsciiDoc table

with data. To instruct the processor to read the data as CSV, set the value of the `format` attribute on the table to `csv`. When the `format` attribute is set to `csv`, the default data separator is a comma (,), as seen in the table below.

```
[%header,format=csv]
|===
Artist,Track,Genre
Baauer,Harlem Shake,Hip Hop
The Lumineers,Ho Hey,Folk Rock
|===
```

*Result: Rendered CSV table*

Artist	Track	Genre
Baauer	Harlem Shake	Hip Hop
The Lumineers	Ho Hey	Folk Rock

This feature is particularly useful when you want to populate a table in your manuscript from data stored in a separate file. You can do so using the `include directive` between the table delimiters, as shown here:

```
[%header,format=csv]
|===
include::tracks.csv[]
|===
```

If your data is separated by tabs instead of commas, set the `format` to `tsv` (tab-separated values) instead.

Now let's consider an example of using delimited data (DSV) to populate an AsciiDoc table with data. To instruct the processor to read the data as DSV, set the value of the `format` attribute on the table to `dsv`. When the `format` attribute is set to `dsv`, the default data separator is a colon (:), as seen in the table below.

```
[%header,format=dsv]
|===
Artist:Track:Genre
Robyn:Indestructable:Dance
The Piano Guys:Code Name Vivaldi:Classical
|===
```

*Result: Rendered DSV table*

Artist	Track	Genre
Robyn	Indestructable	Dance
The Piano Guys	Code Name Vivaldi	Classical

### 23.13.1. Data Table Formats

The CSV and TSV data formats are parsed differently from the DSV data format. The following two sections outline those differences.

#### CSV and TSV

Table data in either CSV or TSV format is parsed according to the following rules, loosely based on [RFC 4180](#):

- The default delimiter for CSV is a comma (,) while the default delimiter for TSV is a tab character.
- Blank lines are skipped (unless enclosed in a quoted value).
- Whitespace surrounding each value is stripped.
- Values can be enclosed in double quotes ("").
  - A quoted value may contain zero or more separator or newline characters.
  - A newline begins a new row unless the newline is enclosed in double quotes.
  - A quoted value may include the double quote character if escaped using another double quote ("").
  - Newlines in quoted values are retained (as of 1.5.7).
- If rows do not have the same number of cells (“ragged” tables), cells are shuffled to fully fill the rows.
  - This is different behavior than Excel, which pads short rows with empty cells.
  - Extra cells at the end of the last row get dropped.
  - As a rule of thumb, data for a single row should be on the same line.

#### DSV

Table data in DSV format is parsed according to the following rules:

- The default delimiter for DSV is a colon (:).
- Blank lines are skipped.
- Whitespace surrounding each value is stripped.
- The delimiter character can be included in the value if escaped using a single backslash (\:).
- If rows do not have the same number of cells (“ragged” tables), cells are shuffled to fully fill the rows.

### 23.13.2. Custom Delimiters

Each data format has a default separator associated with it (csv = comma, tsv = tab, dsv = colon), but the separator can be changed to any character (or even a string of characters) by setting the `separator` attribute on the table.

Here’s an example of a DSV table that uses a custom separator character (i.e., delimiter):

A DSV table with a custom separator

```
[format=csv,separator=;]  
|===  
a;b;c  
d;e;f  
|==
```



To make a TSV table, you can set the `format` attribute to `csv` and the separator to `\t`. Though the `tsv` format is preferred.

The separator is independent of the processing rules for the format. If you set `format=csv` and `separator=,`, the data will be processed using the DSV rules, even though the data looks like CSV.

### 23.13.3. Shorthand Notation for Data Tables

Asciidoctor provides shorthand notation for specifying the data format of a table. The first position of the table block delimiter (i.e., `| ==`) can be replaced by a built-in delimiter to set the table format (e.g., `,` == for CSV).

To make a CSV table, you can use `,` == as the table block delimiter:

```
,===  
Artist,Track,Genre  
  
Baauer,Harlem Shake,Hip Hop  
,==
```

*Result: Rendered CSV table using shorthand syntax*

Artist	Track	Genre
Baauer	Harlem Shake	Hip Hop

To make a DSV table, you can use `:` == as the table block delimiter:

```
:===  
Artist:Track:Genre  
  
Robyn:Indestructable:Dance  
:==
```

*Result: Rendered DSV table using shorthand syntax*

Artist	Track	Genre
Robyn	Indestructable	Dance

When using either the CSV or DSV shorthand, you do not need to set the `format` attribute as it's

implied.

To make a TSV table, you can set the `format` attribute to `tsv` instead of having to set the `format` to `csv` and the separator to `\t`. In this case, you can use either `|==` or `,==` as the table block delimiter. There is no special delimited block notation for a TSV table.

### 23.13.4. Formatting Cells in a Data Table

The delimited formats do not provide a way to express formatting of individual table cells. Instead, you can apply cell formatting to all cells in a given column using the `cols` spec on the table:

```
[format=csv,cols="1h,1a"]
|===
Sky,image::sky.jpg[]
Forest,image::forest.jpg[]
|==
```

Data tables do not support cells that span multiple rows or columns, since that information can only be expressed at the cell level. You are advised to use the PSV format if you need that functionality.

## 23.14. Summary

*Table attributes and values*

Attribute	Description	Value	Description	Notes
<code>cols</code>	comma-separated list of column specifiers	specifiers	see <a href="#">Columns</a> for details	
<code>format</code>	data format of the table's contents	<code>psv</code>	cells are delimited by <code>separator</code> (default <code> </code> ) (aka prefix-separated values)	
		<code>dsv</code>	cells are delimited by a colon ( <code>:</code> ) (aka delimiter-separated values)	
		<code>csv</code>	cells are delimited by a comma ( <code>,</code> ) (aka comma-separated values)	
		<code>tsv</code>	cells are delimited by a tab character (aka tab-separated values)	

Attribute	Description	Value	Description	Notes
<code>separator</code>	character used to separate cells	 ! user-defined	default for top-level tables default for nested tables any sequence of literal characters (e.g.,   or ][]). <i>Ideally a pattern not found in the cell content.</i>	
<code>frame</code>	draws a border around the table	<code>all</code> <code>topbot</code> <code>none</code> <code>sides</code>	border on all sides (default) border on top and bottom no borders border on left and right sides	
<code>grid</code>	draws boundary lines between rows and columns	<code>all</code> <code>cols</code> <code>rows</code> <code>none</code>	draws boundary lines around each cell (default) draws boundary lines between columns draws boundary lines between rows no boundary lines	
<code>stripes</code>	controls row shading (via background color)	<code>none</code> <code>even</code> <code>odd</code> <code>hover</code> <code>all</code>	no rows are shaded even rows are shaded (default in Asciidoctor < 2.0) odd rows are shaded row under the mouse cursor is shaded (HTML only) all rows are shaded	
<code>align</code>	horizontally aligns table within page width	<code>left</code> <code>right</code> <code>center</code>	aligns to left side of page (default) aligns to right side of page horizontally aligns to center of page	Not yet implemented in Asciidoctor. Applies to HTML output only. The <code>align</code> and <code>float</code> attributes are mutually exclusive.

Attribute	Description	Value	Description	Notes
<code>float</code>	floats the table to the specified side of the page	<code>left</code>	floats the table to the left side of the page (default)	Applies to HTML output only. Must be used in conjunction with the table's <code>width</code> attribute to take effect. The <code>float</code> and <code>align</code> attributes are mutually exclusive.
		<code>right</code>	floats the table to the right side of the page	
<code>halign</code>	horizontally aligns all of the cell contents in a table	<code>left</code>	aligns the contents of the cells to the left (default)	Not implemented in Asciidoc. Define instead using column or cell specifiers (e.g., <code>3*&gt;</code> ), which take precedence over this value.
		<code>right</code>	aligns the contents of the cells to the right	
		<code>center</code>	aligns the contents to the cell centers	
<code>valign</code>	vertically aligns all of the cell contents in a table	<code>top</code>	aligns the cell contents to the top of the cell (default)	Define instead using column or cell specifiers (e.g., <code>3*.&gt;</code> ), which take precedence over this value.
		<code>bottom</code>	aligns the cell contents to the bottom of the cell	
		<code>middle</code>	aligns the cell contents to the middle of the cell	
<code>orientation</code>	rotates the table	<code>landscape</code>	rotated 90 degrees counterclockwise	Equivalent to setting the <code>rotate</code> option, which is preferred. DocBook only.
<code>options</code>	comma separated list of option names	<code>header</code>	promotes first row to the table header	header and footer rows are omitted by default
		<code>footer</code>	promotes last row to the table footer	
		<code>breakable</code>	allows the table to split across a page (default)	Mutually exclusive. DocBook only (specifically for generating PDF output).
		<code>unbreakable</code>	prevents the table from being split across a page	
		<code>autowidth</code>	disables explicit column widths (ignores <code>cols</code> attribute)	
		<code>rotate</code>	Prints the table in landscape	Equivalent to setting the orientation to landscape. DocBook only.

Attribute	Description	Value	Description	Notes
width	the table width relative to the available page width	user defined value	a percentage value between 0% and 100%	

# Chapter 24. Horizontal Rules

*Horizontal rule syntax*

```
---
```

*Result: Horizontal rule*

---

## 24.1. Markdown-style horizontal rules

Asciidoctor recognizes Markdown horizontal rules. The motivation here is to ease migration of Markdown documents to AsciiDoc documents.

To avoid conflicts with AsciiDoc's block delimiter syntax, only 3 repeating characters (- or \*) are recognized. As with Markdown, whitespace between the characters is optional.

*Markdown-style horizontal rule syntax*

```
---
```

```
- - -
```

```
***
```

```
* * *
```

*Result: Markdown-style horizontal rules*

---

---

---

A macro definition for the Markdown horizontal rules is included in the AsciiDoc compatibility file so they can be recognized by the `asciidoc` command as well.

# Chapter 25. Page Break

*Page break syntax*

<<<

# Chapter 26. URLs

A Uniform Resource Link (URL) represents the location of a resource on the web. Typical URLs contain a scheme, domain name, file name, and extension.



Asciidoctor recognizes the following common schemes without the help of any markup.

- http
- https
- ftp
- irc
- mailto
- email@email.com

You can think of these like implicit macro names. Since the URL in the example below begins with a protocol (in this case `https` followed by a colon), Asciidoctor will automatically turn it into a hyperlink when it is processed.

The homepage for the Asciidoctor Project is `https://asciidoc.org`. ①

① The trailing period will not get caught up in the link.

To prevent automatic linking of an URL, prepend it with a backslash (\).

If you prefer URLs to be shown without a visible scheme, set the `hide-uri-scheme` attribute in the document's header.

```
:hide-uri-scheme:  
  
https://asciidoc.org
```

When the `hide-uri-scheme` attribute is set, the above URL will render as follows:

```
<a href="https://asciidoc.org">asciidoc.org</a>
```

Note the absence of `https` inside the `<a>` element.

To attach a URL to text, enclose the text in square brackets at the end of the URL.

Chat with other Fedora users in the `irc://irc.freenode.org/#fedora`[Fedora IRC channel].

Additionally, you can format the linked text.

Ask questions on the `http://discuss.asciidoc.org/[*mailing list*]`.

### *Rendered URLs*

The homepage for the Asciidoc Project is <https://asciidoc.org>.

Chat with other Fedora users in the [Fedora IRC channel](#).

Ask questions on the [mailing list](#).

When a URL does not start with one of the [common schemes](#), or the URL is not surrounded by word boundaries, you must use the `link` macro. The `link` macro is a stronger version of a URI macro, which you can think of like an unconstrained macro. The URL is preceded by `link:` and followed by square brackets. The square brackets may include optional link text. The URL is used for the text of the link if link text is not specified. Prior to 1.5.7, if the `linkattrs` document attribute is set, the text in square brackets is parsed as attributes, which allows a window name or role to be specified. Since 1.5.7, attributes are parsed automatically if an equal sign is found after a comma (e.g., `[link text>window=_blank]`).

### *Anatomy of a link macro*

```
link:url[optional link text, optional target attribute, optional role attribute]
```

Let's consider a case where we need to use the link macro (instead of just a URI macro) to expand a link when it's not adjacent to a word boundary (i.e., unconstrained).

```
search/link:https://ecosia.org[Ecosia]
```

```
search/Ecosia
```

If we didn't use the `link:` prefix in this case, the URL macro would not be detected by the parser.

## Troubleshooting Complex URLs

A URL may not display correctly when it contains characters such as underscores (`_`) or carets (`^`). This problem occurs because the markup parser interprets parts of the URL (i.e., the link target) as valid [text formatting markup](#). Most lightweight markup languages have this issue because they don't use a grammar-based parser. Asciidoctor plans to handle URLs more carefully in the future (see [issue #281](#)), which may be solved by moving to a grammar-based parser (see [issue #61](#)). Thankfully, there are many ways to include URLs of arbitrary complexity using the AsciiDoc passthrough mechanisms.

### Solution A

The simplest way to get a link to behave is to assign it to an attribute.

```
= Document Title  
:link-with-underscores: https://asciidoctor.org/now_this_link_works.html
```

This URL has repeating underscores {link-with-underscores}.

Asciidoctor won't break links with underscores when they are assigned to an attribute because [inline formatting markup is substituted before attributes](#). The URL remains hidden while the rest of the document is being formatted (strong, emphasis, monospace, etc).

### Solution B

Another way to solve formatting glitches is to explicitly specify the formatting you want to have applied to a span of text. This can be done by using the [inline pass macro](#). If you want to display a URL, and have it preserved, put it inside the pass macro and enable the [macros substitution](#), which is what substitutes links.

```
This URL has repeating underscores  
pass:macros[https://asciidoctor.org/now_this_link_works.html].
```

The pass macro removes the URL from the document, applies the `macros` substitution to the URL, and then restores the processed URL to its original location once the substitutions are complete on the whole document.

Alternatively, you can use `++` around the URL only. However, when you use this approach, Asciidoctor won't recognize it as a URL any more, so you have to use the explicit `link` prefix.

```
This URL has repeating underscores  
link:++https://asciidoctor.org/now_this_link_works.html++[].
```

For more information, see [issue #625](#).

Next, we'll add a target and role to a link macro.

Prior to 1.5.7, Asciidoctor *does not* parse attributes in the link macro by default. If you want attributes in the link macro to be parsed, you must set the `linkattrs` document attribute in the header. Since 1.5.7, this parsing is automatic (and the attribute is not required) if an equal sign is found after a comma. When attribute parsing is enabled, you can then specify the name of the target window using the `window` attribute.

= Asciidoctor Document Title

Let's view the raw HTML of the `link:view-source:asciidoctor.org[Asciidoctor homepage,window=_blank]`.

Let's view the raw HTML of the [Asciidoctor homepage](#).

Since `_blank` is the most common window name, we've introduced shorthand for it. Just end the link text with a caret (^):

Let's view the raw HTML of the `link:view-source:asciidoctor.org[Asciidoctor homepage^]`.



If you use the caret syntax more than once in a single paragraph, you may need to escape the first occurrence with a backslash.

When attribute parsing is enabled, you can add a role (i.e., CSS class) to the link.

Chat with other Asciidoctor users on the `http://discuss.asciidoctor.org/[*mailing list*^,role=green]`.

Chat with other Asciidoctor users on the [mailing list](#).



Links with attributes (including the subject and body segments on mailto links) are a feature unique to Asciidoctor. When they're enabled, you must surround the link text in double quotes if it contains a comma.

## 26.1. Link to Relative Files

If you want to link to an external file relative to the current document, use the `link` macro in front of the file name.

```
link:protocol.json[Open the JSON file]
```

If your file is an HTML file, you can link directly to a section in the document, append a hash (#) followed by the section's ID to the end of the file name.

```
link:external.html#livereload[LiveReload]
```

## 26.2. Summary

*Link attributes and values*

Attribute	Value(s)	Example Syntax	Comments
linkattrs		:linkattrs:	Must be set in the header to parse link macro attributes prior to 1.5.7. Since 1.5.7, attributes are parsed automatically if an equal sign is found after a comma (e.g., [link text,window=_blank]).
window	blank	http://discuss.asciidoctor.org[Discuss Asciidoctor,window=_blank]	The blank window target can also be specified using ^ at the end of the link text.
window	^	http://example.org["Google, Yahoo, Bing^"] and http://discuss.asciidoctor.org[Discuss Asciidoctor^]	
role	CSS classes available to inline elements	http://discuss.asciidoctor.org[Discuss Asciidoctor,role=teal]	
id	name of element, custom link text	<<section-title,cross reference text>>	Applies to cross references

# Chapter 27. Cross References

A link to another location within the current AsciiDoc document or in another AsciiDoc document is called a *cross reference* (also referred to as an *xref*). To create a cross reference, you first need to define the location where the reference will point (i.e., the anchor). Then, you need to use one of the forms of the inline xref macro to create a reference to that location. From there, you can customize the text of the reference in various ways.

## 27.1. Automatic Anchors

It's important to understand that many anchors are already defined for you. Using default settings, Asciidoc Doctor automatically creates an anchor for every section and discrete heading. It does so by generating an ID for that section (or discrete heading) and registering that ID in the references catalog. You can then use that ID as the target of an cross reference.

For example, considering the following section.

```
= Section Title
```

Asciidoc Doctor automatically assigns the ID `_section_title` to this section, which you can then use as the target of an xref to create a reference to this section. You can also customize how this ID is generated. Refer to [Auto-generated IDs](#) for more information about how Asciidoc Doctor generates these IDs.

If you're referring to a content element other than a section, you'll need to define an anchor on that element explicitly. Read on to learn how.

## 27.2. Defining an Anchor

An anchor (aka ID) can be defined almost anywhere in the document, including on section title, on a discrete heading, on a paragraph, on an image, on a delimited block, on an inline phrase, and so forth. The anchor is declared by enclosing a *valid XML Name* in double square brackets (e.g., `[[idname]]`) or using the shorthand ID syntax (e.g., `[#idname]`) at the start of an attribute list.

The double square bracket form requires the ID to start with a letter, an underscore, or a colon, ensuring the ID is portable. According to the [XML Name](#) rules, a portable ID may not begin with a number, even though a number is allowed elsewhere in the name. The shorthand form in an attribute list does not impose this restriction.

If you want to reference a block element, all you need to do is assign an ID to that block. You can enclose the ID in double square brackets:

*Assigning an ID to a paragraph*

```
[[notice]]
```

```
This paragraph gets a lot of attention.
```

or use the shorthand ID syntax:

*Assigning an ID to a paragraph using shorthand syntax*

```
[#notice]  
This paragraph gets a lot of attention.
```

You can also define an anchor anywhere in content that receives normal substitutions (specifically the macro substitution). You can enclose the ID in double square brackets:

*Defining an inline anchor*

```
[[bookmark-a]]Inline anchors make arbitrary content referenceable.
```

or using the shorthand ID syntax.

*Defining an inline anchor using shorthand syntax*

```
[#bookmark-b]#Inline anchors can be applied to a phrase like this one.#
```

In addition to being able to define anchors on sections and blocks, anchors can be defined inline where ever you can type normal text (anchors are a macro substitution). The anchors in the text get replaced with invisible anchor points in the output.

For example, you would not put an anchor in front of a list item:

*Wrong position for an anchor ID in front of a list item.*

```
[[anchor-point]]* list item text
```

Instead, you would put it at the very start of the text of the list item:

*Defining an inline anchor on a list item*

```
* Fist item  
* [[step2]]Second item  
* Third item
```

Here's how you can define the ID for a section using an inline anchor:

*Defining the ID of a section using an inline anchor*

```
== Version 4.9 [[version-4_9]]
```

To add additional anchors to a section, place them in front of the title.

```
==== [[current]]Version 4.9 [[version-4_9]]
```



If you add additional anchors to a section title, make sure you also assign an explicit ID to that section. Otherwise, the anchor tag gets caught up in the generated ID. See [#840](#) for details.

Remember that inline anchors are discovered where ever the macro substitution is applied (e.g., paragraph text). If text content doesn't belong somewhere, neither does an inline anchor point.

It's possible to customize the text that will be used in the cross reference link (called `xreflabel`). If not defined, Asciidoctor does it best to find suitable text (the solution differs from case to case). In case of an image, the image caption will be used. In case of a section header, the text of the section's title will be used.

To define the `xreflabel`, add it in the anchor definition right after the ID (separated by a comma).

*An anchor ID with a defined xreflabel. The caption will not be used as link text.*

```
[[tiger-image,Image of a tiger]]  
.This image represents a Bengal tiger also called the Indian tiger  
image::tiger.png[]
```

Instead of the bracket form, you can use the macro `anchor` to achieve the same goal.

*Setting an anchor ID using the macro form*

```
anchor:tiger-image[Image of a tiger]
```

## 27.3. Internal Cross References

In Asciidoctor, the inline `xref` macro is used to create cross references (also called in-text or page citations) to content elements (sections, blocks, or phrases) that have an ID (regardless of whether that ID is explicit or auto-generated).

You create a cross reference by enclosing the ID of the target block or section (or the path of another document with an optional anchor) in double angled brackets.

*Cross reference using the ID of the target section*

```
The section <<images>> describes how to insert images into your document.
```

*Rendered cross reference using the ID of the target section*

```
The section Images describes how to insert images into your document.
```

You can also link to a block or section using the title by referencing its title, referred to as a *natural cross reference*. The title must contain at least one space character or contain at least one uppercase letter. (If you are using Ruby < 2.4, that uppercase letter is restricted to the basic Latin charset).

#### *Cross reference using a section's title*

Refer to <>Internal Cross References>>.

#### *Rendered cross reference using a section's title*

Refer to [Internal Cross References](#).

Converters usually use the reftext of the target as the default text of the link. When the document is parsed, attribute references in the reftext are substituted immediately. When the reftext is displayed, additional reftext substitutions are applied to the text (specialchars, quotes, and replacements).

You can override the reftext of the target by specifying alternative text at the location of the cross reference. After the ID, add a comma and then enter the custom text you want the cross reference to display.

#### *Cross reference with custom xreflabel text*

Learn how to <>link-macro-attributes,use attributes within the link macro>>.

#### *Rendered cross reference using custom xreflabel text*

Learn how to [use attributes within the link macro](#).

You can also use the inline xref macro as an alternative to the double angled bracket form.

#### *Inline xref macro*

Learn how to `xref:link-macro-attributes[use attributes within the link macro]`.

## 27.4. Validating Internal Cross References

Asciidoctor provides limited support for validating internal cross references. Validation occurs when a cross reference is first visited. Since there are still some references aren't stored in the parse tree (such as an anchor in the middle of a paragraph), which can lead to false positives, these validations are hidden behind a flag.

You can enable validation of cross references from the CLI by passing the `-v` and from the API by setting the `$VERBOSE` variable to `true`. This puts the processor in pedantic mode. In this mode, the parser will immediately validate cross references, issuing a warning message if the reference is not valid.

Consider the following example:

```
See <<foobar>>.
```

```
[#foobaz]  
== Foobaz
```

If you run Asciidoctor in verbose/pedantic mode on this document, it will send the following warning message to the logger.

```
asciidoctor: WARNING: invalid reference: foobar
```

Asciidoctor only validates references within the same document (after includes are resolved).

## 27.5. Customizing the Cross Reference Text

Starting in Asciidoctor 1.5.6, when you use one of the native converters (HTML, PDF, and EPUB3), you can customize the style of the automatic cross reference text using the `xrefstyle` document attribute. This customization brings the cross reference text formatting from the DocBook toolchain to the native Asciidoctor converters.

By default, the cross reference text matches the title of the referenced element. For example, if you're linking to a section titled "Installation", the text of the cross reference link appears as:

```
Installation
```

If the `reftext` attribute is specified on the referenced element, that value is preferred over its title. For example, let's assume the section from the previous example was written as:

```
[reftext="Installation Procedure"]  
==== Installation
```

In this case, the text of the cross reference link appears as:

```
Installation Procedure
```

Attribute references are substituted in the `reftext` during parsing and `reftext` substitutions (specialchars, quotes, and replacements) are applied to the value when it's used during conversion.

If the `reftext` is not specified, the text of the cross reference is automatically generated. By default, this text is the title of the reference. There are three built-in styles you can choose from to customize the generated text of a cross reference, as controlled by the `xrefstyle` document attribute.

### **:xrefstyle: full**

Uses the signifier for the reference (e.g., Section) followed by the reference number and emphasized (chapter or appendix) or quoted title (e.g., Section 2.3, “Installation” or Figure 1, “Big Cats”).

### **:xrefstyle: short**

Uses the signifier for the reference (e.g., Section) followed by the reference number (e.g., Section 2.3 or Figure 1).

### **:xrefstyle: basic**

Uses the title only (Installation or Big Cats), applying emphasis if the reference is a chapter or appendix.

This formatting only applies to references that have both a title and number (or explicit caption), but no explicit reftext. If the reference is a chapter or an appendix, the title is displayed in italics instead of quotes (even when the xrefstyle is basic).

Let’s assume you want to reference a section titled “Installation” that has the number 2.3. The **full** style is displayed as:

Section 2.3, “Installation”

The **short** style is displayed as:

Section 2.3

The **basic** style is displayed as:

Installation

The **full** and **short** styles only apply for references that have a caption. Specifically, the corresponding **-caption** attribute must be set for the target’s block type (e.g., listing-caption for listing blocks, example-caption for example blocks, table-caption for tables, etc.). Otherwise, the **\*basic** style is used.

You can use document attributes to customize the signifier that is placed in front of the reference’s number. This *reference signifier* indicates the reference’s type (e.g., Chapter or Section).

- **chapter-refsig**—defines the signifier to use for a cross reference to a chapter (default: Chapter)
- **section-refsig**—defines the signifier to use for a cross reference to a section (default: Section)
- **appendix-refsig**—defines the signifier to use for a cross reference to an appendix (default: Appendix)

(The signifier attribute for a part cross reference will be introduced once numeration is supported for parts).

For example, to customize the word “Section”, define the `section-refsig` attribute in the document header:

```
:section-refsig: Sect.
```

The **full** xrefstyle would then be displayed as:

```
Sect. 2.3, “Installation”
```

The **short** xrefstyle would be displayed as:

```
Sect. 2.3
```

If you unset the attribute, the signifier is dropped from the cross reference text. For example:

```
:!section-refsig:
```

In this case, the **full** xrefstyle will display only the number and title:

```
2.3, “Installation”
```

The **short** xrefstyle will fall back to the number only:

```
2.3
```

The **basic** xrefstyle is unaffected by the value of the signifier.

Only the aforementioned styles are provided out of the box. Support for a custom formatting string is planned. Refer to [#2212](#) for details. Until then, you can implement custom formatting in a custom converter or overriding the `xreftext` method on the node.

## 27.6. Inter-document Cross References

The inline xref macro can also link to IDs in other AsciiDoc documents. This eliminates the need to use direct links between documents that are coupled to a particular converter (e.g., HTML links). It also captures the intent of the author to establish a reference to a section in another document.

Here’s how a cross reference is normally defined in Asciidoc:

```
The section <<images>> describes how to insert images into your document.
```

This cross reference creates a link to the section with the ID *images*.

Let's assume the cross reference is defined in the document *document-a.adoc*. If the target section is in a separate document, *document-b.adoc*, the author may be tempted to write:

```
Refer to link:document-b.html#section-b[Section B] for more information.
```

However, this link is coupled to HTML output. What's worse, if *document-b.adoc* is included in the same document as *document-a.adoc*, the link will refer to a document that doesn't even exist!

These problems can be alleviated by using an inter-document xref:

```
Refer to <>document-b.adoc#section-b,Section B>> for more information.
```

The ID of the target is now placed behind a hash symbol (#). Preceding the hash is the name of the reference document (the file extension is optional). We've also added a label since Asciidoctor cannot (yet) resolve the section title in a separate document.

When Asciidoctor generates the link for this cross reference, it first checks to see if *document-b.adoc* is included in the same document as *document-a.adoc*. If not, it will generate a link to *document-b.html*, intelligently substituting the original file extension with the file extension of the output file.

```
<a href="document-b.html#section-b">Section B</a>
```

If *document-b.adoc* is included in the same document as *document-a.adoc*, then the document will be dropped in the link target and look like the output of a normal cross reference:

```
<a href="#section-b">Section B</a>
```

Now you can create inter-document cross references without the headache.

### 27.6.1. Navigating Between Source Files

In certain environments, such as a source repository or the browser preview extensions, you view the generated HTML through the AsciiDoc source URL. This has consequences for inter-document cross references.

Since the default suffix for relative links in the `html5` backend is `.html`, the inter-document cross references in these environments end up pointing to non-existent HTML files. In this case, you need to change the inter-document cross references to refer to other AsciiDoc source files instead. You can achieve this behavior by setting the `outfilesuffix` attribute to the value as `.adoc`, as the example below shows.

```
= Document Title  
ifdef::env-github,env-browser[:outfilesuffix: .adoc]
```

See the <> README#>, README>>.

We could also write the link as link:README{outfilesuffix}[README].

The links in the generated document will now point to *README.adoc* instead of the default, *README.html*.



This configuration is no longer necessary on GitHub since GitHub now sets the value of `outfilesuffix` to match the file extension of the source file. However, it's still required in GitHub-like environments such as GitLab.



You probably don't want to set `outfilesuffix` to `.adoc` without the `ifdef` condition as it could result in Asciidoctor overwriting input files when you run it locally (though there's some protection against this).

While `outfilesuffix` gives you control over the end of the resolved path for an inter-document cross reference, the `relfileprefix` attribute gives you control over the beginning of the path. When resolving the path of an inter-document cross reference, if the `relfileprefix` attribute is set, the value of this attribute gets prepended to the path. Let's look at an example of when these two attributes are used together.

A common practice in website architecture is to move files into their own folder to make the path format agnostic (called "indexify"). For example, the path *filename.html* becomes *filename/* (which targets *filename/index.html*). However, this is problematic for inter-document cross references. Any cross reference that resolves to the path *filename.html* is now invalid since the file has moved to a subfolder (and thus no longer a sibling of the referencing document).

To solve this problem, you can define the following two attributes:

```
:relfileprefix: ../  
:outfilesuffix: /
```

Now, the cross reference <> filename#,Label>> will resolve to *../filename/* instead of *filename.html*. Since this change is specific to the website architecture described, you want to be sure to only set these attributes in that particular environment (either using an `ifdef` directive or via the API).

# Chapter 28. Include Directive

The include directive provides a way to import content from another file into the current document. The include directive must be placed on a line by itself with the following syntax, which is covered in detail in the next section.

```
include::path[<attrlist>]
```

When the document is processed, the include directive is replaced by the contents of the include file. Think of the include directive like a file expander.

This directive is useful if you want to:

- partition a large document into smaller files (for better organization and to make restructuring simpler),
  - (always separate consecutive include directives by a blank line unless your intent is for the included lines to run together)
- insert snippets of source code (so your examples are kept up-to-date with the latest source files),
- populate tables with output from other programs (e.g., CSV data),
- create document variants by combining the include directive with preprocessor conditionals (e.g., `ifdef`), and
- reuse fragments and boilerplate content multiple times within the same document.

You can use the include directive as many times as you want in the same document.



The include directive is *disabled* when Asciidoctor is run in secure mode. In this mode, the include directive is converted to a link in the output document. To learn more about secure mode, refer to the section [Running Asciidoctor Securely](#).

## 28.1. Anatomy

The include directive has the following anatomy:

```
include::path[leveloffset=<em>offset</em>,lines=<em>ranges</em>,tag(s)=<em>name(s)</em>,indent=<em>depth</em>,opts=optional]
```

The leveloffset, lines, tag(s), indent, and opt attributes are optional, making the simplest case look like:

```
include::content.adoc[]
```

The sections that follow go into detail about when the include is processed, how the include file is resolved, and how each attribute is used.

## 28.2. Processing

Although the include directive looks like a block macro, it's not a macro and therefore not processed like one. Instead, it's a *preprocessor* directive, just like `ifdef` and `ifeval`. It's important to understand the distinction.

A preprocessor directive is processed when the lines are read, but before the document structure is parsed. Therefore, it's *not* aware of the surrounding document structure. A preprocessor directive merely adds lines to the reader or takes lines away. The include directive is a preprocessor directive that always adds lines.

The best way to think of the include directive is to imagine that it is being replaced by the lines from the include file (i.e., the imported lines). Only then does the parser read and interpret those lines. That's also why it's important to surround the include directive by blank lines if it imports in a discrete structure. You only want to place include files directly adjacent to one another if the imported content should be directly adjacent.

If you don't want the include directive to be processed, you must escape it using a backslash.

```
\include::just-an-example.ext[]
```

Escaping the directive is necessary *even if it appears in a verbatim block* since it's not aware of the surrounding document structure.

## 28.3. File resolution

The path used in an include directive may be either relative or absolute.

If the path relative, the processor resolves the path using the following rules:

- If the include directive is used in the main (top-level) document, relative paths are resolved relative to the base directory. (The base directory defaults to the directory of the main document and can be overridden from the CLI or API).
- If the include directive is used in a file that has itself been included, the path is resolved relative to the including (i.e., current) file.

These defaults makes it easy to reason about how the path to the include file is resolved.

If the processor cannot locate the file (perhaps because you mistyped the path), you'll still be able to convert the document. However, you will get the following warning message during conversion:

```
asciidoctor: WARNING: master.adoc: line 3: include file not found: /.../content.adoc
```

The following message will also be inserted into the output:

Unresolved directive in master.adoc - include::content.adoc[]

To fix the problem, edit the file path and run the converter again.



If you want Asciidoctor not to trigger a warning when the target is missing, set the optional option on the include directive (e.g., `opts=optional`).

If you store your AsciiDoc files in nested folders at different levels, relative file paths can quickly become awkward and inflexible. A common pattern to help here is to define the paths in attributes defined in the header, then prefix all include paths with a reference to one of these attributes:

```
:includedir: _includes  
:sourcedir: ../src/main/java  
  
include::{includedir}/fragment1.adoc[]  
  
[source,java]  
----  
include::{sourcedir}/org/asciidoctor/Asciidoctor.java[]  
----
```

Keep in mind that no matter how Asciidoctor resolves the path to the file, access to that file is limited by the safe mode setting under which Asciidoctor is run. If a path violates the security restrictions, it may be truncated.

## 28.4. Partitioning large documents and using leveloffset

When your document gets large, you can split it up into subsections for easier editing as follows:

```
= My book  
  
include::chapter01.adoc[]  
  
include::chapter02.adoc[]  
  
include::chapter03.adoc[]
```

Take note of the blank lines between the include directives. The blank line between include directives prevents the first and last lines of the included files from being adjoined. This practice is **strongly** encouraged when combining document parts. If you don't include these blank lines, you might find that the AsciiDoc processor swallows section titles. This happens because the leading section title can get interpreted as the last line of the final paragraph in the preceding include. Only place include directives on consecutive lines if the intent is for the includes to run together (such as in a listing block).



The leveloffset attribute can help here by pushing all headings in the included document down by the specified number of levels. This allows you to publish each chapter as a standalone document (complete with a document title), but still be able to include the chapters into a master document (which has its own document title).

You can easily assemble your book so that the chapter document titles become level 1 headings using:

```
= My Book  
  
include::chapter01.adoc[leveloffset=+1]  
  
include::chapter02.adoc[leveloffset=+1]  
  
include::chapter03.adoc[leveloffset=+1]
```

Because the leveloffset is *relative* (it begins with + or -), this works even if the included document has its own includes and leveloffsets.

If you have lots of chapters to include and want them all to have the same offset, you can save some typing by setting leveloffset around the includes:

```
= My book  
  
:leveloffset: +1  
  
include::chapter01.adoc[]  
  
include::chapter02.adoc[]  
  
include::chapter03.adoc[]  
  
:leveloffset: -1
```

The final line returns the leveloffset to 0.

Alternatively, you could use absolute levels:

```
:leveloffset: 1  
  
//includes  
  
:leveloffset: 0
```

Relative levels are preferred. Absolute levels become awkward when you have nested includes since they aren't context aware.

## 28.5. AsciiDoc vs non-AsciiDoc files

The include directive performs a simple file merge, so it works with any text file. The content of all included content is normalized. This means that the encoding is forced to UTF-8 (or converted from UTF-16 to UTF-8 if the file contains a BOM) and trailing whitespace and endlines are removed from each line and replaced with a Unix line feed. This normalization is important to how Asciidoctor works.

If the file is recognized as an AsciiDoc file (i.e., it has one of the following extensions: `.asciidoc`, `.adoc`, `.ad`, `.asc`, or `.txt`), Asciidoctor runs the preprocessor on the lines, looking for and interpreting the following directives:

- includes
- preprocessor conditionals (e.g., `ifdef`)

This allows includes to be nested, and provides lot of flexibility in constructing radically different documents with a single master document and a few command line attributes.

Including non-AsciiDoc files is normally done to merge output from other programs or populate table data:

```
.2016 Sales Results  
,====  
include::sales/2016/results.csv[]  
,====
```

In this case, the include directive does not do any processing of AsciiDoc directives. The content is inserted as is (after being normalized).

## 28.6. Select Portions of a Document to Include

The include directive enables you to select portions of a file to include instead of including the whole file. Use the `lines` attribute to include individual lines or a range of lines (by line number). Use the `tags` attribute (or `tag` attribute for the singular case) to select lines that fall between regions marked with user-defined tags.

When including multiple line ranges or multiple tags, each entry in the list must be separated by

either a comma or a semi-colon. If commas are used, the entire value must be enclosed in quotes. You can eliminate this requirement by using the semi-colon as the data separator instead.

## 28.6.1. By tagged regions

Tags are useful when you want to identify specific regions of a file to include. You can then select the lines between the boundaries of the `include`/`end` directives to include using the `tags` attribute.



If the target file has tagged lines, and you just want to ignore those lines, use the `tags` attribute to filter them out. See [Tag filtering](#) for details.

The example below shows how you tag a region of content inside a file containing multiple code examples.

*Tagged code snippets in a file named core.rb*

```
# tag::timings[] ① ②
if timings
    timings.record :read
    timings.start :parse
end
# end::timings[] ③ ④
# tag::parse[] ⑤
doc = (options[:parse] == false ? (Document.new_lines, options) :
       (Document.new_lines,options).parse)
timings.record :parse if timings
doc
# end::parse[] ⑥
```

① To indicate the start of a tagged region, insert a comment line in the code.

② Assign a unique name to the `tag` directive. In this example the tag is called *timings*.

③ Insert another comment line where you want the tagged region to end.

④ Assign the name of the region you want to terminate to the `end` directive.

⑤ This is the start a tagged snippet named *parse*.

⑥ This is the end of the tagged snippet named *parse*.



The `tag::[]` and `end::[]` directives should be placed after a line comment as defined by the language of the source file. The directives must also appear at the *end* of the line. In the previous example, we choose to prefix the lines with a hash (#) because that's the start of a line comment in Ruby.



For languages that only have circumfix comments, such as XML, you can enclose the tag and end directives in the respective circumfix comment markers. For example, in XML files, you can use `<!-- tag::name[] -->` and `<!-- end::name[] -->` (the spaces around the tag are required).

In the next example, the tagged region named *parse* is selected by the `include` directive.

#### *Selecting the parse code snippet from a document*

```
[source,ruby]
-----
include::core.rb[tag=parse] ①
-----
```

- ① In the directive's brackets, set the `tag` attribute and assign it the unique name of the code snippet you tagged in your code file.

You can include multiple tags from the same file.

#### *Selecting the timings and the parse code snippets from a document*

```
[source,ruby]
-----
include::core.rb[tags=timings;parse]
-----
```

It is also possible to have fine-grained tagged regions inside larger tagged regions.

For example, if your include file has the following content:

```
// tag::snippets[]
// tag::snippet-a[]
snippet a
// end::snippet-a[]

// tag::snippet-b[]
snippet b
// end::snippet-b[]
// end::snippets[]
```

And you include this file using the following include directive:

```
include::file-with-snippets.adoc[tag=snippets]
```

The following lines will be selected and displayed:

```
snippet a

snippet b
```

Notice that none of the lines with the tag directives are displayed.

## 28.6.2. Tag filtering

The previous section showed how to select tagged regions explicitly, but you can also use wildcards and exclusions. These expressions give you the ability to include or exclude tags in bulk. Even if you're not including by tag, you can use the double wildcard (\*\*\*) to ignore all lines with an include tag/end directive.

The modifiers you can use for filtering are as follows:

\*

Select all tagged regions.

\*\*

Select all the lines in the document **except for lines with an include tag/end directive**. Use this symbol if you want to include a file that has tag/end directives, but you want to ignore all those lines.

!

Negate the match.

The wildcards are applied first, then the exclusions are applied. Technically, the order you list them doesn't matter, but it's customary to list them in this order.

Here are some of the permutations that you can use:

- \*\*\* — selects all the lines in the document; matches the default behavior of the include directive, except lines containing a tag directive; shorthand for \*\*\*; \* when used alone
- \* — selects all tagged regions in the document.
- \*\*; \* — selects all the lines outside and inside of tagged regions, but not the lines containing a tag directive.
- `foo;!bar` — selects regions tagged *foo*, but excludes any nested regions tagged *bar*.
- \*; !*foo* — selects all tagged regions, but excludes any regions tagged *foo* (nested or otherwise).
- \*\*; !*foo* — selects all the lines of the document except for regions tagged *foo*.
- \*\*; !\* — selects only the regions of the document outside of tags (i.e., non-tagged regions).

There are also some shorthands if only exclusions are specified:

- !*foo* — equivalent to \*\*; !*foo*; an exclusion without an inclusion implicitly starts by selecting all the lines.
- !\* — equivalent to \*\*; !\*; only selects regions that are not tagged.

## 28.6.3. By line ranges

To include content by line range, assign a starting line number and an ending line number separated by a pair of dots (e.g., `lines=1..5`) to the `lines` attribute.

```
include::filename.txt[lines=5..10]
```

You can specify multiple ranges by separating each range by a comma. Since commas are normally used to separate individual attributes, you must quote the comma-separated list of ranges.

```
include::filename.txt[lines="1..10,15..20"]
```

To avoid having to quote the list of ranges, you can instead separate them using semi-colons.

```
include::filename.txt[lines=7;14..25;28..43]
```

If you don't know the number of lines in the document, or you don't want to couple the range to the length of the file, you can refer to the last line of the document using the value -1.

```
include::filename.txt[lines=12..-1]
```

## 28.7. Normalize Block Indentation

Source code snippets from external files are often padded with a leading block indent. This leading block indent is relevant in its original context. However, once inside the documentation, this leading block indent is no longer needed.

The attribute `indent` allows the leading block indent to be stripped and, optionally, a new block indent to be set for blocks with verbatim content (listing, literal, source, verse, etc.).

- When `indent` is 0, the leading block indent is stripped
- When `indent` is > 0, the leading block indent is first stripped, then the content is indented by the number of columns equal to this value.

If the `tabsize` attribute is set on the block or the document, tabs are also replaced with the number of spaces specified by that attribute, regardless of whether the `indent` attribute is set.

For example, this AsciiDoc source:

```
[source,ruby,indent=0]
-----
def names
  @name.split ' '
end
-----
```

Produces:

```
def names
  @name.split ''
end
```

This AsciiDoc source:

```
[indent=2]
-----
def names
  @name.split ''
end
-----
```

Produces:

```
def names
  @name.split ''
end
```

## 28.8. Include Content from a URI

The include directive recognizes when the target is a URI and can include the content referenced by that URI.

### *Including content from a URI*

This example demonstrates how to include an AsciiDoc file from a GitHub repo directly into your document.

```
include::https://raw.githubusercontent.com/asciidoc/asciidoc/master/README.adoc[]
```

For security reasons, this capability is *not enabled* by default. To allow content to be read from a URI, you must enable the URI read permission by:

1. running Asciidoctor in **SERVER** mode or less and
2. setting the **allow-uri-read** attribute securely (i.e., from the CLI or API).

Here's an example that shows how to run Asciidoctor from the console so it can read content from a URI:

```
$ asciidoctor -a allow-uri-read filename.adoc
```

Remember that Asciidoctor executes in SAFE mode by default when run from the command line.

Here's an example that shows how to run Asciidoctor from the API so it can read content from a URI:

```
Asciidoctor.convert_file 'filename.adoc', safe: :safe, attributes: { 'allow-uri-read' => '' }
```



Including content from sources outside your control carries certain risks, including the potential to introduce malicious behavior into your documentation. Because `allow-uri-read` is a potentially dangerous feature, it is forcefully disabled when the safe mode is `SECURE` or higher.

## URI vs URL

URI stands for [Uniform Resource Identifier](#). When we talk about a URI, we're usually talking about a URL, or Uniform Resource Locator. A URL is simply a URI that points to a resource over a network, or web address.

As far as Asciidoctor is concerned, all URIs share the same restriction, whether or not it's actually local or remote, or whether it points to a web address (`http` or `https` prefix), FTP address (`ftp` prefix), or some other addressing scheme.

The same restriction described in this section applies when embedding an image referenced from a URI, such as when `data-uri` is set or when converting to PDF using Asciidoctor PDF.

## 28.9. Caching URI Content

Reading content from a URI is obviously much slower than reading it from a local file.

Asciidoctor provides a way for the content read from a URI to be cached, which is highly recommended.

To enable the built-in cache, you must:

1. Install the `open-uri-cached` gem.
2. Set the `cache-uri` attribute in the document.

When these two conditions are satisfied, Asciidoctor caches content read from a URI according to [HTTP caching recommendations](#).

## 28.10. Include a File Multiple Times in the Same Document

A document can include the same file any number of times. The problem comes if there are IDs in

the included file; the output document (HTML or DocBook) will then have duplicate IDs which will make it not well-formed. To fix this, you can reference a dynamic variable from the main document in the ID.

For example, let's say you want to include the same subsection describing a bike chain in both the operation and maintenance chapters:

```
= Bike Manual  
  
:chapter: operation  
== Operation  
  
include::fragment-chain.adoc[]  
  
:chapter: maintenance  
== Maintenance  
  
include::fragment-chain.adoc[]
```

Write *fragment-chain.adoc* as:

```
[id='chain-{chapter}']  
==== Chain  
  
See xref:chain-{chapter}[].
```

The first time the *fragment-chain.adoc* file is included, the ID of the included section resolves to **chain-operation**. The second time the file included, the ID resolves to **chain-maintenance**.

In order for this to work, you must use the long-hand forms of both the ID assignment and the cross-reference. The single quotes around the variable name in the assignment are required to force variable substitution (aka interpolation).

## 28.11. Using an Include in a List Item

You can use the `include` directive to include the content of a list item from another file, but there are some things you need to be aware of.

Recall that the `include` directive must be defined on a line by itself. This presents a challenge with lists since each list item must begin with the list marker. We can solve this by using the built-in `blank` attribute to initiate the list item, then follow that line with the `include` directive to bring in the actual content.

Here's an example of how to use the `blank` attribute and the `include` directive to define a list item, then include the primary text from another file:

```
* {blank}
include::item-text.adoc[]
```

This technique works well if you control the contents of the included file and can ensure it only contains adjacent lines of text. If a list item does not contain adjacent lines, the list may be terminated. So we need a bit more syntax.

If you can't guarantee that all the included lines will be adjacent, you'll want to tuck the include directive inside an open block. This keeps all the include lines together, enclosed inside the boundaries of the block. You then attach this block to the list item using a [list continuation](#) (i.e., `+`).

Here's an example of how to include complex content from another file into a list item:

```
* {blank}
+
--
include::complex-list-item.adoc[]
--
```

See [\[#dropping-the-principal-text\]](#) for another example of this technique.

# Chapter 29. Images

To include an image on its own line (i.e., a *block image*), use the `image::` prefix in front of the file name and square brackets after it.

```
image::sunset.jpg[]
```



If you want to specify alt text, include it inside the square brackets:

```
image::sunset.jpg[Sunset]
```

You can also give the image an id, a title, set its dimensions and make it a link.

```
[#img-sunset] ①
.A mountain sunset ②
[link=https://www.flickr.com/photos/javh/5448336655] ③
image::sunset.jpg[Sunset,300,200] ④ ⑤
```

① Assigns an ID to the block; see [Defining an Anchor](#).

② Defines the title of the block image, which gets displayed underneath the image when rendered.

③ `link` makes the image a link (this can also be defined inside the attribute list of the block macro)

④ The first positional attribute, *Sunset*, is the image's alt text.

⑤ The second and third positional attributes define the width and height, respectively.

A hyperlinked image with caption



Figure 1: A mountain sunset

Block images are prefixed by a caption label (Figure) and number automatically. To turn off figure caption labels and numbers, add the `figure-caption` attribute to the document header and unset it.

```
:figure-caption!: 
```

If you want to include an image inline, use the `image:` prefix instead (notice there is only one colon):

Click `image:icons/play.png[Play, title="Play"]` to get the party started.

Click `image:icons/pause.png[title="Pause"]` when you need a break.

Click to get the party started.

Click when you need a break.

For inline images, the optional title is displayed as a tooltip.

## 29.1. Setting the Location of Images

The path to the location of images is controlled by the `imagesdir` document attribute. The value of this attribute (empty by default) is added to the beginning of *every* image target. Therefore, you never need to reference this attribute. You only need to set it.

*Incorrect*

```
image::{imagesdir}/name-of-image.png[]
```

*Correct*

```
image::name-of-image.png[]
```

The resolved location of the image will be: <imagesdir> + <target>.

The benefit of the processor adding the value of the `imagesdir` attribute to the start of all image targets is that you can globally control the folder where images are located per converter. We refer to this folder as the image catalog. Since different output formats require the images to be in different locations, this attribute makes it possible to accommodate many different scenarios.

We recommend relying on the `imagesdir` attribute when defining the target of your image to avoid hard-coding that common path in every single image macro. Always think about where the image is relative to the image catalog.

The `imagesdir` attribute value can be an absolute path, relative path or URI. By default, the value of the `imagesdir` attribute is empty, which means these images are resolved relative to the document. If the image target is an absolute path or URI, the `imagesdir` prefix is *not* added to the path.



You can set the `imagesdir` attribute in multiple places in your document, as long as it is not locked by the API. This technique is useful if you store images for different parts, chapters, or sections of your document in different locations.

### 29.1.1. Include Images by Full URL

Asciidoctor supports remote (i.e., images with a URL target) block and inline images. You can reference images served from any URL (e.g., your blog, an image hosting service, your docs server, etc.) and never have to worry about downloading the images and putting them somewhere locally.

Here are a few examples of images that have a URL target:

*Block image with a URL target*

```
image::https://upload.wikimedia.org/wikipedia/commons/3/35/Tux.svg[Tux,250,350]
```

[Tux] | <https://upload.wikimedia.org/wikipedia/commons/3/35/Tux.svg>

*Inline image with a URL target*

You can find

```
image:https://upload.wikimedia.org/wikipedia/commons/3/35/Tux.svg[Linux,25,35]
```

everywhere these days.

You can find [Linux] everywhere these days.



The value of `imagesdir` is ignored when the image target is a URI.

If you want to avoid typing the URL prefix for every image, and all the images are located on the same server, you can use the `imagesdir` attribute to set the base URL:

## Using a URL as the base URL for images

```
:imagesdir-old: {imagesdir}  
:imagesdir: https://upload.wikimedia.org/wikipedia/commons  
  
image::3/35/Tux.svg[Tux,250,350]  
  
:imagesdir: {imagesdir-old}
```

This time, the `imagesdir` is used since the image target is not a URL (the `imagesdir` just happens to be one).



This feature is included in the AsciiDoc compatibility file so that AsciiDoc gets it right too.

## 29.2. Putting Images in Their Place

Images are a great way to enhance the text, whether to illustrate an idea, show rather than tell, or just help the reader connect with the text.

Out of the box, images and text behave like oil and water. Images don't like to share space with text. They are kind of "pushy" about it. That's why we focused on tuning the controls in the image macros so you can get the images and the text to flow together.

There are two approaches you can take when positioning your images:

1. Named attributes
2. Roles

### 29.2.1. Positioning attributes

Asciidoctor supports the `align` attribute on block images to align the image within the block (e.g., left, right or center). The named attribute `float` can be applied to both the block and inline image macros. Float pulls the image to one side of the page or the other and wraps block or inline content around it, respectively.

Here's an example of a floating block image. The paragraphs or other blocks that follow the image will float up into the available space next to the image. The image will also be positioned horizontally in the center of the image block.

*A block image pulled to the right and centered within the block*

```
image::tiger.png[Tiger,200,200,float="right",align="center"]
```

Here's an example of a floating inline image. The image will float into the upper-right corner of the paragraph text.

An inline image pulled to the right of the paragraph text

```
image:linux.png[Linux,150,150,float="right"]  
You can find Linux everywhere these days!
```

When you use the named attributes, CSS gets added inline (e.g., `style="float: left"`). That's bad practice because it can make the page harder to style when you want to customize the theme. It's far better to use CSS classes for these sorts of things, which map to roles in AsciiDoc terminology.

## 29.2.2. Positioning roles

Here are the examples from above, now configured to use roles that map to CSS classes in the default Asciidoctor stylesheet:

*Block image macro using positioning roles*

```
[.right.text-center]  
image::tiger.png[Tiger,200,200]
```

*Inline image macro using positioning role*

```
image:sunset.jpg[Sunset,150,150,role="right"] What a beautiful sunset!
```

The following table lists all the roles available out of the box for positioning images.

*Roles for positioning images*

Role	Float		Align		
	left	right	text-left	text-right	text-center
Block Image	✓	✓	✓	✓	✓
Inline Image	✓	✓	✗	✗	✗

Merely setting the float direction on an image is not sufficient for proper positioning. That's because, by default, no space is left between the image and the text. To alleviate this problem, we've added sensible margins to images that use either the positioning named attributes or roles.

If you want to customize the image styles, perhaps to customize the margins, you can provide your own additions to the stylesheet (either by using your own stylesheet that builds on the default stylesheet or by adding the styles to a docinfo file).



The shorthand syntax for a role `(.)` can not yet be used with image styles.

## 29.2.3. Framing roles

It's common to frame the image in a border to further offset it from the text. You can style any block or inline image to appear as a thumbnail using the `thumb` role (or `th` for short).



The **thumb** role doesn't alter the dimensions of the image. For that, you need to assign the image a height and width.

Here's a common example for adding an image to a blog post. The image floats to the right and is framed to make it stand out more from the text.

```
image:logo.png[role="related thumb right"] Here's text that will wrap around the image to the left.
```

Notice we added the **related** role to the image. This role isn't technically required, but it gives the image semantic meaning.

#### 29.2.4. Control the float

When you start floating images, you may discover that too much content is floating around the image. What you need is a way to clear the float. That is provided using another role, **float-group**.

Let's assume that we've floated two images so that they are positioned next to each other and we want the next paragraph to appear below them.

```
[.left]
.Image A
image::a.png[A,240,180]

[.left]
.Image B
image::b.png[B,240,180,title="Image B"]

Text below images.
```

When this example is converted, then viewed a browser, the paragraph text appears to the right of the images. To fix this behavior, you just need to "group" the images together in a block with self-contained floats. Here's how it's done:

```
[.float-group]
--
[.left]
.Image A
image::a.png[A,240,180]

[.left]
.Image B
image::b.png[B,240,180]
--

Text below images.
```

This time, the text will appear below the images where we want it.

## 29.3. Sizing Images

Since images often need to be sized according to the medium, there are several ways to specify an image size.

In most output formats, the specified width is obeyed unless the image would exceed the content width or height, in which case it scaled to fit while maintaining the original aspect ratio (i.e., responsive scaling).

### 29.3.1. width and height

The primary way to specify the size of an image is to define the `width` and `height` attributes on the image macro. Since these two attributes are so common, they are the second and third (unnamed) positional attributes on the image macros.

```
image::flower.jpg[Flower,640,480]
```

That's equivalent to the long-hand version:

```
image::flower.jpg[alt=Flower,width=640,height=480]
```

While the values of `width` and `height` can be used to scale the image, these attributes are primarily intended to specify the intrinsic (or faux-intrinsic, aka too-lazy-to-resize) size of the image in CSS pixels.<sup>[1]</sup> The `width` and `height` attributes are mapped to attributes of the same name on the `<img>` element in the HTML output. These attributes are important because they provide a hint to the browser to tell it how much space to reserve for the image during layout to minimize page reflows.

### Automatic image scaling

The default Asciidoctor stylesheet implements responsive images (using width-wise scaling). If the width of the screen is smaller than the width of the image, the image will be scaled down to fit. To support this feature, the original aspect ratio of the image is preserved at all sizes. Thus, when you set the dimensions, the values should reflect the original aspect ratio of the image as this will be enforced. If the values don't match the aspect ratio, the height is ignored by the browser.

### 29.3.2. pdfwidth

Asciidoctor recognizes the following attributes to size images when converting to PDF using Asciidoctor PDF:

- `pdfwidth` - The preferred width of the image in the PDF when converting using Asciidoctor PDF.

The `pdfwidth` attribute accepts the following units:

**px**

Output device pixels (assumed to be 96 dpi)

**pt (or none)**

Points (1/72 of an inch)

**pc**

Picas (1/6 of an inch)

**cm**

Centimeters

**mm**

Millimeters

**in**

Inches

**%**

Percentage of the content width (area between margins)

**vw**

Percentage of the page width (edge to edge)

If `pdfwidth` is not provided, Asciidoctor PDF also accepts `scaledwidth`, or `width` (no units, assumed to be pixels), in that order.

See [image scaling in Asciidoctor PDF](#) for more details.

### 29.3.3. `scaledwidth`

Asciidoctor recognizes the following attributes to size images when converting to PDF using the DocBook toolchain:

- `scaledwidth` - The preferred width of the image when converting to PDF using the DocBook toolchain. (mutually exclusive with `scale`)
- `scale` - Scales the original image size by this amount when converting to PDF using the DocBook toolchain (mutually exclusive with `scaledwidth`).

`scaledwidth` sizes images much like `pdfwidth`, except it does not accept the `vw` unit.

The value of `scaledwidth` when used with DocBook can have the following units:

**px**

Output device pixels (assumed to be 72 dpi)

**pt**

Points (1/72 of an inch)

**pc**

Picas (1/6 of an inch)

**cm**

Centimeters

**mm**

Millimeters

**in**

Inches

**em**

Ems (current font size)

**% (or none)**

Percentage of intrinsic image size

DocBook also accepts the `width` attribute if `scaledwidth` is not provided.

### 29.3.4. Image Sizing Recap

*Image sizing attributes*

Backend	Absolute size	Relative to original size	Relative to content width	Relative to page width
html	width=120 (assumed to be px)	Not possible	width=50%	Not possible
pdf	pdfwidth=100mm (or cm, in, pc, pt, px)	Not possible	pdfwidth=80%	pdfwidth=50vw
docbook	scaledwidth=100mm (or cm, em, in, pc, pt, px)	scale=80	scaledwidth=50%	Not possible

Here's an example of how you might bring these attributes together to control the size of an image in various output formats:

```
image::flower.jpg[Flower,640,480, pdfwidth=50%, scaledwidth=50%]
```

A practice you might consider is using attributes to set the values for each output:

```

ifdef::backend-html5[]
:twoinches: width='144'
:full-width: width='100%'
:half-width: width='50%'
:half-size:
:thumbnail: width='60'
endif::[]
ifdef::backend-pdf[]
:twoinches: pdfwidth='2in'
:full-width: pdfwidth='100vw'
:half-width: pdfwidth='50vw'
:half-size: pdfwidth='50%'
:thumbnail: pdfwidth='20mm'
endif::[]
ifdef::backend-docbook5[]
:twoinches: width='50mm'
:full-width: scaledwidth='100%'
:half-width: scaledwidth='50%'
:half-size: width='50%'
:thumbnail: width='20mm'
endif::[]

```

Then you can specify a half-size image using:

```
image::image.jpg[{half-size}]
```

In addition to providing consistency across your document, this technique will help insulate you from future changes. For a more detailed example, see [this thread](#) on the discussion list.

## 29.4. Taming SVGs

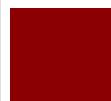
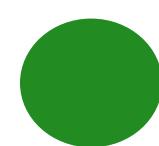
Both block and inline image macros have built-in support for scalable vector graphics (SVGs). But there's more than one way to include an SVG into a web page, and the strategy used can effect how the SVG behaves (or misbehaves). Therefore, these macros provide additional options to control how the SVG is included (i.e., referenced).

When the image target is an SVG, the `options` attribute on the macro accepts one of three values:

- `none` (default)
- `interactive`
- `inline`

The following table demonstrates the impact these options have.

*Demonstration of option values for SVG images*

<code>image::sample.svg[Static,300]</code>			
<code>image::sample.svg[Interactive,300,opts=interactive]</code>			
Observe that the color changes when hovering over the SVG.			
<code>image::sample.svg[Embedded,300,opts=inline]</code>			
Observe that the color changes when hovering over the SVG. The SVG also inherits CSS from the document stylesheets.			

How the options value works and when it should be used is described below:

#### *Option values for SVG images (HTML output)*

Option	HTML Element Used	Effect	When To Use
<code>none</code> (default)	<code>&lt;img&gt;</code>	Image is rasterized	Static image, no interactivity, no custom fonts
<code>interactive</code>	<code>&lt;object&gt;</code>	Image embedded as a live, interactive object (aka “content document”)	For using CSS animations, scripting, webfonts
<code>inline</code>	<code>&lt;svg&gt;</code>	The SVG is embedded directly into the HTML itself.	For using CSS animations, scripting, webfonts, when you require search engines to search the SVG content.  To allow SVG content reachable by JavaScript in the main DOM or to inherit styles from the main DOM.

When using the `inline` or `interactive` options, the `viewBox` attribute must be defined on the root `<svg>` element in order for scaling to work properly.

When using the `inline` option, if you specify a width or height on the image macro in AsciiDoc, the

`width`, `height` and `style` attributes on the `<svg>` element will be removed. Additionally, when using `inline` the primary SVG elements (e.g., `<svg>`) cannot have a namespace.

If using the `interactive` option, you must link to the CSS that declares the fonts in the SVG file using an XML stylesheet declaration.

If you're inserting an SVG using either the `inline` or `interactive` options, we strongly recommend you optimize your SVG using a tool like [svgo](#) or [SVG Editor](#).

As you work with SVG, you'll become more comfortable making the decision about which method to employ given the circumstances. It's only confusing when you first encounter the choice. To learn more about using SVG on the web, consult the online book [SVG on the Web: A Practical Guide](#) as well as [these articles about SVG](#).

## 29.5. Summary

*Block and inline image attributes and values*

Attribute	Value(s)	Example Syntax	Comments
<code>imagesdir</code>	empty, absolute path, relative path or base URL	<code>:imagesdir: media</code>	If the image target is a URL or an absolute path, the <code>imagesdir</code> prefix is <i>not</i> added; Default is empty value
<code>id</code>	User defined text	<code>id=sunset-img</code> (or <code>[[sunset-img]]</code> or <code>#[sunset-img]</code> above block macro)	
<code>alt</code>	User defined text in first position of attribute list or named attribute	<code>image::sunset.jpg[Brilliant sunset]</code> (or <code>alt=Sunset</code> )	
<code>title</code>	User defined text	<code>title="A mountain sunset"</code> (or <code>.A mountain sunset</code> above block macro)	Blocks: title displayed below image; Inline: title displayed as tooltip
<code>caption</code>	User defined text	<code>caption="Figure 8: "</code>	Only applies to block images.
<code>width</code>	User defined size in pixels	<code>image::sunset.jpg[Sunset,300]</code> (or <code>width=300</code> )	
<code>height</code>	User defined size in pixels	<code>image::sunset.jpg[Sunset,300,2 00]</code> (or <code>height=200</code> )	
<code>link</code>	User defined location of external URI	<code>link=https://www.flickr.com/photos/javh/5448336655</code>	

Attribute	Value(s)	Example Syntax	Comments
<code>scale</code>	A scaling factor to apply to the intrinsic image dimensions	<code>scale=80</code>	DocBook only
<code>scaledwidth</code>	User defined width for block images	<code>scaledwidth=25%</code>	DocBook only
<code>pdfwidth</code>	User defined width for images in a PDF	<code>pdfwidth=80vw</code>	Asciidoctor PDF only
<code>align</code>	left, center, right	<code>align=left</code>	Block images only; Align and float attributes are mutually exclusive
<code>float</code>	left, right	<code>float=right</code>	Block images only; float and align attributes are mutually exclusive; to scope the float, use a <a href="#">float group</a> .
<code>role</code>	left, right, th, thumb, related, rel	<code>role="thumb right"</code> (or <code>[.thumb.right]</code> above block macro)	Inline images can use role to float images left and right; Role shorthand <code>(.)</code> can not be used on images

[1] See <https://www.w3.org/TR/2014/REC-html5-20141028/embedded-content-0.html#dimension-attributes>

# Chapter 30. Video

The block video macro enables you to embed videos into your documentation. You can embed self-hosted videos or videos shared on popular video hosting sites such as YouTube and Vimeo.

The video formats Asciidoctor supports is dictated by the formats supported by the browser (and, in turn, the user's system). While this was once a precarious ordeal, HTML5 has brought sanity to video support in the browser by adding a dedicated `<video>` element and by introducing several standard video formats. Those formats are now widely supported across browsers and systems.

For a canonical list of supported web video formats and their interaction with modern browsers, see the [Mozilla Developer Supported Media Formats](#) documentation.

## A recommendation for serving video to browsers

Where appropriate, we recommend using a video hosting service like YouTube or Vimeo to serve videos in online documentation. These services specialize in streaming optimized video to the browser, with the lowest latency possible given hardware, software, and network capabilities of the device viewing the video.

Vimeo even offers a white label mode so users aren't made aware that the video is being served through its service.

See [YouTube and Vimeo videos](#) for details about how to serve videos from these services.

*Basic video file include*

```
video::video_file.mp4[]
```

You can control the video settings using additional attributes on the macro. For instance, you can offset the start time of playback using the `start` attribute and enable autoplay using the `autoplay` option.

*Setting attributes for local video playback*

```
video::video_file.mp4[width=640, start=60, end=140, options=autoplay]
```

You can include a caption on the video using the `title` attribute.

*Adding a caption to a video*

```
.A walkthrough of the product  
video::video_file.mp4[]
```

## 30.1. YouTube and Vimeo videos

The video macro supports embedding videos from external video hosting services like YouTube and Vimeo. Asciidoctor automatically generates the correct code to embed the video in the HTML output.

To use this feature, put the video ID in the macro target and the name of the hosting service in the first positional attribute.

*Embedding a YouTube video*

```
video::rPQoq7ThGAU[youtube]
```

*Embedding a Vimeo video*

```
video::67480300[vimeo]
```

## 30.2. Supported Attributes

*Video attributes and values*

Attribute	Value(s)	Example Syntax	Comments
title	User defined text	.An ocean sunset	
poster	A URL to an image to show until the user plays or seeks.	poster=sunset.jpg	Can be specified as the first positional (unnamed) attribute. Also used to specify the service when referring to a video hosted on YouTube (youtube) or Vimeo (vimeo).
width	User-defined size in pixels.	width=640	Can be specified as the second positional (unnamed) attribute.
height	User-defined size in pixels.	height=480	Can be specified as the third positional (unnamed) attribute.
options	autoplay, loop, modest, nocontrols, nfullscreen	opts="autoplay,loop"	The controls are enabled by default. The modest option enables modest branding for a YouTube video.
start	User-defined playback start time in seconds.	start=30	

Attribute	Value(s)	Example Syntax	Comments
<code>end</code>	User-defined playback end time in seconds.	<code>end=90</code>	
<code>theme</code>	The YouTube theme to use for the frame.	<code>theme=light</code>	Valid values are dark (the default) and light.
<code>lang</code>	The language used in the YouTube frame.	<code>lang=fr</code>	A two-letter language code or fully specified locale.

# Chapter 31. Audio

*Audio attributes and values*

Attribute	Value(s)	Example Syntax	Comments
options	autoplay, loop, controls, nocontrols	audio::ocean_waves.mp3[options ="autoplay,loop"]	The controls value is enabled by default

# Chapter 32. Admonition

There are certain statements you may want to draw attention to by taking them out of the content's flow and labeling them with a priority. These are called admonitions. It's rendered style is determined by the assigned label (i.e., value). Asciidoctor provides five admonition style labels:

- **NOTE**
- **TIP**
- **IMPORTANT**
- **CAUTION**
- **WARNING**

## Caution vs. Warning

When choosing the admonition type, you may find yourself getting confused between "caution" and "warning" as these words are often used interchangeably. Here's a simple rule to help you differentiate the two:

- Use **CAUTION** to advise the reader to *act* carefully (i.e., exercise care).
- Use **WARNING** to inform the reader of danger, harm, or consequences that exist.

To find a deeper analysis, see <https://www.differencebetween.com/difference-between-caution-and-vs-warning/>.

When you want to call attention to a single paragraph, start the first line of the paragraph with the label you want to use. The label must be uppercase and followed by a colon (:).

### Admonition paragraph syntax

WARNING: Wolpertingers are known to nest in server racks. ① ②  
Enter at your own risk.

① The label must be uppercase and immediately followed by a colon (:).

② Separate the first line of the paragraph from the label by a single space.

### Result: Admonition paragraph

**WARNING** Wolpertingers are known to nest in server racks. Enter at your own risk.

When you want to apply an admonition to complex content, set the label as a style attribute on a block. As seen in the next example, admonition labels are commonly set on example blocks. This behavior is referred to as **masquerading**. The label must be uppercase when set as an attribute on a block.

## Admonition block syntax

[IMPORTANT] ①

.Feeding the Werewolves

==== ②

While werewolves are hardy community members, keep in mind the following dietary concerns:

- . They are allergic to cinnamon.
- . More than two glasses of orange juice in 24 hours makes them howl in harmony with alarms and sirens.
- . Celery makes them sad.

====

① Set the label in an attribute list on a delimited block. The label must be uppercase.

② Admonition styles are commonly set on example blocks. Example blocks are delimited by four equal signs (====).

## Result: Admonition block

### *Feeding the Werewolves*

While werewolves are hardy community members, keep in mind the following dietary concerns:

#### IMPORTANT

1. They are allergic to cinnamon.
2. More than two glasses of orange juice in 24 hours makes them howl in harmony with alarms and sirens.
3. Celery makes them sad.

In the examples above, the admonition is rendered in a callout box with the style label in the gutter. You can replace the textual labels with icons by setting the `icons` attribute on the document. This is how the WARNING admonition paragraph renders when icons is set and assigned the `font` value.

## Admonition paragraph with icons set

WARNING: Wolpertingers are known to nest in server racks.

Enter at your own risk.

## Result: Admonition paragraph with icons set



Wolpertingers are known to nest in server racks. Enter at your own risk.

Learn more about using Font Awesome or custom icons with admonitions in the [icons](#) section.

# Chapter 33. Sidebar

Use a sidebar for ancillary content that doesn't fit into the flow of the document's narrative. A sidebar can be titled and contain any type of content such as source code and images.

*Sidebar syntax*

```
.AsciiDoc history ①  
**** ②  
AsciiDoc was first released in Nov 2002 by Stuart Rackham.  
It was designed from the start to be a shorthand syntax  
for producing professional documents like DocBook and LaTeX.  
****
```

① A title is optional.

② A sidebar is delimited by four asterisks (\*\*\*\*).

*Result: Sidebar with title*

## AsciiDoc history

AsciiDoc was first released in Nov 2002 by Stuart Rackham. It was designed from the start to be a shorthand syntax for producing professional documents like DocBook and LaTeX.

# Chapter 34. Example

*Example syntax*

```
.Sample document
=====
Here's a sample AsciiDoc document:

[listing]
....
= Title of Document
Doc Writer
:toc:

This guide provides...
....

The document header is useful, but not required.
=====
```

*Result: Example block*

*Sample document*

```
Here's a sample AsciiDoc document:

= Title of Document
Doc Writer
:toc:

This guide provides...
```

The document header is useful, but not required.

# Chapter 35. Prose Excerpts, Quotes and Verses

Prose excerpts, quotes and verses share the same syntax structure, including:

- block name, either `quote` or `verse`
- name of who the content is attributed to
- bibliographical information of the book, speech, play, poem, etc., where the content was drawn from
- excerpt text

## 35.1. Quote

For content that doesn't require the preservation of line breaks, set the `quote` attribute in the first position of the attribute list. Next, set the attribution and relevant citation information. However, these positional attributes are optional.

*Anatomy of a basic quote*

```
[quote, attribution, citation title and information]  
Quote or excerpt text
```

If the quote is a single line or paragraph, you can place the attribute list directly on top of the text.

*Quote paragraph syntax*

```
.After landing the cloaked Klingon bird of prey in Golden Gate park: ①  
[quote, Captain James T. Kirk, Star Trek IV: The Voyage Home] // <2> ③ ④  
Everybody remember where we parked. ⑤
```

- ① Mark lead-in text explaining the context or setting of the quote using a period (.). (optional)
- ② For content that doesn't require the preservation of line breaks, set `quote` in the first position of the attribute list.
- ③ The second position contains who the excerpt is attributed to. (optional)
- ④ Enter additional citation information in the third position. (optional)
- ⑤ Enter the excerpt or quote text on the line immediately following the attribute list.

*Result: Quote paragraph*

```
After landing the cloaked Klingon bird of prey in Golden Gate park:
```

```
Everybody remember where we parked.
```

```
— Captain James T. Kirk, Star Trek IV: The Voyage Home
```

If the quote or excerpt is more than one paragraph, place the text between delimiter lines consisting of four underscores ().

#### *Quote block syntax*

```
[quote, Monty Python and the Holy Grail]
```

```
----  
Dennis: Come and see the violence inherent in the system. Help! Help! I'm being  
repressed!
```

```
King Arthur: Bloody peasant!
```

```
Dennis: Oh, what a giveaway! Did you hear that? Did you hear that, eh? That's what I'm  
on about! Did you see him repressing me? You saw him, Didn't you?
```

```
----
```

#### *Result: Quote block*

```
Dennis: Come and see the violence inherent in the system. Help! Help!  
I'm being repressed!
```

```
King Arthur: Bloody peasant!
```

```
Dennis: Oh, what a giveaway! Did you hear that? Did you hear that, eh?  
That's what I'm on about! Did you see him repressing me? You saw him,  
Didn't you?
```

```
— Monty Python and the Holy Grail
```

Asciidoctor also provides three alternative ways to markup quotes and prose excerpts.

### **35.1.1. Quoted paragraph**

You can turn a single paragraph into a blockquote by:

1. surrounding it with double quotes
2. adding an optional attribution (prefixed with two dashes) below the quoted text

#### *Quoted paragraph syntax*

```
"I hold it that a little rebellion now and then is a good thing,  
and as necessary in the political world as storms in the physical."  
-- Thomas Jefferson, Papers of Thomas Jefferson: Volume 11
```

*Result: Quoted paragraph*

I hold it that a little rebellion now and then is a good thing, and as necessary in the political world as storms in the physical.

— Thomas Jefferson, Papers of Thomas Jefferson: Volume 11

### 35.1.2. Air quotes

[Air quotes](#) are two double quotes on each line, emulating the gesture of making quote marks with two fingers on each hand.

[, James Baldwin]

""

Not everything that is faced can be changed.  
But nothing can be changed until it is faced.

""

*Result: Air quotes*

"" Not everything that is faced can be changed. But nothing can be changed until it is faced. """

### 35.1.3. Markdown-style blockquotes

Asciidoctor supports Markdown-style blockquotes.

*Markdown-style blockquote syntax*

```
> I hold it that a little rebellion now and then is a good thing,  
> and as necessary in the political world as storms in the physical.  
> -- Thomas Jefferson, Papers of Thomas Jefferson: Volume 11
```

*Result: Markdown-style blockquote*

I hold it that a little rebellion now and then is a good thing, and as necessary in the political world as storms in the physical.

— Thomas Jefferson, Papers of Thomas Jefferson: Volume 11

Like Markdown, Asciidoctor supports block content inside the blockquote, including nested blockquotes.

### *Markdown-style blockquote containing block content*

```
> > What's new?  
>  
> I've got Markdown in my AsciiDoc!  
>  
> > Like what?  
>  
> * Blockquotes  
> * Headings  
> * Fenced code blocks  
>  
> > Is there more?  
>  
> Yep. AsciiDoc and Markdown share a lot of common syntax already.
```

Here's how this conversation renders.

### *Result: Markdown-style blockquote with block content*

```
| What's new?  
| I've got Markdown in my AsciiDoc!  
| | Like what?  
| | • Blockquotes  
| | • Headings  
| | • Fenced code blocks  
| Is there more?  
| Yep. AsciiDoc and Markdown share a lot of common syntax already.
```

## 35.2. Verse

When you need to preserve indents and line breaks, use the `verse` block name. Verses are defined by setting `verse` on a paragraph or an excerpt block delimited by four underscores (`____`).

### *Verse paragraph syntax*

```
[verse, Carl Sandburg, two lines from the poem Fog]  
The fog comes  
on little cat feet.
```

*Result: Verse paragraph*

The fog comes  
on little cat feet.

— Carl Sandburg, two lines from the poem Fog

When the verse content includes blank or indented lines, enclose it in an excerpt block.

*Verse delimited block syntax*

[verse, Carl Sandburg, Fog]

----  
The fog comes  
on little cat feet.

It sits looking  
over harbor and city  
on silent haunches  
and then moves on.

----

*Result: verse block*

The fog comes  
on little cat feet.

It sits looking  
over harbor and city  
on silent haunches  
and then moves on.

— Carl Sandburg, Fog

# Chapter 36. Comments

## *Line*

```
// A single-line comment.
```



Single-line comments can be used to divide elements, such as two adjacent lists.

## *Block*

```
////
```

```
A multi-line comment.
```

```
Notice it's a delimited block.
```

```
////
```

# Controlling Your Content

# Chapter 37. Text Substitutions

Text substitution elements replace characters, markup, attribute references, and macros with converter specific styles and values. When Asciidoctor processes a document it uses a set of six text substitution elements. The processor runs the text substitution elements in the following order.

1. Special characters
2. Quotes
3. Attribute references
4. Replacements
5. Inline macros
6. Post replacements

In turn, these substitutions are organized into composite value groups. The table below shows which substitution elements are included in each group.

*Substitution groups*

Group	Special characters	Quotes	Attributes	Replacements	Macros	Post replacements
Header	✓	✗	✓	✗	✗	✗
None	✗	✗	✗	✗	✗	✗
Normal	✓	✓	✓	✓	✓	✓
Pass	✗	✗	✗	✗	✗	✗
Verbatim	✓	✗	✗	✗	✗	✗

By default, the `normal` substitution group is applied to most block and inline elements. However, there are a few exceptions.

The `header` substitution group is applied to metadata lines (author and revision information) in the document header. This group is also applied to the values of attribute entries, regardless of whether those entries are defined in the header or elsewhere in the document. In the header, only special characters and attribute references are replaced. In attribute entries, you can also use the [inline pass macro](#).

Fenced, literal, listing, and source blocks are processed with the `verbatim` substitution group. Only special characters are replaced in these blocks.

The `pass` substitution group can only be applied to passthrough elements. Attribute references and macros are replaced in passthroughs.

The `none` substitution group is applied to comment blocks.

## The title substitution group

The `title` substitution group includes the same text substitutions as the normal group. However, the order that the substitutions are executed is slightly different. Text substitutions are applied to titles in the following sequence:

1. Special characters
2. Quotes
3. Replacements
4. Inline macros
5. Attribute references
6. Post replacements

## 37.1. Special Characters

When applicable, the first text substitution to occur is the replacement of any special characters. This process is handled by the `specialchars` element. The `specialchars` element searches for three characters (`<`, `>`, `&`) and replaces them with their [named character references](#).

- The less than symbol, `<`, is replaced with the named character reference `&lt;`.
- The greater than symbol, `>`, is replaced with the named character reference `&gt;`.
- An ampersand, `&`, is replaced with the named character reference `&amp;`.

By default, the special characters substitution occurs on all inline and block elements except for comments and certain passthroughs. The substitution of special characters can be controlled on blocks using the [subs attribute](#) and on inline elements using the [passthrough macro](#).



Special character substitution precedes attribute substitution, so you will need to manually escape any attributes containing special characters that you set in the CLI or API. For example, on the command line, type `-a toc-title="Sections, Tables & Figures"` instead of `-a toc-title="Sections, Tables & Figures"`.

## 37.2. Quotes

The `quotes` substitution replaces the formatting markup on inline elements.

For example, when a document is converted to HTML, any asterisks enclosing text are replaced with `<strong>` HTML tags.

*Syntax input*

```
Happy werewolves are *really* slobbery.
```

## *HTML output*

Happy werewolves are <**strongstrong**> slobbery.

The following table shows the HTML markup that is generated by the quotes substitution process.

### *HTML markup generated from AsciiDoc formatting syntax*

Name	AsciiDoc	HTML
emphasis	_word_	&lt;em&gt;word&lt;/em&gt;
strong	*word*	&lt;strong&gt;word&lt;/strong&gt;
monospace	`word`	&lt;code&gt;word&lt;/code&gt;
superscript	^word^	&lt;sup&gt;word&lt;/sup&gt;
subscript	~word~	&lt;sub&gt;word&lt;/sub&gt;
double curved quotes	"`word`"	&#8220;word&#8221;
single curved quotes	'`word`'	&#8216;word&#8217;

The quotes substitution occurs on formatted text within title, paragraph, example, quote, sidebar, and verse blocks.

### *Elements subject to quotes text substitution*

Element	quotes substitution
Attribute Entry Value	✗
Comment	✗
Example	✓
Fenced	✗
Header	✗
Literal	✗
Listing	✗
Macro	✓
Open	Varies
Paragraph	✓
Passthrough	✗
Quote	✓
Sidebar	✓
Source	✗
Special sections	✓

Element	quotes substitution
Table	Varies
Title	✓
Verse	✓

### 37.3. Attributes

Attribute references are replaced with their values when they're processed by the [attributes](#) substitution.

*Elements subject to attributes text substitution*

Element	attributes substitution
Attribute Entry Value	✓
Comment	✗
Example	✓
Fenced	✗
Header	✓
Literal	✗
Listing	✗
Macro	✓
Open	Varies
Paragraph	✓
Passthrough	✗
Quote	✓
Sidebar	✓
Source	✗
Special sections	✓
Table	Varies
Title	✓
Verse	✓

### 37.4. Replacements

The replacements substitution processes textual characters such as marks, arrows and dashes and replaces them with the decimal format of their Unicode code point, i.e. their [numeric character reference](#).

## Textual symbol replacements

Name	Syntax	Unicode Replacer	Rendered	Notes
Copyright	(C)	&#169;	©	
Registered	(R)	&#174;	®	
Trademark	(TM)	&#8482;	™	
Em dash	--	&#8212;	—	Only replaced if between two word characters, between a word character and a line boundary, or flanked by spaces.
				When flanked by space characters (e.g., a -- b), the normal spaces are replaced by thin spaces (&#8201;).
Ellipsis	...	&#8230;	...	
Single right arrow	->	&#8594;	→	
Double right arrow	=>	&#8658;	⇒	
Single left arrow	&lt;-	&#8592;	←	
Double left arrow	&lt;=	&#8656;	⇐	
Typographic apostrophe	Sam's	Sam&#8217;s	Sam’s	The typewriter apostrophe is replaced with the typographic (aka curly) apostrophe.



The `replacements` element depends on the substitutions completed by the `specialcharacters` element. This is important to keep in mind when applying custom substitutions to a block. See the section about [applying custom substitutions](#) for more information.

The replacements substitution also recognizes [HTML and XML character references](#) as well as [decimal and hexadecimal Unicode code points](#) and substitutes them for their corresponding decimal form Unicode code point.

For example, to produce the `&sect;` symbol you could write `&#sect;`, `&#x00A7;`, or `&#167;`. When the document is processed, `replacements` will replace the section symbol reference, regardless of whether it is a named character reference or a numeric character reference, with `&#167;`. In turn, `&#167;` will display as `&sect;`.

## Anatomy of a character reference

A character reference is a standard sequence of characters that is substituted for a single character when Asciidoctor processes a document. There are two types of character references: named character references and numeric character references.

A named character reference (often called a *character entity reference*) is a short name that refers to a character (i.e., glyph). To make the reference, the name must be prefixed with an ampersand (`&`) and end with a semicolon (`;`).

For example:

- `&dagger;` displays as &dagger;;
- `&euro;` displays as &euro;;
- `&loz;` displays as &loz;;

Numeric character references are the decimal or hexadecimal Universal Character Set/Unicode code points which refer to a character.

- The decimal code point references are prefixed with an ampersand (`&`), followed by a hash (#), and end with a semicolon (`;`).
- Hexadecimal code point references are prefixed with an ampersand (`&`), followed by a hash (#), followed by a lowercase `x`, and end with a semicolon (`;`).

For example:

- `&x2020;` or `&#8224;` displays as †
- `&x20AC;` or `&#8364;` displays as €
- `&#x25CA;` or `&#9674;` displays as &#x25CA;;

Developers may be more familiar with using **Unicode escape sequences** to perform text substitutions. For example, to produce an @ sign using a Unicode escape sequence, you would prefix the hexadecimal Unicode code point with a backslash (\) and an uppercase or lowercase `u`, i.e. `u0040`. However, Asciidoctor doesn't process and replace Unicode escape sequences at this time.



Asciidoctor also provides numerous built-in attributes for representing characters and symbols. These attributes and their corresponding output are listed in [Predefined attributes for character replacements](#) <sup>[1][2][3]</sup>.

The replacements substitution occurs within title, paragraph, example, quote, sidebar, and verse blocks.

*Elements subject to replacements text substitution*

Element	replacements substitution
Attribute Entry Value	✗
Comment	✗
Example	✓
Fenced	✗
Header	✗
Literal	✗
Listing	✗
Macro	✓
Open	Varies
Paragraph	✓
Passthrough	✗
Quote	✓
Sidebar	✓
Source	✗
Special sections	✓
Table	Varies
Title	✓
Verse	✓

## 37.5. Macros

Macros are processed by the `macros` element. The macros substitution replaces a macro's content with the appropriate built-in and user-defined configuration.

*Elements subject to macros substitution*

Element	macros substitution
Attribute Entry Value	inline pass macro only
Comment	✗
Example	✓
Fenced	✗
Header	✗
Literal	✗
Listing	✗

Element	macros substitution
Macro	✓
Open	Varies
Paragraph	✓
Passthrough	✓
Quote	✓
Sidebar	✓
Source	✗
Special sections	✓
Table	Varies
Title	✓
Verse	✓

## 37.6. Post Replacements

The line break character, `+`, is replaced when the `post_replacements` process runs.

*Elements subject to post replacements text substitution*

Element	post_replacements substitution
Attribute Entry Value	✗
Comment	✗
Example	✓
Fenced	✗
Header	✗
Literal	✗
Listing	✗
Macro	✓
Open	Varies
Paragraph	✓
Passthrough	✗
Quote	✓
Sidebar	✓
Source	✗

Element	post_replacements substitution
Special sections	✓
Table	Varies
Title	✓
Verse	✓

## 37.7. Applying Substitutions

Specific substitution elements can be applied to any block or paragraph by setting the `subs` attribute. The `subs` attribute can be assigned a comma separated list of the following substitution elements and groups.

### none

Disables substitutions

### normal

Performs all substitutions except for callouts

### verbatim

Replaces special characters and processes callouts

### specialchars, specialcharacters

Replaces `<`, `>`, and `&` with their corresponding entities

### quotes

Applies text formatting

### attributes

Replaces attribute references

### replacements

Substitutes textual and character reference replacements

### macros

Processes macros

### post\_replacements

Replaces the line break character (`+`)

Let's look at an example where you only want to process special characters, formatting markup, and callouts in a literal block. By default, literal blocks are only subject to special characters substitution. But you can change this behavior by setting the `subs` attribute in the block's attribute list.

```
[source,java,subs="verbatim,quotes"] ①  
----  
System.out.println("Hello *bold* text"). ②  
----
```

- ① The subs attribute is set in the attribute list and assigned the `verbatim` and `quotes` values.  
② The formatting markup in this line will be replaced when the quotes substitution runs.

```
System.out.println("Hello <strong>bold</strong> text"). ① ②
```

- ① The `verbatim` value enables the callouts to be processed.  
② The `quotes` value enables the text formatting to be processed.

If you are applying the same set of substitutions to numerous blocks, you should consider making them an attribute entry to ensure consistency.

 :markup-in-source: verbatim,quotes  
[source,java,subs="{markup-in-source}"]  
----  
System.out.println("Hello \*bold\* text").  
----

Another way to ensure consistency and keep your documents clean and simple is to use the [TreeProcessor extension](#).

## 37.8. Incremental Substitutions

When you set the `subs` attribute on a block, you automatically remove all of its default substitutions. For example, if you set `subs` on a literal block, and assign it a value of `attributes`, only attributes are substituted. The `verbatim` substitution will not be applied. To remedy this situation, Asciidoctor provides a syntax to append or remove substitutions instead of replacing them outright.

You can add or remove a substitution from the default substitution list using the plus (+) and minus (-) modifiers. These are known as *incremental substitutions*.

`<substitution>+`

Prepends the substitution to the default list.

`+<substitution>`

Appends the substitution to the default list.

## -<substitution>

Removes the substitution from the default list.

For example, you can add the **attributes** substitution to a listing block's default substitution list.

*Add attributes substitution to a default substitution list*

```
[source,xml,subs="attributes+"]
-----
<version>{version}</version>
-----
```

Similarly, you can remove the **callouts** substitution.

*Remove callouts substitution from a default substitution list*

```
[source,xml,subs="-callouts"]
.An illegal XML tag
-----
①
  content inside "1" tag
</1>
-----
```

You can also specify whether the substitution is placed at the beginning or end of the substitution list. If a **+** comes before the name of the substitution, then it's added to the end of the existing list, and if a **+** comes after the name, it's added to the beginning of the list.

```
[source,xml,subs="attributes+,+replacements,-callouts"]
-----
<version>{version}</version>
<copyright>(C) ACME</copyright>
①
  content inside "1" tag
</1>
-----
```

In the above example, the quotes, then the special characters, and then the attributes substitutions will be applied to the listing block.

### 37.8.1. Applying Substitutions to Inline Elements

Custom substitutions can also be applied to some inline elements, such as the **pass macro**.

For example, the quotes text substitution value is assigned in the inline pass macro below.

```
The text pass:q[<u>underline *me*</u>] is underlined and the word "'me'" is bold.
```

The text <u>underline <strong>me</strong></u> is underlined and the word “me” is bold.

## 37.9. Preventing Substitutions

Asciidoctor provides several approaches for preventing substitutions.

### *Backslash escaping*

To prevent punctuation from being interpreted as formatting markup, precede it with a backslash (\). If the formatting punctuation begins with two characters (e.g., \_\_), you need to precede it with two backslashes (\\\). This is also how you can prevent character and attribute references from substitution. When your document is processed, the backslash is removed so it doesn’t display in your output.

\\*Stars\* will appear as \*Stars\*, not as bold text.

\&sect; will appear as an entity, not the &sect; symbol.

\\\\_func\\_ will appear as \_\_func\_\_, not as emphasized text.

\{two-semicolons} will appear {two-semicolons}, not resolved as ;;.

You can also prevent substitutions with [macro](#) and [block passthroughs](#).

# Chapter 38. Literal Text and Blocks

Literal paragraphs and blocks display the text you write exactly as you enter it. Literal text is treated as preformatted text. The text is shown in a fixed-width font and endlines are preserved. Only [special characters](#) and callouts are replaced when the document is converted.

Literal blocks are defined three ways:

1. Indenting the first line of a paragraph by one or more spaces
2. Applying the `literal` attribute to a paragraph or block
3. Using the literal block delimiter (`....`)

When a line begins with one or more spaces it is displayed as a literal paragraph. This method is a quick and easy way to insert code snippets.

*Implicit literal text*

```
~/secure/vault/defops
```

*Result: Implicit literal text*

```
~/secure/vault/defops
```

When you want an entire block of text to be literal and would prefer not to indent it, set the `literal` attribute on top of the element.

*Literal style paragraph syntax*

```
[literal]
error: The requested operation returned error: 1954 Forbidden search for defensive
operations manual
absolutely fatal: operation initiation lost in the dodecahedron of doom
would you like to die again? y/n
```

*Result: Literal style paragraph*

```
error: The requested operation returned error: 1954 Forbidden search for defensive
operations manual
absolutely fatal: operation initiation lost in the dodecahedron of doom
would you like to die again? y/n
```

Finally, you can surround the content you want rendered as literal by enclosing it in a set of literal block delimiters (`....`). This method is useful when the content consists of several elements that are separated by blank lines.

*Literal delimited block syntax*

....  
Lazarus: Where is the \*defensive operations manual\*?

Computer: Calculating ...  
Can not locate object that you are not authorized to know exists.  
Would you like to ask another question?

Lazarus: Did the werewolves tell you to say that?

Computer: Calculating ...

....

Notice in the output that the bold text formatting is not applied to the text nor are the three consecutive periods replaced by the ellipsis Unicode character.

*Result: Literal delimited block*

Lazarus: Where is the \*defensive operations manual\*?

Computer: Calculating ...  
Can not locate object that you are not authorized to know exists.  
Would you like to ask another question?

Lazarus: Did the werewolves tell you to say that?

Computer: Calculating ...

# Chapter 39. Listing Blocks

Like literal blocks, the content in listing blocks is displayed exactly as you entered it. Listing block content is converted to `<pre>` text. The content in listing blocks is only subject to [special character](#) and callout substitutions.

The `listing` block name can be applied to content two ways.

1. Set the `listing` attribute on the element.
2. Contain the content within a delimited listing block.

The listing block name is applied to an element, such as a paragraph, by setting the `listing` attribute on that element.

*Listing paragraph syntax*

```
[listing]
```

This is an example of a paragraph styled with 'listing'.

Notice that the monospace markup is preserved in the output.

*Result: Listing paragraph*

```
This is an example of a paragraph styled with 'listing'.
```

Notice that the monospace markup is preserved in the output.

A delimited listing block is surrounded by lines composed of four hyphens (----).

*Delimited listing block syntax*

```
----
```

This is an example of a \_listing block\_.

The content inside is displayed as `<pre>` text.

```
----
```

Here's how the block above appears when rendered as HTML.

*Result: Listing block*

```
This is an example of a _listing block_.
```

The content inside is displayed as `<pre>` text.

You should notice a few things about how the content is processed.

- the HTML tag `<pre>` is escaped

- then endlines are preserved
- the phrase *listing block* is not italicized, despite having underscores around it.

Listing blocks are good for displaying raw source code, especially when used in tandem with the `source` and `source-highlighter` attributes. The example below shows a listing block with `source` and the language `ruby` applied to its content.

#### *Source block syntax*

```
.app.rb
[source,ruby]
-----
require 'sinatra'

get '/hi' do
  "Hello World!"
end
-----
```

#### *Result: Listing block with the source attribute set*

```
app.rb

require 'sinatra'

get '/hi' do
  "Hello World!"
end
```

Detailed instructions for using the `source` and `source-highlighter` attributes are provided in the [source code blocks](#) section.

#### *Listing block with custom substitutions syntax*

```
:version: 1.5.6.1

[source,xml,subs="verbatim,attributes"]
-----
<dependency>
  <groupId>org.asciidoctor</groupId>
  <artifactId>asciidoctor-java-integration</artifactId>
  <version>{version}</version>
</dependency>
-----
```

*Result: Listing block with custom substitutions applied*

```
<dependency>
  <groupId>org.asciidoctor</groupId>
  <artifactId>asciidoctor-java-integration</artifactId>
  <version>1.5.6.1</version>
</dependency>
```

## 39.1. To Wrap or to Scroll

The default Asciidoctor stylesheet wraps long lines in listing and literal blocks by applying the CSS `white-space: pre-wrap` and `word-wrap: break-word`. The lines are wrapped at word boundaries, similar to how most text editors wrap lines. This prevents horizontal scrolling which some users considered a greater readability problem than line wrapping.

However, this behavior is configurable because there are times when you don't want the lines in listing and literal blocks to wrap.

There are two ways to prevent lines from wrapping so that horizontal scrolling is used instead:

- `nowrap` block option
- unset the `prewrap` document attribute (on by default)

You can use the `nowrap` option on literal or listing blocks to prevent lines from being wrapped in the HTML.

*Listing block with nowrap option syntax*

```
[source%nowrap,java]
-----
public class ApplicationConfigurationProvider extends HttpConfigurationProvider
{
    @Override
    public Configuration getConfiguration(ServletContext context)
    {
        return ConfigurationBuilder.begin()
            .addRule()
            .when(Direction.isInbound().and(Path.matches("/{path}")))
            .perform(Log.message(Level.INFO, "Client requested path: {path}"))
            .where("path").matches(".");
    }
}
-----
```

When the nowrap option is used, the `nowrap` class is added to the `<pre>` element. This class changes the CSS to `white-space: pre` and `word-wrap: normal`.

*Result: Listing block with nowrap option applied*

```
public class ApplicationConfigurationProvider extends HttpConfigurationProvider
{
    @Override
    public Configuration getConfiguration(ServletContext context)
    {
        return ConfigurationBuilder.begin()
            .addRule()
            .when(Direction.isInbound().and(Path.matches("/{path}")))
            .perform(Log.message(Level.INFO, "Client requested path: {path}"))
            .where("path").matches(".");
    }
}
```

To prevent lines from wrapping globally, unset the `prewrap` attribute on the document.

*Disable prewrap globally (thus, enabling nowrap)*

```
:prewrap!:
```

When the `prewrap` attribute is unset, the `nowrap` class is added to any `<pre>` elements.

Now, you can use the line wrapping strategy that works best for you and your readers.

# Chapter 40. Passthroughs

The passthrough is the primary mechanism by which to escape content in AsciiDoc (far more comprehensive and consistent than using a backslash). As the name implies, a passthrough passes content directly through to the output document without applying any [substitutions](#). When using the passthrough block or macro, it's possible to specify a custom set of substitutions to be applied before the content is reintroduced. Other forms of passthroughs, such as the single and double plus, apply substitutions strictly for compliance with the output format.



Using passthroughs to pass content (without substitutions) can couple your content to a specific output format, such as HTML. In these cases, you should use conditional preprocessor directives to route passthrough content for different output formats based on the current backend.

## 40.1. Passthrough Macros

Asciidoctor supports several forms of the passthrough macro.

### inline pass macro

An inline macro named `pass` that can be used to passthrough content. Supports an optional set of substitutions.

```
pass:[content like #{variable} passed directly to the output] followed by normal content.
```

```
content with only select substitutions applied: pass:c,a[__<{email}>__]
```

### single and double plus

A special syntax for preventing text from being formatted. Only escapes special characters for compliance with the output format and doesn't support explicit substitutions.

### triple plus

A special syntax for designating passthrough content. Does not apply any substitutions (equivalent to the inline pass macro) and doesn't support explicit substitutions.

### double dollar (**deprecated**)

A deprecated special syntax for designating passthrough content. Like the triple plus, does not apply any substitutions and doesn't support explicit substitutions.



Asciidoctor does not implement the block pass macro. Instead, you should use a [pass block](#).

### 40.1.1. Inline pass macro and explicit substitutions

To exclude a phrase from substitutions and disable escaping of special characters, enclose it in the inline pass macro. For example, here's one way to format text as underline when generating HTML

from AsciiDoc:

The text pass:[<u>underline me</u>] is underlined.

The text <u>underline me</u> is underlined.

If you want to enable ad-hoc `quotes` substitution, then assign the `macros` value to `subs` and use the inline pass macro.

```
[subs=+macros] ①
-----
I better not contain *bold* or _italic_ text.
pass:quotes[But I should contain *bold* text.] ②
-----
```

① `macros` is assigned to `subs`, which allows any macros within the block to be processed.

② The pass macro is assigned the `quotes` value. Text within the square brackets will be formatted.

The inline pass macro does introduce additional markup into the source code that could make it invalid in raw form. However, the output it produces will be valid when viewed in a viewer (HTML, PDF, etc.).

I better not contain \*bold\* or \_italic\_ text.  
But I should contain <strong>bold</strong> text.

The inline pass macro also accepts shorthand values for specifying substitutions.

- `c` = special characters
- `q` = quotes
- `a` = attributes
- `r` = replacements
- `m` = macros
- `p` = post replacements

For example, the quotes text substitution value is assigned in the inline passthrough macro below:

The text pass:q[<u>underline \*me\*</u>] is underlined and the word "me" is bold.

The text <u>underline <strong>me</strong></u> is underlined and the word "me" is bold.

### 40.1.2. Triple plus passthrough

The triple-plus passthrough works much the same way as the pass macro. To exclude content from substitutions, enclose it in triple pluses (+++).

```
+++content passed directly to the output+++ followed by normal content.
```

The triple-plus macro is often used to output custom HTML or XML.

```
The text +++<u>underline me</u>+++ is underlined.
```

```
The text <u>underline me</u> is underlined.
```

### 40.1.3. Nesting blocks and passthrough macros

When you're using passthroughs inside literal and listing blocks, it can be easy to forget that the triple-plus, double-dollar, and backtick passthroughs are macros (not quotes). If you want to enable the passthroughs, make sure to assign the `macros` value to the `subs` attribute.

```
[source,java,subs="+quotes,+macros"]
-----
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
        **.antMatchers("/resources/**").permitAll()**
        .anyRequest().authenticated()
        .and()
        .formLogin()
        .loginPage("/login")
        .permitAll();
-----
```

```
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
        <strong>.antMatchers("/resources/**").permitAll()</strong>
        .anyRequest().authenticated()
        .and()
        .formLogin()
        .loginPage("/login")
        .permitAll();
```

## 40.2. Passthrough Blocks

The passthrough block and **pass** block style exclude block content from all substitutions (unless specified otherwise).

A passthrough block is delimited by four plus signs (++++). The **pass** style is implied.

```
++++  
<video poster="images/movie-reel.png">  
  <source src="videos/writing-zen.webm" type="video/webm">  
</video>  
++++
```

(Keep in mind that AsciiDoc has a video macro, so this example is merely for demonstration. However, a passthrough could come in handy if you need to output more sophisticated markup than what the built-in HTML converter produces).

The pass style can also be set on a paragraph or an open block.

```
[pass]  
<u>underline me</u> is underlined.
```

You can use the **subs** attribute to specify a comma-separated list of substitutions. These substitutions will be applied to the content prior to it being reintroduced to the output document.

```
[subs=attributes]  
++++  
{name}  
image:tiger.png[]  
++++
```

The content of the pass block does not get wrapped in a paragraph. Therefore, you can use the **pass** style in combination with the **normal** substitution category to output content without generating a paragraph.

```
[subs=normal]  
++++  
Normal content which is not enclosed in a paragraph.  
++++
```

# Chapter 41. Open Blocks

The most versatile block of all is the open block.

*Open block syntax*

```
--  
An open block can be an anonymous container,  
or it can masquerade as any other block.  
--
```

*Result: Open block*

```
An open block can be an anonymous container, or it can masquerade as any other block.
```

An open block can act as any other block, with the exception of *pass* and *table*. Here's an example of an open block acting as a sidebar.

*Open block masquerading as a sidebar syntax*

```
[sidebar]  
.Related information  
--  
This is aside text.  
  
It is used to present information related to the main content.  
--
```

*Result: Open block masquerading as a sidebar*

## Related information

This is aside text.

It is used to present information related to the main content.

This is an open block acting as a source block.

*Open block masquerading as a source block syntax*

```
[source]  
--  
puts "I'm a source block!"  
--
```

*Result: Open block masquerading as a source block*

```
puts "I'm a source block!"
```

# Enriching Your Content

# Chapter 42. Equations and Formulas

If you need to get technical in your writing, Asciidoctor integrates with [MathJax](#). MathJax is the standard library for displaying Science, Technology, Engineering and Math (STEM) expressions in the browser.

Thanks to [MathJax JavaScript library](#), Asciidoctor supports both [AsciiMathML](#) and [TeX and LaTeX](#) math notation in the same document.

# Chapter 43. Activating stem support

To activate equation and formula support, simply set the `stem` attribute in the document's header (or by passing the attribute to the command line or API).

*Setting the stem attribute*

```
= My Diabolical Mathematical Opus  
Jamie Moriarty  
:stem: ①
```

① The default interpreter value, `asciimath`, is assigned implicitly.

By default, Asciidoctor's stem support assumes all equations are AsciiMath if not specified explicitly. If you want to use the LaTeX interpreter by default, assign `latexmath` to the `stem` attribute.

The html backend will just use mathjax client-side to render math notation, whereas the docbook backend will use it to convert math to mathml.

*Assigning an alternative interpreter to the stem attribute*

```
= My Diabolical Mathematical Opus  
Jamie Moriarty  
:stem: latexmath
```



You can still use both interpreters in the same document. The value of the `stem` attribute merely sets the default interpreter. To set the interpreter explicitly for a given block or inline span, just use `asciimath` or `latexmath` in place of `stem` as explained in [Using Multiple Stem Interpreters](#).

Stem content can be displayed inline with other content or as discrete blocks. No substitutions are applied to the content within a stem macro or block.

## 43.1. Inline Stem Content

The best way to mark up an inline formula is to use the `stem` macro.

*Inline stem macro syntax*

```
stem:[sqrt(4) = 2] ① ②
```

Water (stem:[H\_2O]) is a critical component.

① The inline stem macro contains only one colon (:).

② Place the content you want interpreted within the square brackets ([]) of the macro.

*Rendered inline stem content*

```
sqrt(4) = 2
```

Water ( $H_2O$ ) is a critical component.

If the inline stem equation contains a right square bracket, you must escape this character using a backslash.

*Inline stem macro with a right square bracket*

```
A matrix can be written as stem: [[[a,b\], [c,d\]\]]((n),(k)).
```

*Rendered inline stem macro with a right square bracket*

```
A matrix can be written as [[a,b],[c,d]]((n),(k)).
```

A stem macro is an implicit passthrough macro. That's why, despite the fact that the `x` expression matches the syntax of an attribute reference, you don't have to escape it.

## 43.2. Block Stem Content

Block formulas are marked up by assigning the `stem` style to a delimited passthrough block.

*Delimited stem block syntax*

```
[stem] ①  
++++ ②  
sqrt(4) = 2  
++++
```

① Assign the stem style to the passthrough block.

② A passthrough block is delimited by a line of four consecutive plus signs (`++++`).

The result is rendered beautifully in the browser thanks to MathJax!

*Rendered delimited stem block*



You don't need to add special delimiters around the expression as the [MathJax documentation](#) suggests. Asciidoctor handles that for you automatically!

## 43.3. Using Multiple Stem Interpreters

You can use multiple interpreters for stem content within the same document by using the interpreter's name instead of the default `stem` name.

For example, if you want LaTeXMath to interpret an inline equation, name the macro `latexmath`.

#### *Inline latexmath macro syntax*

```
latexmath:[C = \alpha + \beta Y^{\gamma} + \epsilon]
```

#### *Rendered latexmath content*

```
C = \alpha + \beta Y^{\gamma} + \epsilon
```

The name that maps to the interpreter you want to use can also be applied to block stem content.

#### *Delimited asciimath block syntax*

```
= My Diabolical Mathematical Opus
Jamie Moriarty
:stem: latexmath

[asciimath]
++++
sqrt(4) = 2
++++
```

## 43.4. Enabling STEM expressions in the DocBook Toolchain

When converting to HTML, Asciidoctor relies on MathJax to parse and render the STEM expressions in the browser. We can't rely on MathJax when converting to DocBook, so Asciidoctor must explicitly convert the expressions into something the DocBook toolchain can understand. Enter the asciimath gem.

The asciimath gem converts AsciiMath expressions to MathML. The DocBook converter uses this functionality if the gem is available. Thus, to enable AsciiMath support when converting to DocBook, you need to install the asciimath gem:

```
$ gem install asciimath
```

For full functionality, you'll also need at least Asciidoctor version 1.5.4.

The asciimath gem converts AsciiMath to MathML. If you're generating a PDF from the DocBook, the MathML needs to be interpreted and drawn into the PDF. In the DocBook FOP pipeline, this is handled by JEuclid.

Fortunately, `fopub` is already configured to use the JEuclid integration. When fopub generates the PDF, the JEuclid FOP plugin processes any MathML found in the DocBook file, including the expressions transformed from AsciiMath to MathML by the asciimath gem. As a result, AsciiMath

stem expressions in the AsciiDoc file will render as expected in the generated PDF.

There's no equivalent gem for converting STEM expressions written in LaTeX to MathML. Instead, you can convert the DocBook to PDF using the dblatex pipeline, which obviously supports LaTeX expressions.

# Chapter 44. User Interface Macros

We are looking for feedback on these macros before setting them in stone. If you have suggestions, we want to hear from you!



You **must** set the `experimental` attribute to enable the UI macros.

## 44.1. Keyboard shortcuts

Asciidoctor recognizes a macro for creating keyboard shortcuts using the syntax `kbd:[key(+key)*]`.

*Keyboard macro syntax*

```
|===
|Shortcut |Purpose
|
|kbd:[F11]
|Toggle fullscreen
|
|kbd:[Ctrl+T]
|Open a new tab
|
|kbd:[Ctrl+Shift+N]
|New incognito window
|
|kbd:[\ ]
|Used to escape characters
|
|kbd:[Ctrl+\]]
|Jump to keyword
|
|kbd:[Ctrl + +]
|Increase zoom
|==
```

*Result: Keyboard macros displaying common browser keyboard shortcuts*

Shortcut	Purpose
<code>F11</code>	Toggle fullscreen
<code>Ctrl+T</code>	Open a new tab
<code>Ctrl+Shift+N</code>	New incognito window
<code>\</code>	Used to escape characters
<code>Ctrl+\]</code>	Jump to keyword
<code>Ctrl++</code>	Increase zoom

You no longer have to struggle to explain to users what keys they are supposed to press.

## 44.2. Menu selections

Trying to explain to someone how to select a menu item can be a pain. With the `menu` macro, the symbols do the work.

*Menu macro syntax*

To save the file, select `menu:File[Save]`.

Select `menu:View[Zoom > Reset]` to reset the zoom level to the default setting.

The instructions in the example above appear below.

*Result: Menu macros displaying menu selections*

To save the file, select **File** › **Save**.

Select **View** › **Zoom** › **Reset** to reset the zoom level to the default setting.

## 44.3. UI buttons

It can be equally difficult to communicate to the reader that they need to press a button. They can't tell if you are saying "OK" or they are supposed to look for a button labeled **OK**. It's all about getting the semantics right. The `btn` macro to the rescue!

*Button macro syntax*

Press the `btn:[OK]` button when you are finished.

Select a file in the file navigator and click `btn:[Open]`.

*Result: Button macros displaying UI buttons*

Press the **[ OK ]** button when you are finished.

Select a file in the file navigator and click **[ Open ]**.

# Chapter 45. Icons

Icons are used for admonition labels, callout numbers and inline symbols. Asciidoctor provides three strategies for display icons: as text, as images or as a characters selected from an icon font. The strategy is controlled using the `:icons` attribute. The default behavior is to display the fallback text.

If you want icons to display using images, set the `:icons` attribute to an empty value in the document header. This strategy is recommended if you are converting to DocBook or you want an easy way to make HTML for viewing offline. The DocBook toolchain provides images for the admonition and callout icons, which you can replace with your own custom icons. If you use the inline icon macro, you'll need to provide the images for those icons.

Alternatively, you can have Asciidoctor “draw” icons using the [Font Awesome](#) font-based icon set. To use this feature, set the value of the `:icons` document attribute to `font` in the document header. You can see the available icons on the [icons page](#). Using Font Awesome provides many more images, but requires online access by default.



The default CSS stylesheet (or any stylesheet produced by the [Asciidoctor stylesheet factory](#)) is required when using font-based icons.

## 45.1. Admonition Icons

When font-based icons are enabled, Asciidoctor will draw the icons for the 5 built-in admonition types using Font Awesome.

Here's an example that shows how to enable font-based icons, starting with the AsciiDoc source:

*Set icons attribute to font in the document header*

```
= Document Title  
:icons: font
```

NOTE: Asciidoctor supports font-based admonition icons, powered by Font Awesome!

Asciidoctor will then emit HTML markup that selects an appropriate font character from the Font Awesome font for each admonition block:

*Result: HTML output when the icons attribute is set to font*

```
<div class="admonitionblock note">
<table>
<tr>
<td class="icon">
<i class="fa icon-note" title="Note"></i>
</td>
<td class="content">
Asciidoctor supports font-based admonition icons, powered by Font Awesome!
</td>
</tr>
</table>
</div>
```

This is how the admonition looks rendered.

*Result: Admonition block label is displayed as an icon when the icons attribute is set*



Asciidoctor supports font-based admonition icons, powered by Font Awesome!

Asciidoctor adds a reference to the Font Awesome stylesheet and font files served from a CDN to the document header:

```
<link rel="stylesheet"
      href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-
      awesome.min.css">
```

The icons chosen are selected by the stylesheet. The default stylesheet maps icons to the following 5 CSS classes:

- .admonitionblock td.icon .icon-note
- .admonitionblock td.icon .icon-tip
- .admonitionblock td.icon .icon-warning
- .admonitionblock td.icon .icon-caution
- .admonitionblock td.icon .icon-important

If you want to customize the icon or the color that is used, you'll need to provide a custom stylesheet or override the styles using a docinfo file. Here's an example that shows how to change the icon for the note admonition to sticky note:

```
.admonitionblock td.icon .icon-note::before {  
    content: "\f24a";  
    color: black;  
}
```

### 45.1.1. Unicode Admonition Icons

Instead of image or font-based icons, it's possible to use Unicode glyphs for admonition icons by applying a little trick.

If the `:icons` attribute is not set on the document, Asciidoctor outputs a textual label that identifies the admonition type. The text for this label comes from an AsciiDoc attribute. The name of the attribute is `<type>-caption`, where `<type>` is the admonition type in lowercase. For example, the attribute for a tip admonition is `tip-caption`.

Instead of a word, you can assign a Unicode glyph to this attribute:

```
:tip-caption: ⓘ  
  
[TIP]  
It's possible to use Unicode glyphs as admonition icons.
```

Here's the HTML that Asciidoctor outputs for the icon cell:

```
<td class="icon">  
  <div class="title"> ⓘ</div>  
</td>
```

Instead of entering the glyph directly, you can enter a character reference instead. However, since you're defining the character reference in an attribute entry, you (currently) have to disable substitutions on the value.

```
:tip-caption: pass:[\#128161;]  
  
[TIP]  
It's possible to use Unicode glyphs as admonition icons.
```

On GitHub, the HTML output that Asciidoctor emits is run through a postprocessing filter that substitutes emoji shortcodes with emoji symbols. That means you can use these shortcodes in the value of the attribute:

```
ifdef::env-github[]
:tip-caption: :bulb:
endif::[]
```

[TIP]

It's possible to use emojis as admonition icons on GitHub.

When the document is processed through the GitHub interface, the shortcodes get replaced with real emojis. This is the only known way to get admonition icons to work on GitHub. Give it a try!

## 45.2. Inline Icons

An icon can be inserted at an arbitrary place in paragraph content with an inline macro.

Here's an example that inserts the Font Awesome tags icon in front of a list of tag names.

*Inline icon macro syntax*

```
icon:tags[] ruby, asciidoctor
```

Here's how the HTML converter converts the above syntax when the `icons` attribute is assigned the `font` value.

*Result: HTML output*

```
<div class="paragraph">
<p><span class="icon"><i class="fa fa-tags"></i></span> ruby, asciidoctor</p>
</div>
```

More importantly, here's how it looks!

*Result: Inline icon macro*

» ruby, asciidoctor

You can even give the icon color by assigning it a role.

*Inline icon macro and role syntax*

```
icon:tags[role="blue"] ruby, asciidoctor
```

*Result: Inline icon macro and role*

» ruby, asciidoctor

If you aren't using font-based icons, Asciidoctor looks for icon images on disk, in the `iconsdir`,

naturally. Here's how the HTML converter converts an icon macro when the `icons` attribute is not set or empty.

*Result: HTML output*

```
<div class="paragraph">
<p><span class="image"></span> ruby,
ascidoctor</p>
</div>
```

Here's how the DocBook converter converts to icon macro. DocBook does not support font-based icons, so the DocBook output is not affected by the value of the `icons` attribute.

*Result: DocBook output*

```
<inlinemediaobject>
  <imageobject>
    <imagedata fileref=".//images/icons/tags.png"/>
  </imageobject>
  <textobject><phrase>tags</phrase></textobject>
</inlinemediaobject> ruby, asciidoctor
```

*Relationship to the inline image macro*

The inline icon macro is similar to the inline image macro with a few exceptions:

- If the `icons` attribute has the value `font`, the macro will translate to a font-based icon in the HTML converter (e.g., `<i class="icon-tags"></i>`)
- If the `icons` attribute does not have the value `font`, or the converter is DocBook, the macro will insert an image into the document that resolves to a file in the `iconsdir` directory (e.g., ``)

The file resolution strategy when using image-based icons is the same used to locate images for the admonition icons. The file extension is set using the `icontype` attribute, which defaults to `PNG` (`png`).

### 45.2.1. Size, Rotate, and Flip

The icon macro has a few attributes that can be used to modify the size and orientation of the icon. At the moment, these are specific to Font Awesome and therefore only apply to HTML output when icon fonts are enabled.

#### `size`

First positional attribute; scales the icon; values: `1x` (default), `2x`, `3x`, `4x`, `5x`, `lg`, `fw`

#### `rotate`

Rotates the icon; values: `90`, `180`, `270`

#### `flip`

Flips the icon; values: `horizontal`, `vertical`

The first unnamed attribute is assumed to be the size. For instance, to make the icon twice the size as the default, simply add `2x` inside the square brackets.

```
icon:heart[2x]
```

This is equivalent to:

```
icon:heart[size=2x]
```

And this is how the  displays.

The previous example emits the following HTML:

```
<span class="icon"><i class="fa fa-heart fa-2x"></i></span>
```

If you want to line up icons so that you can use them as bullets in a list, use the `fw` size as follows:



```
[%hardbreaks]
icon:bolt[fw] bolt
icon:heart[fw] heart
```

To rotate and flip the icon, specify these options using attributes:

```
icon:shield[rotate=90, flip=vertical]
```

The  looks like this.

The previous example emits the following HTML:

```
<span class="icon"><i class="fa-shield fa-rotate-90 fa-flip-vertical"></i></span>
```

## 45.2.2. Link and Window

Like an inline image, it's possible to add additional metadata to an inline icon.

Below are the possible attributes that apply to both font-based and image-based icons:

### link

The URI target used for the icon, which will wrap the converted icon in a link

## window

The target window of the link (when the `link` attribute is specified) (HTML converter)

Here's an example of an icon with a link:

```
icon:download[link="https://rubygems.org/downloads/asciidoc-1.5.2.gem"]
```

The previous example emits the following HTML:

```
<span class="icon"><a class="image" href="https://rubygems.org/downloads/asciidoc-1.5.2.gem"><i class="fa-download"></i></a></span>
```

### 45.2.3. Image Icon Attributes

Below are the possible attributes that apply in the case that font-based icons are **not** in use:

#### alt

The alternative text on the `<img>` tag (HTML backend) or text for `<inlinemediaobject>` (DocBook converter)

#### width

The width applied to the image

#### height

The height applied to the image

#### title

The title of the image displayed when the mouse hovers over it (HTML converter)

#### role

The role applied to the element that surrounds the icon

Currently, the inline icon macro doesn't support any options to change its physical position (such as alignment left or right).

## 45.3. Favicon

When using Asciidoc to generate a standalone HTML document (i.e., the `header_footer` option is `true`), you can instruct the processor to include a link to a favicon by setting the `favicon` attribute in the document header.

```
= Document Title  
:favicon:
```

By default, the processor will add a link reference of type "icon" that refers to a file named

`favicon.ico` (relative to the HTML document):

```
<link rel="icon" type="image/x-icon" href="favicon.ico">
```

This reference gets added inside the HTML `<head>` tag (which explains why this feature is not available when generating an embeddable HTML document).

To modify the name or location of the icon file, simply assign a value to the `favicon` attribute:

```
= Document Title  
:favicon: ./images/favicon/favicon.png
```

This will now produce the following HTML tag:

```
<link rel="icon" type="image/png" href="../images/favicon/favicon.png">
```

Notice the mimetype is automatically set based on the file extension of the image.

The value of the `iconsdir` attribute is not prepend to the favicon path as it is with icons in the content. If you want this directory to be included in the favicon path, you must reference it explicitly:

```
:favicon: {iconsdir}/favicon.png
```



If you're converting a set of AsciiDoc files in multiple directories for the purpose of making a website, and the favicon is located in a shared location, you'll likely want to use a forward slash (/) at the beginning of the favicon path.

If you're looking for more control over how the favicon is declared, you should use a [head docinfo file](#). Keep in mind that if you add an icon link in a head docinfo file and also set the `favicon` attribute, you'll end up with two icon links in the generated HTML document.

# Chapter 46. Syntax Highlighting Source Code

Developers are accustomed to seeing source code colorized to emphasize the code's structure (i.e., keywords, types, delimiters, etc.). This technique is known as *syntax highlighting*. Since this technique is so prevalent—one could say expected—Asciidoctor integrates a wealth of libraries to syntax highlight the source code blocks in your document. The list of integrated libraries includes Rouge, CodeRay, Pygments, highlight.js, and prettify.

## 46.1. Enabling Source Highlighting

When enabled, syntax highlighting gets applied to listing or literal blocks that have the `source` block style and a source language. However, syntax highlighting is *not* enabled by default.

To enable syntax highlighting in a document, you must set the `source-highlighter` document attribute. You can set this attribute in the document or from the CLI or API.

If you set the attribute in the document, it *must* be defined in the [document header](#).

```
= Document Title  
:source-highlighter: <value>
```

For example, here's how to enable syntax highlighting using Rouge:

```
= Document Title  
:source-highlighter: rouge
```

### Source Highlighter vs. Syntax Highlighter

You might notice that the `source-highlighter` attribute uses the term “source highlighter”, whereas the library that performs the highlighting is referred to as a “syntax highlighter”. What's the difference?

- The generally accepted term for a syntax (aka code) highlighter is “syntax highlighter”.
- The syntax highlighter is applied to source blocks in AsciiDoc, hence why we say “source highlighter”.

In other words, the `source-highlighter` attribute means “use this syntax highlighter to colorize source blocks”.

## 46.2. Available Source Highlighters

The following table lists the recognized values for the `source-highlighter` attribute.

*Recognized values and supported environments for the `source-highlighter` attribute*

Library Name	Attribute Value	Supported Environments
CodeRay	coderay	Asciidoctor, AsciidoctorJ, Asciidoctor PDF
highlight.js	highlightjs	Asciidoctor, AsciidoctorJ, Asciidoctor.js
prettyfy	prettyfy	Asciidoctor, AsciidoctorJ, Asciidoctor.js
Pygments	pygments	Asciidoctor, Asciidoctor PDF
Rouge	rouge	Asciidoctor, Asciidoctor PDF

To use Rouge, CodeRay, or Pygments, you must have the appropriate library installed on your system. See the [Rouge](#), [CodeRay](#), or [Pygments](#) section to find installation instructions.



On the other hand, if you're using a client-side syntax highlighting library like highlight.js or prettyfy, there's no need to install additional libraries. The generated HTML will load the required source files from a CDN (or custom URL or file path).

## 46.3. Applying Source Highlighting

To apply highlighting to source code, you must add the `source` block style to a listing block, literal block, or paragraph and also specify a source language.

*Code block with title and syntax highlighting*

```
.app.rb ①
[#src-listing] ②
[source,ruby] ③ ④
-----
require 'sinatra'

get '/hi' do
  "Hello World!"
end
-----
```

① An optional title can be added to the block.

② An optional ID can be added to the block. See [Defining an Anchor](#).

③ Assign the block name `source` to the first position in the attribute list.

④ Assign a source language to the second position.

⑤ The source block name is typically assigned to listing and literal blocks.

*Result: Source block with title and syntax highlighting*

*app.rb*

```
require 'sinatra'

get '/hi' do
  "Hello World!"
end
```

*Source paragraph*

```
[source,xml] ①
<meta name="viewport"
  content="width=device-width, initial-scale=1.0">
```

This is normal content. ②

① Place the attribute list directly on the paragraph.

② Once an empty line is encountered the syntax highlighting is unset.

*Result: Source paragraph*

```
<meta name="viewport"
  content="width=device-width, initial-scale=1.0">
```

This is normal content.

If the majority of your source blocks use the same source language, you can set the `source-language` attribute in the document header and assign a language to it.

## *Source language attribute*

```
:source-highlighter: pygments
:source-language: java

[source]
-----
public void setAttributes(Attributes attributes) {
    this.options.put(ATTRIBUTES, attributes.map());
}
-----
```

You can override the global source language by specifying a source language on the block.

```
[source,ruby]
require 'sinatra'
```

Additionally, you can use an [include directive](#) to insert source code into an AsciiDoc document directly from a file.

## *Code block inserted from another file*

```
[source,ruby]
-----
include::app.rb[]
-----
```



If you specify custom substitutions on the source block using the `subs` attribute, make sure to include the `specialcharacters` substitution if you want to preserve syntax highlighting. However, if you do plan to modify the substitutions, we recommend using [incremental substitutions](#) instead.

## Highlighting PHP source code

The PHP language has two modes. It can either be used as a standalone language (pure mode) or it can be mixed with HTML (mixed mode) by putting it inside PHP tags (a form that resembles an XML processing instruction). This presents some challenges for the syntax highlighter.

If the code in the source block is pure PHP, you should use the language tag `php`. For example:

```
[source,php]
-----
echo "Hello, World!";
-----
```

If the PHP source is mixed with HTML, you should either use the language tag `html+php`, as shown here:

```
[source,html+php]
-----
<p>
<?php echo "Hello, World!"; ?>
</p>
-----
```

Or you could use the language tag `php` and set the `mixed` option on the source block, as shown here:

```
[source%mixed,php]
-----
<p>
<?php echo "Hello, World!"; ?>
</p>
-----
```

Under the covers, the syntax highlighter is configured to assume an implicit start PHP tag is present when the language tag is `php`. Both the `mixed` option and the language tag `html+php` disable this setting.

## 46.4. Rouge

[Rouge](#) is an elegant, extendable code highlighter written in pure Ruby that supports a vast array of languages.

In order to use Rouge with Asciidoctor, you need the [rouge](#) RubyGem. You can use one of the following methods to install Rouge.

## Install using `gem` (all systems)

```
$ gem install rouge
```

## Install using `apt-get` (Debian-based systems)

```
$ sudo apt-get install rouge
```

## Install using `dnf` (Fedora-based systems)

```
$ sudo dnf install rubygem-rouge
```

Once you've installed the RubyGem, assign the `rouge` value to the `source-highlighter` attribute in the document header to activate it.

```
:source-highlighter: rouge
```

You can further customize the source block output with additional Rouge attributes.

### `rouge-css`

Controls what method is used for applying CSS to the tokens. Can be `class` or `style`. Default: `class`.

### `rouge-linenums-mode`

Controls how line numbers are laid out. Can be `table` or `inline`. If line wrapping is enabled on preformatted blocks (i.e., `prewrap`), and you want to use line numbering on source blocks, you must set the value of this attribute to `inline` in order for the numbers to line up properly with their target lines. Default: `table`.

### `rouge-style`

Controls the color theme used to for highlighting. You can find the list of themes in the [Rouge code repository](#).

```
:source-highlighter: rouge
:rouge-linenums-mode: inline

[source,ruby,linenums]
-----
ORDERED_LIST_KEYWORDS = {
  'loweralpha' => 'a',
  'lowerroman' => 'i',
  'upperalpha' => 'A',
  'upperroman' => 'I'
  #'lowergreek' => 'a'
  #'arabic'      => '1'
  #'decimal'     => '1'
}
-----
```

#### 46.4.1. Highlight Select Lines

Not to be confused with syntax highlighting, you can configure Rouge to highlight (i.e., emphasize) select lines in order to call attention to them.

The highlight attribute accepts a comma or semi-colon delimited list of line ranges. A line range is represented by two numbers separated by a double period (e.g., 2..5). These numbers correspond to the line numbers of the source block, inclusive, where the first line is 1. This feature is activated on a source block if the highlight attribute is defined and at least one of the line numbers falls in this range.

*Highlight select lines in a source block*

```
:source-highlighter: rouge
:docinfo: private

[source,ruby,linenums,highlight=2..5]
-----
ORDERED_LIST_KEYWORDS = {
  'loweralpha' => 'a',
  'lowerroman' => 'i',
  'upperalpha' => 'A',
  'upperroman' => 'I',
}
-----
```

You will need to give it some help using extra CSS provided by the following docinfo file:

*Docinfo file to support line highlighting with Rouge*

```
<style>
pre.rouge .hll {
  background-color: #ffc;
}
pre.rouge .hll * {
  background-color: initial;
}
</style>
```

## 46.5. Pygments

Pygments is a popular syntax highlighter that supports a broad range of [programming](#), [template](#) and [markup languages](#).

In order to use Pygments with Asciidoctor, you need [Python 2](#) and the [pygments.rb gem](#).



You *do not* need to install Pygments itself. It comes bundled with the pygments.rb gem.



You must have Python 2 installed to use pygments.rb. Python 3 is not compatible with the pygments.rb gem. Check that you have a `python2` (Linux), `python` (macOS), or `py -2` (Windows) executable on your PATH. (On macOS, verify that the `python` executable uses Python 2 by running `python -V`).

*Installing Python and the pygments.rb gem via the CLI (cross platform)*

```
$ ``\which apt-get || \which dnf || \which yum || \which brew`` install python ①
$ gem install pygments.rb ②
```

① Install Python using your package manager

② Install the pygments.rb gem

Once you've installed these libraries, assign `pygments` to the `source-highlighter` attribute in your document's header.

```
:source-highlighter: pygments
```

You can further customize the source block output with additional Pygments attributes.

### **pygments-style**

Sets the name of the color theme Pygments uses. To see the list of available style names, see [Available Pygments style names](#). Default: `pastie`.

## pygments-css

Controls what method is used for applying CSS to the tokens. Can be `class` (CSS classes) or `style` (inline styles). See [Pygments stylesheet](#) to learn more about how the value `class` is handled. Default: `class`.

## pygments-linenums-mode

Controls how line numbers are arranged when line numbers are enabled on the source block. Can be `table` or `inline`. If line wrapping is enabled on preformatted blocks (i.e., `prewrap`), and you want to use line numbering on source blocks, you must set the value of this attribute to `inline` in order for the numbers to line up properly with their target lines. Default: `table`.

*Customizing a source block with Pygments attributes*

```
:source-highlighter: pygments
:pygments-style: manni
:pygments-linenums-mode: inline
```

```
[source,ruby,linenums]
```

```
-----
ORDERED_LIST_KEYWORDS = {
    'loweralpha' => 'a',
    'lowerroman' => 'i',
    'upperalpha' => 'A',
    'upperroman' => 'I'
    #'lowergreek' => 'a'
    #'arabic'      => '1'
    #'decimal'     => '1'
}
```

*Result: Source block using inline line numbers and the manni theme*

## The Manni Theme

```
1 ORDERED_LIST_KEYWORDS = {
2     'loweralpha' => 'a',
3     'lowerroman' => 'i',
4     'upperalpha' => 'A',
5     'upperroman' => 'I'
6     #'lowergreek' => 'a'
7     #'arabic'      => '1'
8     #'decimal'     => '1'
9 }
```

### 46.5.1. Highlight Select Lines

Not to be confused with syntax highlighting, you can configure Pygments to highlight (i.e., emphasize) select lines in order to call attention to them. This feature is activated on a source block if the `highlight` attribute is defined.

The `highlight` attribute accepts a comma or semi-colon delimited list of line ranges. A line range is represented by two numbers separated by a double period (e.g., `2..5`). These numbers correspond to the line numbers of the source block, inclusive, where the first line is 1.

#### *Highlight select lines in a source block*

```
:source-highlighter: pygments

[source,ruby,highlight=2..5]
-----
ORDERED_LIST_KEYWORDS = {
  'loweralpha' => 'a',
  'lowerroman' => 'i',
  'upperalpha' => 'A',
  'upperroman' => 'I',
}
-----
```

### 46.5.2. Available Pygments style names

To list the available Pygments styles, run the following command in a terminal:

```
$ $(dirname $(gem which pygments.rb))/../vendor/pygments-main/pygmentize -L styles
```

The `pygments.rb` gem uses a bundled version of Pygments (often ahead of the latest release). This command ensures that you are invoking the `pygmentize` command from the Pygments used by that gem.

### 46.5.3. Using a custom Pygments installation

If you already have Pygments installed on your system, you want to use your own fork, or you want to customize how Pygments is configured, you can get Asciidoctor to use a custom version of Pygments instead of the one bundled with the `pygments.rb` gem.

First, install your own version of Pygments. You can do this, for instance, by cloning the upstream Pygments repository:

```
$ hg clone https://bitbucket.org/birkenfeld/pygments-main pygments
```

Find the directory that contains the file `pygmentize` or the `Makefile`. That's your Pygments installation path. Make note of it.

Next, create a script to run *before* invoking Asciidoctor for the first time. Let's call it `pygments_init.rb`. Populate the script with the following content:

```
require 'pygments'

# use a custom Pygments installation (directory that contains pygmentize)
Pygments.start '/path/to/pygments'

# example of registering a missing or additional lexer
#Pygments::Lexer.create name: 'Turtle', aliases: ['turtle'],
#  filenames: ['*.ttl'], mimetypes: ['text/turtle', 'application/x-turtle']
```



You could enhance this script to read the Pygments installation path from an environment variable (or configuration file).

Now just require this script before you invoke Asciidoctor the first time. When using the `asciidoctor` command, pass the script using the `-r` flag:

```
$ asciidoctor -r ./pygments_init.rb document.adoc
```

When using the Asciidoctor API, require the script using `require` or `require_relative`:

```
require 'asciidoctor'
require_relative './pygments_init.rb'

Asciidoctor.convert_file 'document.adoc', safe: :safe
```

Now Asciidoctor is using your custom installation of Pygments instead of the one bundled with the `pygments.rb` gem.

## 46.6. CodeRay

[CodeRay](#) is an encoding-aware, syntax highlighter that supports the languages listed below.

C	C++	Clojure
CSS	Delphi	diff
ERB	Go	Groovy
HAML	HTML	Java
JavaScript	JSON	Lua
PHP	Python	Ruby
Sass	SQL	Taskpaper
XML	YAML	

In order to use CodeRay with Asciidoctor, you need the [coderay RubyGem](#). You can use one of the

following methods to install CodeRay.

### Install using `gem` (all systems)

```
$ gem install coderay
```

### Install using `apt-get` (Debian-based systems)

```
$ sudo apt-get install coderay
```

### Install using `dnf` (Fedora-based systems)

```
$ sudo dnf install rubygem-coderay
```

Once you've installed the RubyGem, assign the `coderay` value to the `source-highlighter` attribute in the document header to activate it.

```
:source-highlighter: coderay
```

You can further customize the source block output with additional CodeRay attributes.

#### `coderay-css`

Controls what method is used for applying CSS to the tokens. Can be `class` or `style`. Default: `class`.

#### `coderay-linenums-mode`

Controls how line numbers are laid out. Can be `table` or `inline`. If line wrapping is enabled on preformatted blocks (i.e., `prewrap`), and you want to use line numbering on source blocks, you must set the value of this attribute to `inline` in order for the numbers to line up properly with their target lines. Default: `table`.

```
:source-highlighter: coderay
:coderay-linenums-mode: inline

[source,ruby,linenums]
-----
ORDERED_LIST_KEYWORDS = {
  'loweralpha' => 'a',
  'lowerroman' => 'i',
  'upperalpha' => 'A',
  'upperroman' => 'I'
  #'lowergreek' => 'a'
  #'arabic'      => '1'
  #'decimal'     => '1'
}
-----
```

See the [CodeRay stylesheet section](#) to learn about the `coderay-css` attribute.

#### 46.6.1. Highlight Select Lines

Not to be confused with syntax highlighting, you can configure CodeRay to highlight (i.e., emphasize) select lines in order to call attention to them. This feature is activated on a source block if the following conditions are met:

- Line numbering is enabled (i.e., the `linenums` option). (In CodeRay, the highlight feature is coupled with line numbering).
- The `highlight` attribute is defined.

The `highlight` attribute accepts a comma or semi-colon delimited list of line ranges. A line range is represented by two numbers separated by a double period (e.g., `2..5`). These numbers correspond to the line numbers of the source block, inclusive, where the first line is 1.

*Highlight select lines in a source block*

```
:source-highlighter: coderay

[source,ruby,linenums,highlight=2..5]
-----
ORDERED_LIST_KEYWORDS = {
  'loweralpha' => 'a',
  'lowerroman' => 'i',
  'upperalpha' => 'A',
  'upperroman' => 'I',
}
-----
```

## 46.7. highlight.js

[Highlight.js](#) is a popular client-side syntax highlighter that supports a broad range of [languages](#).

To use highlight.js, simply add the following attribute entry to the header of your AsciiDoc file:

```
:source-highlighter: highlightjs
```

By default, Asciidoctor will link to the highlight.js library and stylesheet hosted on [cdnjs](#).

The version of the highlight.js library hosted on cdnjs only includes the basic language bundle. To enable support for a wider range of languages (or to use a different version), follow these steps:

1. Create your custom bundle on the [download page](#).
2. Download and unpack the zip into a folder called *highlight* adjacent to your AsciiDoc file (or in the output directory, if different)
3. Rename *highlight/highlight.pack.js* to *highlight/highlight.min.js*
4. Rename *highlight/styles/github.css* to *highlight/styles/github.min.css*
  - Replace `github` with the name of the `highlightjs-theme` you are using, if different.
5. Add the attribute entry `:highlightjsdir: highlight` to the header of your AsciiDoc file.
  - Alternatively, you can pass the `-a highlightjsdir=highlight` flag when invoking the Asciidoctor CLI.

The output file will use your personal copy of the highlight.js library and stylesheet instead of the one hosted on cdnjs.

# Chapter 47. Callouts

Callout numbers (aka callouts) provide a means to add annotations to lines in a verbatim block.

Each callout number used in a verbatim block must appear twice. The first use, which goes within the verbatim block, marks the line being annotated (i.e., the target). The second use, which goes below the verbatim block, defines the annotation text. Multiple callout numbers may be used on a single line.



The callout number (at the target) must be placed at the end of the line.

Here's a basic example of a verbatim block that uses callouts:

```
[source,ruby]
-----
require 'sinatra' ①

get '/hi' do ② ③
  "Hello World!"
end
-----
<1> Library import
<2> URL mapping
<3> Response block
```

Here's how this example gets rendered:

```
require 'sinatra' ①

get '/hi' do ② ③
  "Hello World!"
end
```

- ① Library import
- ② URL mapping
- ③ Response block

Since callout numbers can interfere with the syntax of the code they are annotating, Asciidoctor provides several features to hide the callout numbers from both the source and the converted document. The sections that follow detail these features.

## 47.1. Copy and Paste Friendly Callouts

In versions prior to Asciidoctor 0.1.4, when a reader visiting an HTML page generated by

Asciidoctor selected source code from a listing that contained callouts and copied it, the callout numbers would get caught up in the copied text. If the reader pasted that code and tried to run it, likely the extra characters from the callouts would cause compile or runtime errors.

Asciidoctor uses CSS to prevent callouts from being selected.

On the other side of the coin, you don't want the callout annotations or CSS messing up your raw source code either. You can tuck your callouts neatly behind line comments. Asciidoctor will recognize the line comments characters in front of a callout number, optionally offset by a space, and remove them when converting the document.

Here are the line comments that are supported:

```
----  
line of code ①  
line of code ②  
line of code ③  
----  
<1> A callout behind a line comment for C-style languages.  
<2> A callout behind a line comment for Ruby, Python, Perl, etc.  
<3> A callout behind a line comment for Clojure.
```

Here's how it looks when rendered:

```
line of code ①  
line of code ②  
line of code ③
```

- ① A callout behind a line comment for C-style languages.
- ② A callout behind a line comment for Ruby, Python, Perl, etc.
- ③ A callout behind a line comment for Clojure.

### 47.1.1. XML Callouts

XML doesn't have line comments, so our "tuck the callout behind a line comment" trick doesn't work here. To use callouts in XML, you must place the callout's angled brackets around the the XML comment and callout number.

Here's how it appears in a listing:

```
[source,xml]
-----
<section>
  <title>Section Title</title> ①
</section>
-----
① The section title is required.
```

Here's how it looks when rendered:

```
<section>
  <title>Section Title</title> ①
</section>
```

① The section title is required.

Notice the comment has been replaced with a circled number that cannot be selected (if not using font icons it will be rendered differently and selectable). Now both you and the reader can copy and paste XML source code containing callouts without worrying about errors.

## 47.2. Callout Icons

The font icons setting also enables callout icons drawn using CSS.

```
= Document Title
:icons: font ①
```

NOTE: Asciidoctor now supports font-based admonition icons, powered by Font Awesome!  
②

① Activates the font-based icons in the HTML5 backend.

② Admonition block that uses a font-based icon.

# Chapter 48. Conditional Preprocessor Directives

You can include or exclude lines of text in your document using one of the following conditional preprocessor directives:

- `ifdef`
- `ifndef`
- `ifeval`

These directives tell the processor whether to include the enclosed content based on certain conditions. The conditions are based on the presence or value of document attributes.

For example, say you want to include a certain section of content only when converting to HTML. Conditional preprocessor directives make this possible. You simply check for the presence of the `basebackend-html` attribute using an `ifdef` directive. Details of this example, as well as others, are described in the following sections.

## 48.1. Processing

Although a conditional preprocessor directive looks like a block macro, it's not a macro and therefore not processed like one. Instead, it's a *preprocessor* directive, just like `include`. It's important to understand the distinction.

A preprocessor directive is processed when the lines are read, but before the document structure is parsed. Therefore, it's *not* aware of the surrounding document structure. A preprocessor directive merely adds lines to the reader or takes lines away. The conditional preprocessor directives determine which lines to add and which ones to take away based on the condition.

If you don't want a conditional preprocessor directive to be processed, you must escape it using a backslash.

```
\ifdef::just-an-example[]
```

Escaping the directive is necessary *even if it appears in a verbatim block* since it's not aware of the surrounding document structure.

## 48.2. `ifdef` Directive

Content between the `ifdef` and `endif` directives gets included if the specified attribute is set:

### *ifdef example*

```
ifdef::env-github[]  
This content is for GitHub only.  
endif::[]
```

The syntax of the start directive is `ifdef::<attribute>[]`, where `<attribute>` is the name of an attribute.

Keep in mind that the content is not limited to a single line. You can have any amount of content between the `ifdef` and `endif` directives.

If you have a large amount of content inside the `ifdef` directive, you may find it more readable to use the long-form version of the directive, in which the attribute (aka condition) is referenced again in the `endif` directive.

### *ifdef long-form example*

```
ifdef::env-github[]  
This content is for GitHub only.
```

So much content in this section, I'd get confused reading the source without the closing 'ifdef' directive.

It isn't necessary for short blocks, but if you are conditionally including a section it may be something worth considering.

Other readers reviewing your docs source code may go cross-eyed when reading your source docs if you don't.  
endif::env-github[]



A great example of long-form conditional formatting is the source of this user manual! We use it to show and hide entire sections when building individual content for separate guides.

If you're only dealing with a single line of text, you can put the content directly inside the square brackets and drop the `endif` directive.

### *ifdef single line example*

```
ifdef::revnumber[This document has a version number of {revnumber}.]
```

The single-line block above is equivalent to this formal `ifdef` directive:

```
ifdef::revnumber[]  
This document has a version number of {revnumber}.  
endif::[]
```

## 48.3. `ifndef` Directive

`ifndef` is the logical opposite of `ifdef`. Content between `ifndef` and `endif` gets included only if the specified attribute is *not* set:

*ifndef Example*

```
ifndef::env-github[]  
This content is not shown on GitHub.  
endif::[]
```

The syntax of the start directive is `ifndef::<attribute>[]`, where `<attribute>` is the name of an attribute.

The `ifndef` directive supports the same single-line and long-form variants as `ifdef`.

## 48.4. Checking multiple attributes (`ifdef` and `ifndef` only)

Both the `ifdef` and `ifndef` directives accept multiple attribute names. The combinator can be “and” or “or”.

### Any attribute (or)

Multiple comma-separated (,) directive names evaluate to true, allowing the content to be included, if one or more of the directives is defined. Otherwise the content is not included.

*Any attribute example*

```
ifdef::backend-html5,backend-docbook5[Only shown when converting to HTML5 or  
DocBook 5.]
```

### All attributes (and)

Multiple plus-separated (+) directive names evaluate to true, allowing the content to be included, if all of the directives are defined. Otherwise the content is not included.

*All attributes example*

```
ifdef::env-github+backend-html5[Only shown when converting to HTML5 on GitHub.]
```

The `ifndef` directive negates the results of the expression.

Starting in Asciidoctor 1.5.6, the operator logic in the `ifndef` directive changed to align with the behavior of AsciiDoc Python. Specifically, when attributes are separated by commas, content is only included if none of the attributes are defined. When attributes are separated by pluses, content is included if at least one of the attributes is undefined. See [issue #1983](#) to find the discussion about this behavior and the rationale for the change.



## 48.5. `ifeval` directive

Content between `ifeval` and `endif` gets included if the expression inside the square brackets evaluates to true.

*ifeval example*

```
ifeval::[{sectnumlevels} == 3]
If the `sectnumlevels` attribute has the value 3, this sentence is included.
endif::[]
```

The `ifeval` directive does not have a single-line or long-form variant like `ifdef` and `ifndef`.

### 48.5.1. Anatomy

The expression consists of a left-hand value and a right-hand value with an operator in between.

*ifeval expression examples*

```
ifeval::[2 > 1]
...
endif::[]

ifeval::["{backend}" == "html5"]
...
endif::[]

ifeval::[{sectnumlevels} == 3]
...
endif::[]

// the value of outfilesuffix includes a leading period (e.g., .html)
ifeval::["{docname}{outfilesuffix}" == "master.html"]
...
endif::[]
```

### 48.5.2. Values

Each expression value can reference the name of zero or more AsciiDoc attribute using the attribute reference syntax (for example, `{backend}`).

Attribute references are resolved (substituted) first. Once attributes references have been resolved, each value is coerced to a recognized type.

When the expected value is a string (i.e., a string of characters), we recommend that you enclose the expression in quotes.

The following values types are recognized:

#### **number**

Either an integer or floating-point value.

#### **quoted string**

Enclosed in either single ('') or double ("") quotes.

#### **boolean**

Literal value of `true` or `false`.

### **How value type coercion works**

If a value is enclosed in quotes, the characters between the quotes are preserved and coerced to a string.

If a value is not enclosed in quotes, it is subject to the following type coercion rules:

- an empty value becomes nil (aka null).
- a value of `true` or `false` becomes a boolean value.
- a value of only repeating whitespace becomes a single whitespace string.
- a value containing a period becomes a floating-point number.
- any other value is coerced to an integer value.

### **48.5.3. Operators**

The value on each side is compared using the operator to derive an outcome.

`==`

Checks if the two values are equal.

`!=`

Checks if the two values are not equal.

`<`

Checks whether the left-hand side is less than the right-hand side.

`<=`

Checks whether the left-hand side is less than or equal to the right-hand side.

`>`

Checks whether the left-hand side is greater than the right-hand side.

>=

Checks whether the left-hand side is greater than or equal to the right-hand side.



The operators follow the same rules as operators in Ruby.

# Chapter 49. Docinfo Files

You can add custom content to the head or footer of an output document using docinfo files. Docinfo files are useful for injecting auxiliary metadata, stylesheet, and script information into the output not added by the converter.

Different docinfo files get used depending on whether you are converting to HTML or DocBook (but don't yet apply when converting to PDF).

## 49.1. Head docinfo files

The content of head docinfo files gets injected into the top of the document. For HTML, the content is append to the `<head>` element. For DocBook, the content is appended to the root `<info>` element.

The docinfo file for HTML output may contain valid elements to populate the HTML `<head>` element, including:

- `<base>`
- `<link>`
- `<meta>`
- `<noscript>`
- `<script>`
- `<style>`



Use of the `<title>` element is not recommend as it's already emitted by the converter.

You do not need to include the enclosing `<head>` tag as it's assumed to be the envelop.

Here's an example:

*A head docinfo file for HTML output*

```
<meta name="keywords" content="open source, documentation">
<meta name="description" content="The dangerous and thrilling adventures of an open
source documentation team.">
<link rel="stylesheet" href="basejump.css">
<script src="map.js"></script>
```

Docinfo files for HTML output must be saved with the `.html` file extension. See [Naming docinfo files](#) for more details.

The docinfo file for DocBook 5.0 output may include any of the `<info>` element's children, such as:

- `<address>`
- `<copyright>`

- <edition>
- <keywordset>
- <publisher>
- <subtitle>
- <revhistory>

The following example shows some of the content a docinfo file for DocBook might contain.

*A docinfo file for DocBook 5.0 output*

```
<author>
  <personname>
    <firstname>Karma</firstname>
    <surname>Chameleon</surname>
  </personname>
  <affiliation>
    <jobtitle>Word SWAT Team Leader</jobtitle>
  </affiliation>
</author>

<keywordset>
  <keyword>open source</keyword>
  <keyword>documentation</keyword>
  <keyword>adventure</keyword>
</keywordset>

<printhistory>
  <para>April, 2019. Twenty-sixth printing.</para>
</printhistory>
```

Docinfo files for DocBook output must be saved with the `.xml` file extension. See [Naming docinfo files](#) for more details.

To see a real-world example of a docinfo file for DocBook, checkout the [RichFaces Developer Guide docinfo file](#).

## 49.2. Footer docinfo files

Footer docinfo files are differentiated from head docinfo files by the addition of `-footer` to the file name. In the HTML output, the footer content is inserted immediately after the footer div (i.e., `<div id="footer">`). In the DocBook output, the footer content is inserted immediately before the ending tag (e.g., `</article>` or `</book>`).



One possible use of the footer docinfo file is to completely replace the default footer in the standard stylesheet. Just set the attribute `nofooter`, then apply a custom footer docinfo file.

## 49.3. Naming docinfo files

The file that gets selected to provide the docinfo depends on which converter is in use (html, docbook, etc) and whether the docinfo scope is configured for a specific document (“private”) or for all documents in the same directory (“shared”). The file extension of the docinfo file must match the file extension of the output file (as specified by the `outfilesuffix` attribute, a value which always begins with a period (.)).

### *Docinfo file naming*

Mode	Location	Behavior	Docinfo file name
Private	Head	Adds content to <code>&lt;head&gt;/&lt;info&gt;</code> for <code>&lt;docname&gt;.adoc</code> files.	<code>&lt;docname&gt;-docinfo&lt;outfilesuffix&gt;</code>
	Footer	Adds content to end of document for <code>&lt;docname&gt;.adoc</code> files.	<code>&lt;docname&gt;-docinfo-footer&lt;outfilesuffix&gt;</code>
Shared	Head	Adds content to <code>&lt;head&gt;/&lt;info&gt;</code> for any document in same directory.	<code>docinfo&lt;outfilesuffix&gt;</code>
	Footer	Adds content to end of document for any document in same directory.	<code>docinfo-footer&lt;outfilesuffix&gt;</code>

To specify which file(s) you want to apply, set the `docinfo` attribute to any combination of these values:

- `private-head`
- `private-footer`
- `private` (alias for `private-head, private-footer`)
- `shared-head`
- `shared-footer`
- `shared` (alias for `shared-head, shared-footer`)

Setting `docinfo` with no value is equivalent to setting the value to `private`.

For example:

```
:docinfo: shared,private-footer
```

This docinfo configuration will apply the shared docinfo head and footer files, if they exist, as well as the private footer file, if it exists.

`docinfo` attribute values were introduced in Asciidoctor 1.5.5 to replace the less descriptive `docinfo1` and `docinfo2` attributes. Here are the equivalents of the old attributes using the new values:



- `:docinfo: = :docinfo: private`
- `:docinfo1: = :docinfo: shared`
- `:docinfo2: = :docinfo: shared,private`

Let's apply this to an example:

You have two AsciiDoc documents, `adventure.adoc` and `insurance.adoc`, saved in the same folder. You want to add the same content to the head of both documents when they're converted to HTML.

1. Create a docinfo file containing `<head>` elements.
2. Save it as `docinfo.html`.
3. Set the `docinfo` attribute in `adventure.adoc` and `insurance.adoc` to `shared`.

You also want to include some additional content, but only to the head of `adventure.adoc`.

1. Create **another** docinfo file containing `<head>` elements.
2. Save it as `adventure-docinfo.html`.
3. Set the `docinfo` attribute in `adventure.adoc` to `shared,private-head`

If other AsciiDoc files are added to the same folder, and `docinfo` is set to `shared` in those files, only the `docinfo.html` file will be added when converting those files.

## 49.4. Locating docinfo files

By default, docinfo files are searched for in the same folder as the document file. If you want to keep them anywhere else, set the `docinfodir` attribute to their location:

```
:docinfodir: common/meta
```

Note that if you use this attribute, only the specified folder will be searched; docinfo files in the document folder will no longer be found.

## 49.5. Attribute substitution in docinfo files

Docinfo files may include attribute references. Which substitutions get applied is controlled by the `docinfosubs` attribute, which takes a comma-separated list of substitution names. The value of this attribute defaults to `attributes`.

For example, if you created the following docinfo file:

*Docinfo file containing a revnumber attribute reference*

```
<edition>{revnumber}</edition>
```

And this source document:

*Source document including a revision number*

```
= Document Title  
Author Name  
v1.0, 2019-06-01  
:doctype: book  
:backend: docbook  
:docinfo:
```

Then the converted DocBook output would be:

```
<?xml version="1.0" encoding="UTF-8"?>  
<book xmlns="http://docbook.org/ns/docbook"  
      xmlns:xlink="http://www.w3.org/1999/xlink" version="5.0" lang="en">  
  <info>  
    <title>Document Title</title>  
    <date>2019-06-01</date>  
    <author>  
      <personname>  
        <firstname>Author</firstname>  
        <surname>Name</surname>  
      </personname>  
    </author>  
    <authorinitials>AN</authorinitials>  
    <revhistory>  
      <revision>  
        <revnumber>1.0</revnumber> ①  
        <date>2019-06-01</date>  
        <authorinitials>AN</authorinitials>  
      </revision>  
    </revhistory>  
  </info>  
</book>
```

① The revnumber attribute reference was replaced by the source document's revision number in the converted output.

# Chapter 50. Counters

Counters are used to store and display ad-hoc sequences of numbers or latin characters. They are designed for simple use cases. Possible uses include inline itemized lists, paragraph numbers and serial item numbers.

A counter is implemented as a specialized document attribute. You declare and display a counter using an augmented attribute reference, in which the attribute name is prefixed with `counter:` (e.g., `{counter:name}`). Since counters are attributes, counter names follow the same rules as [attribute names](#). The most important rule to note is that letters in counter names *must be lowercase*.

The counter value is incremented and displayed every time the `counter:` attribute reference is used. The term *increment* means advance to the next value in the sequence. If the counter value is an integer, add 1. If the counter value is a character, move to the next letter in the Latin alphabet. The default start value of a counter is 1.

To create a sequence starting at 1, use the simple form `{counter:name}` as shown here:

```
The salad calls for {counter:seq1}) apples, {counter:seq1}) oranges and  
{counter:seq1}) pears.
```

Here's the resulting output:

```
The salad calls for 1) apples, 2) oranges and 3) pears.
```

To increment the counter without displaying it (i.e., to skip an item in the sequence), use the `counter2` prefix instead:

```
{counter2:seq1}
```



A `counter2` attribute reference on a line by itself will produce an empty paragraph. You'll need to adjoin it to the nearest content to avoid this side effect.

To display the current value of the counter without incrementing it, reference the counter name as you would any other attribute:

```
{counter2:pnum}This is paragraph {pnum}.
```

To create a character sequence, or start a number sequence with a value other than 1, specify a start value by appending it to the first use of the counter:

```
Dessert calls for {counter:seq1:A}) mangoes, {counter:seq1}) grapes and  
{counter:seq1}) cherries.
```



Character sequences either run from a,b,c,...x,y,z,{,|... or A,B,C,...,X,Y,Z,[,... depending on the start value. Therefore, they aren't really useful for more than 26 items.

The start value of a counter is only recognized if the counter is *unset* at that point in the document. Otherwise, the start value is ignored.

To reset a counter attribute, unset the corresponding attribute using an attribute entry. The attribute entry must be adjacent to a block or else it is ignored.

```
The salad calls for {counter:seq1:1}) apples, {counter:seq1}) oranges and {counter:seq1}) pears.
```

```
:!seq1:
```

```
Dessert calls for {counter:seq1:A}) mangoes, {counter:seq1}) grapes and {counter:seq1}) cherries.
```

This gives:

The salad calls for 1) apples, 2) oranges and 3) pears.

Dessert calls for A) mangoes, B) grapes and C) cherries.

Here's a full example that shows how to use a counter for part numbers in a table.

```
[caption=""]
.Parts{counter2:index:0}
|===
|Part Id |Description
|PX-{counter:index}
|Description of PX-{index}
|PX-{counter:index}
|Description of PX-{index}
|===
```

Here's the output of that table:

#### Parts

Part Id	Description
PX-1	Description of PX-1
PX-2	Description of PX-2

# Structuring, Navigating, and Referencing Your Content

Asciidoctor provides macros or attributes for the specialized frontmatter and backmatter sections commonly found in journal articles, academic papers, and books. In the following sections, you'll learn the dependency, structure, and output rules for the special section macros and attributes and how to use them.

# Chapter 51. Colophon

Book colophons list information such as the ISBN, publishing house, edition and copyright dates, legal notices and disclaimers, cover art, design, and book layout credits, and any significant production notes. In most mass market books, the colophon is on the verso (back side) of the title page, but it can also be located at the end of the book.

[colophon]

= Colophon

The Asciidoctor Press, Ceres and Denver

(C) 2018 by The Asciidoctor Press

Published in the Milky Way Galaxy

This book is designed by Dagger Flush, Denver, Colorado.

The types are handset Chinchilla and Dust, designed by Leeloo.

Leeloo designed the typefaces to soften the bluntness of documentation.

Created in Asciidoctor and Fedora 27.

The printing and binding is by Ceres Lithographing, Inc., Ceres, Milky Way.

# Chapter 52. Table of Contents

A table of contents (TOC) is an index of section and subsection titles that can be automatically generated from the document's structure when converting a document with Asciidoctor.

To enable the auto-generated TOC, you must set the `toc` attribute.

The `toc` attribute can be specified via the command line (`-a toc`),

*TOC enabled via the CLI*

```
$ asciidoctor -a toc adventure.adoc
```

or in the document header (`:toc:`).

*TOC enabled via the document header*

```
= The Dangerous and Thrilling Documentation Chronicles  
Kismet Chameleon  
:toc:
```

This journey begins on a bleary...

-- Cavern Glow

The river rages through the cavern, rattling its content...

When no other options are specified, the TOC is inserted directly below the document header (document title, author, and revision lines), it has the title *Table of Contents*, and contains section 1 and section 2 level titles only.

# The Dangerous and Thrilling Documentation Chronicles

Kismet Chameleon

## Table of Contents

[Cavern Glow](#)

This journey begins on a bleary...

## Cavern Glow

The river rages through the cavern, rattling its content...

Asciidoctor allows you to customize the layout, placement, and title of the table of contents. However, not all customizations are supported by all converters. See the [Table of Contents Summary](#) to find out which customizations are available to each backend.

### 52.1. In-Document Placement

`toc` values of `auto` (the default), `preamble`, and `macro` place the TOC in the main document area.

When the value of `toc` attribute is unspecified or `auto`, the TOC is inserted directly below the document header (document title, author and revision lines).

When `toc` is set to `preamble`, the TOC is placed immediately below the preamble.



When using the preamble placement, the TOC will not appear if your document does not have a `preamble`. To fix this problem, just set the `toc` attribute to an empty value (or `auto`).

To place the TOC anywhere else in the document, set the `toc` attribute to `macro`. Then, put the block toc macro (`toc::[]`) at the location in the document where you want the TOC to appear. The toc macro should appear at most once in any document.

If `toc` is not set to `macro`, any toc macro in the document will be silently ignored.

*Setting toc value to macro and using the block toc macro*

```
= The Dangerous and Thrilling Documentation Chronicles  
Kismet Chameleon  
:toc: macro ①
```

This journey begins on a bleary...

```
-- Cavern Glow
```

```
toc::[] ②
```

The river rages through the cavern, rattling its content...

① The toc attribute must be set to `macro` in order to enable the use of the toc macro.

② In this example, the toc macro is placed below the first section's title, indicating that this is the location where the TOC will be displayed once the document is rendered.

*Result: Setting toc value to macro and using the block toc macro*

# The Dangerous and Thrilling Documentation Chronicles

Kismet Chameleon

This journey begins on a bleary...

## Cavern Glow

### Table of Contents

- [Cavern Glow](#)
- [The Ravages of Writing](#)
- [A Recipe for Potion](#)
- [Dawn on the Plateau](#)

The river rages through the cavern, rattling its content...

## 52.2. Side Column Placement

When converting to HTML, you can position the TOC to the left or right of the main content column by assigning the value `left` or `right` to the `toc` attribute, respectively. In this case, the sidebar column containing the TOC is both fixed and scrollable.

### Assigning the `left` value to `toc`

```
= The Dangerous and Thrilling Documentation Chronicles  
Kismet Chameleon  
:toc: left
```

This journey begins on a bleary...

-- Cavern Glow

The river rages through the cavern, rattling its content...

-- The Ravages of Writing

Her finger socks had been vaporized by crystalline nuggets of...

--- A Recipe for Potion

Two fresh Burdockian leaves, harvested by the light of the teal moons...

### Result: Assigning the `left` value to `toc`

#### Table of Contents

[Cavern Glow](#)  
[The Ravages of Writing](#)  
[A Recipe for Potion](#)

## The Dangerous and Thrilling Documentation Chronicles

Kismet Chameleon

This journey begins on a bleary...

### Cavern Glow

The river rages through the cavern, rattling its content...

This positioning is achieved using CSS, which means it depends on support from the stylesheet. However, it's only honored if there's sufficient room on the screen for a sidebar column. If there isn't sufficient room available (i.e., the screen width falls below a certain breakpoint), the TOC will automatically shift back to the center, appearing directly below the document title.



The TOC is always placed in the center in an embeddable HTML document, regardless of the value of the `toc` attribute.

## 52.3. Title

The `toc-title` attribute allows you to change the title of the TOC.

### *Defining a new TOC title*

```
= The Dangerous and Thrilling Documentation Chronicles  
Kismet Chameleon  
:toc: ①  
:toc-title: Table of Adventures ②
```

This journey begins on a bleary...

-- Cavern Glow

The river rages through the cavern, rattling its content...

① The toc attribute must be set in order to use toc-title.

② toc-title is set and assigned the value **Table of Adventures** in the document's header.

### *Result: Defining a new TOC title*

# The Dangerous and Thrilling Documentation Chronicles

Kismet Chameleon

## Table of Adventures

[Cavern Glow](#)  
[The Ravages of Writing](#)  
[A Recipe for Potion](#)  
[Dawn on the Plateau](#)

This journey begins on a bleary...

## Cavern Glow

The river rages through the cavern, rattling its content...

## 52.4. Levels

By default, the TOC will display level 1 and level 2 section titles. You can set a different depth with the **toclevels** attribute.

## *Defining a new toclevels value*

```
= The Dangerous and Thrilling Documentation Chronicles  
Kismet Chameleon  
:toc: ①  
:toclevels: 4 ②
```

This journey begins on a bleary...

== Cavern Glow

The river rages through the cavern, rattling its content...

== The Ravages of Writing

Her finger socks had been vaporized by crystalline nuggets of...

==== A Recipe for Potion

Two fresh Burdockian leaves, harvested by the light of the teal moons...

===== Searching for Burdockian

Crawling through the twisted understory...

① The `toc` attribute must be set in order to use `toclevels`.

② `toclevels` is set and assigned the value **4** in the document's header. The TOC will list the titles of the section 1, 2, 3, and 4 levels when the document is rendered.

# The Dangerous and Thrilling Documentation Chronicles

Kismet Chameleon

## Table of Contents

[Cavern Glow](#)

[The Ravages of Writing](#)

[A Recipe for Potion](#)

[Searching for Burdockian](#)

[Dawn on the Plateau](#)

This journey begins on a bleary...

## Cavern Glow

The river rages through the cavern, rattling its content...

## 52.5. Using a TOC with Embeddable HTML

When AsciiDoc is converted to embeddable HTML (i.e., the `header_footer` option is `false`), there are only three valid values for the `toc` attribute:

- auto (or unspecified value)
- preamble
- macro

All of the following environments convert AsciiDoc to embeddable HTML:

- the file viewer on GitHub and GitLab
- the AsciiDoc preview in an editor like Atom, Brackets or AsciidocFX
- the Asciidoctor browser extensions

 The side column placement (left or right) isn't available in this mode. That's because the embeddable HTML doesn't have the outer framing (or the CSS) necessary to support a side column TOC.

## 52.6. Table of Contents Summary

*Table of contents attributes and allowed values*

Attribute	Values	Example Syntax	Comments	Backends
toc	auto, left, right, macro, preamble	:toc: left	Not set by default. Defaults to <code>auto</code> if value is unspecified.	html
	auto, macro, preamble	:toc: macro	Not set by default. Defaults to <code>auto</code> if value is unspecified.	html (embeddable)
	auto	:toc:	Not set by default. The table of contents is placed after the title page (See <a href="#">issue #233</a> for developments).	pdf
	auto	:toc:	Not set by default. The placement and styling of the table of contents is determined by the DocBook toolchain configuration.	docbook
toclevels	1–5	:toclevels: 4	Default value is 2.	html, pdf
toc-title	<text>	:toc-title: Contents	Default value is <i>Table of Contents</i> .	html, pdf

# Chapter 53. Abstract

An abstract is a concise overview of an article or of a chapter in a book. They are frequently found in the frontmatter of academic, research, and analytical papers. A complete (i.e., informative) abstract states the key topics and findings while a limited (i.e., descriptive) abstract briefly describes the structure of the content.

The abstract may be written using a section, open block, or paragraph and must bear the abstract style. If used, the abstract must appear before the first section of an article (at the start of the [preamble](#)) or at the start of a chapter in a book. An abstract may not be used *before* a part or chapter in a book.

Here's an example of an abstract at the beginning of an article, defined using a section:

```
= Article Title  
  
[abstract]  
== Abstract  
  
Documentation is a distillation of many long, squiggly adventures.  
  
== First Section
```

Here's an example of the same abstract defined using a paragraph:

```
= Article Title  
  
[abstract]  
.Abstract  
Documentation is a distillation of many long, squiggly adventures.  
  
== First Section
```

In the book doctype, the abstract section must be a level below the chapter.

```
== Chapter Title  
  
[abstract]  
==== Chapter Abstract  
  
Documentation is a distillation of many long, squiggly adventures.  
  
==== First Section
```

An abstract defined using an open block or paragraph does not require a title and does not depend on a subsequent section to terminate.

```
= Article Title
```

```
[abstract]
```

```
.Optional Abstract Title
```

```
--
```

```
This article will take you on a wonderful adventure of knowledge.
```

```
You'll start with the basics.
```

```
Beyond that, where you go is up to you.
```

```
--
```

```
Your journey begins here.
```



To include a quote at the beginning of a chapter in a book, wrap the quote block inside an abstract block.

# Chapter 54. Preface

The preface is a special section of a book that precedes the first part or chapter of the book (or the first chapter of a part). The preface can contain subsections.

An author explains how the idea behind her book manifested and developed in the preface. The preface can include acknowledgments and may be signed by the author with her name, writing date and location.

For multi-part books, the preface must be defined as a level-0 section and any subsections must start at level 2.

```
= Book Title  
:doctype: book
```

```
[preface]  
= Documentation Preface
```

The basis for this documentation germinated when I awoke one morning and was confronted by the dark and stormy eyes of the chinchilla.  
She had conquered the mountain of government reports that, over the course of six months, had eroded into several minor foothills and a creeping alluvial plain of loose papers.

```
==== Acknowledgments in Preface
```

I'd like to thank the Big Bang and String Theory.  
Would we be without them?

```
= Part 1
```

```
...
```

For books without parts, the preface must be defined as a level-1 section and any subsections must start at level 2.

```
= Book Title  
:doctype: book  
  
[preface]  
== Documentation Preface
```

The basis for this documentation germinated when I awoke one morning and was confronted by the dark and stormy eyes of the chinchilla.  
She had conquered the mountain of government reports that, over the course of six months, had eroded into several minor foothills and a creeping alluvial plain of loose papers.

```
==== Acknowledgments in Preface
```

I'd like to thank the Big Bang and String Theory.  
Would we be without them?

```
== Chapter 1
```

```
...
```

The same structure can be used to define a preface for a part.

As an alternative to a special section, Asciidoctor will automatically promote the preamble to a preface. The title of the preface is set using the **preface-title** document attribute. When defined this way, the preface may not contain subsections.

```
= Book Title  
:doctype: book  
:preface-title: Documentation Preface
```

The basis for this documentation germinated when I awoke one morning and was confronted by the dark and stormy eyes of the chinchilla.  
She had conquered the mountain of government reports that, over the course of six months, had eroded into several minor foothills and a creeping alluvial plain of loose papers.

```
== Chapter 1
```

```
...
```

The special section is the preferred way of defining a preface.

# Chapter 55. Dedication

The dedication page is where the author expresses gratitude toward a person.

[dedication]

= Dedication

For S.S.T.--

thank you for the plague of archetypes.

# Chapter 56. Book Parts and Chapters

You may group the sections of your document into parts when you set the `doctype` to `book`. In fact, parts can only be used when the `doctype` is `book`.

When a document has its `doctype` set to `book` and contains at least one part, it implicitly becomes a multi-part book. There is no dedicated doctype value for a multi-part book to distinguish it from a regular book. That distinction is made by the content.

Part titles are specified with a single equals sign (`=`), the equivalent to the structure of a document title (i.e., level-0 section). Each part must contain at least one section (a chapter or special section). Immediately after the part title, you may insert an optional introduction, which is designated by the `partintro` style.

```
[partintro]
.Optional part introduction title
--
Optional part introduction goes here.
--
```

A part can also include its own preface, bibliography, glossary and index.

```
= Title of Part 1

[partintro]
--
This is the introduction to the first part of our mud-encrusted journey.
--

== Chapter 1

There was mud...

== Chapter 2

Great gobs of mud...

[glossary]
== Part 1 Glossary

[glossary]
mud:: wet, cold dirt

= Part 2

[preface]
== Part 2 Preface

This part was written because...

== Chapter 1

The mud had turned to cement...
```

When using the PDF converter (i.e., Asciidoctor PDF), the value of the `chapter-label` attribute (followed by a space) is automatically added to the beginning of the chapter title. A chapter title is a level-1 section title when the doctype is book.

You can modify this prefix by redefining the `chapter-label` attribute.

```
:chapter-label: Chapter ~
```

To use no prefix, set the value to blank.

# Chapter 57. Appendix

You can indicate that a section, part, or chapter is an appendix by adding an [appendix] line above the section title.

```
[appendix]
== Copyright and License
```

While the AsciiDoc structure allows appendices to be placed anywhere, it's customary to place them at the end of the document, after all other sections.

Sections marked as appendix have a different title, which is built as follows:

- A fixed prefix (taken from the value of the `appendix-caption` attribute, which defaults to "Appendix")
- A letter that represents the number of this appendix within the sequence of appendices (A, B, C, ...)
- A colon
- The original section title

For example:

```
Appendix A: Copyright and License
```

The prefix can be modified by defining the `appendix-caption` attribute. This value can be changed using:

```
:appendix-caption: Appx
```

Or it can be unset using:

```
:appendix-caption!: 
```

Appendices can have subsections. However, there are some rules to follow depending on which type of document you are creating.

For articles, the appendix must be defined as a level-1 section. (If a level-0 section is used, a level-1 section is implied). For example:

```
= Article Title
:appendix-caption: Appx
:sectnums:
:toc:

== First Section

==== Subsection

[appendix]
== First Appendix

==== First Subsection

==== Second Subsection

[appendix]
== Second Appendix
```

The table of contents will appear as follows:

```
1. First Section
  1.1. Subsection
Appx A: First Appendix
  1.1. First Subsection
  1.2. Second Subsection
Appx B: Second Appendix
```

For books, the appendix must be defined as a level-1 section if you want the appendix to be adjacent to other chapters (and in the current part if the book has multiple parts). In a multi-part book, if you want the appendix to be adjacent to other parts, the appendix must be defined as a level-0 section. In either case, the first subsection of the appendix must be a level-3 section, not a level-2 section. That's because an appendix cannot contain chapters.

The following example shows how to define an appendix for a multi-book (take away the part title if you're creating a simple book):

```
= Book Title  
:doctype: book  
:appendix-caption: Appx  
:sectnums:  
:toc:  
  
= First Part  
  
== First Chapter  
  
==== Subsection  
  
[appendix]  
= First Appendix  
  
==== First Subsection  
  
==== Second Subsection  
  
[appendix]  
= Second Appendix
```

The table of contents will appear as follows:

```
First Part  
1. First Chapter  
    1.1. Subsection  
Appx A: First Appendix  
    1.1. First Subsection  
    1.2. Second Subsection  
Appx B: Second Appendix
```

# Chapter 58. Glossary

You can include a glossary of definitions by including the [glossary] marker before the section header and before the first definition.

```
[glossary]
```

```
== Glossary
```

```
[glossary]
```

```
mud:: wet, cold dirt
```

```
rain::
```

```
water falling from the sky
```

# Chapter 59. Bibliography

AsciiDoc has basic support for bibliographies. AsciiDoc doesn't concern itself with the structure of the bibliography entry itself, which is entirely freeform. What it does is provide a way to make references to the entries from the same document and output the bibliography with proper semantics for processing by other toolchains (such as DocBook).

In order to reference a bibliography entry, you need to assign a *non-numeric* label to the entry. To assign this label, prefix the entry with the label enclosed in a pair of triple square brackets (e.g., `[[[label]]]`). We call this a bibliography anchor. Using this label, you can then reference the entry from anywhere above the bibliography in the same document using the normal cross reference syntax (e.g., `<<label>>`).

By default, the bibliography anchor and reference to the bibliography entry is converted to `[<label>]`, where `<label>` is the ID of the entry. If you specify `xreftext` on the bibliography anchor (e.g., `[[[label,xreftext]]]`), the bibliography anchor and reference to the bibliography entry converts to `[<xreftext>]` instead.

If you want the bibliography anchor and reference to appear as a number, assign the number of the entry using the `xreftext`. For example, `[[[label,1]]]` will be converted to `[1]`.

The bibliography entries themselves are declared as items in an unordered list. You promote a normal unordered list to a bibliography list by adding the `bibliography` style. However, to be conforming, bibliography lists must be contained inside a bibliography section. A bibliography section is a section with the special `bibliography` style. By adding the `bibliography` style to the section, you implicitly add it to each unordered list in that section.



If the `bibliography` style is used on an unordered list outside of a bibliography section, the resulting behavior is undefined.

Let's consider an example.

## Bibliography with inbound references

```
_The Pragmatic Programmer_ <<pp>> should be required reading for all developers.  
To learn all about design patterns, refer to the book by the "Gang of Four" <<gof>>.  
  
[bibliography]  
== References  
  
- [[[pp]]] Andy Hunt & Dave Thomas. The Pragmatic Programmer: From Journeyman to Master. Addison-Wesley. 1999.  
- [[[gof,2]]] Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. 1994.
```

This renders as:

*The Pragmatic Programmer* [pp] should be required reading for all developers. To learn all about design patterns, refer to the book by the “Gang of Four” [2].

## References

- [pp] Andy Hunt & Dave Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley. 1999.
- [2] Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley. 1994.

If you want more advanced features such as automatic numbering and custom citation styles, try the [asciidoc-bibtex](#) project.



To escape a bibliography anchor anywhere in the text, use the syntax `[[[word]]]`. This prevents the anchor from being matched as a bibliography anchor or a normal anchor.

# Chapter 60. Index

You can mark index terms explicitly in AsciiDoc content. Index terms form a controlled vocabulary that can be used to navigate the document by keyword starting from an index.

## 60.1. Index Terms

There are two types of index terms in AsciiDoc:

### flow index term

`indexterm2:[<primary>] or ((<primary>))`

An index term that appears in the flow of text (i.e., a visible term) and in the index. This type of index term can only be used to define a primary entry.

### concealed index term

`indexterm:[<primary>, <secondary>, <tertiary>] or (((<primary>, <secondary>, <tertiary>)))`

A group of index terms that appear only in the index. This type of index term can be used to define a primary entry as well as optional secondary and tertiary entries.

Here's an example that shows the two forms in use.

```
The Lady of the Lake, her arm clad in the purest shimmering samite,  
held aloft Excalibur from the bosom of the water,  
signifying by divine providence that I, ((Arthur)), ①  
was to carry Excalibur (((Sword, Broadsword, Excalibur))). ②  
That is why I am your king. Shut up! Will you shut up?!  
Burn her anyway! I'm not a witch.  
Look, my liege! We found them.
```

```
indexterm2:[Lancelot] was one of the Knights of the Round Table. ③  
indexterm:[knight, Knight of the Round Table, Lancelot] ④
```

- ① The double parenthesis form adds a primary index term and includes the term in the generated output.
- ② The triple parenthesis form allows for an optional second and third index term and *does not* include the terms in the generated output (i.e., concealed index term).
- ③ The inline macro `indexterm2:[primary]` is equivalent to the double parenthesis form.
- ④ The inline macro `indexterm:[primary, secondary, tertiary]` is equivalent to the triple parenthesis form.

If you're defining a concealed index term (i.e., the `indexterm` macro), and one of the terms contains a comma, you must surround that segment in double quotes so the comma is treated as content. For example:

```
I, King Arthur.  
indexterm:[knight, "Arthur, King"]
```

or

```
I, King Arthur.  
(((knight, "Arthur, King")))
```

## 60.2. Index Catalog

Although index terms are always processed, only Asciidoctor PDF and the DocBook toolchain support creating an index catalog automatically. To create an index, define a level-1 section marked with the style **index** at the end of your document.

```
[index]  
== Index
```

Both Asciidoctor PDF and the DocBook toolchain will automatically populate an index into this seed section.



The built-in HTML5 converter in Asciidoctor does not currently generate an index. Follow [issue #450](#) to track the progress of this feature.

# Chapter 61. Footnotes

You can add footnotes to your document using the footnote macro (`footnote:[]`). If you plan to reference a footnote more than once, use the footnote macro with a target (`footnote:id[]`).

## *Footnote syntax*

```
The hail-and-rainbow protocol can be initiated at five levels:  
double, tertiary, supernumerary, supermassive, and apocalyptic party.footnote:[The  
double hail-and-rainbow level makes my toes tingle.] ① ②  
A bold statement!footnote:disclaimer[Opinions are my own.] ③
```

```
Another outrageous statement.footnote:disclaimer[] ④
```

- ① Insert the footnote macro directly after any punctuation. Note that the footnote macro only uses a single colon (:).
- ② Insert the footnote's content within the square brackets ([]). The text may span several lines.
- ③ If you plan to reuse a footnote, specify a unique ID in the target position.
- ④ To reference an existing footnote, you only need to insert the ID in the target slot (leaving the text in the square brackets empty).

The footnotes will be numbered consecutively throughout the article.

## *Rendered footnotes*

The hail-and-rainbow protocol can be initiated at five levels: double, tertiary, supernumerary, supermassive, and apocalyptic party.<sup>[4]</sup> A bold statement!<sup>[5]</sup>

Another outrageous statement.<sup>[3]</sup>

## 61.1. Externalizing a Footnote

Since footnotes are defined using an inline macro, the footnote content ends up right alongside the text it's annotating, making that text harder to read. You can solve this probably by externalizing your footnotes using document attributes.

Here's the previous example with the footnotes defined in document attributes and inserted using attribute references. Notice you still get the benefit of seeing where the footnote is placed without all the noise.

### *Externalized footnote*

```
:fn-hail-and-rainbow: footnote:[The double hail-and-rainbow level makes my toes tingle.]
```

```
:fn-disclaimer: footnote:disclaimer[Opinions are my own.]
```

The hail-and-rainbow protocol can be initiated at five levels:  
double, tertiary, supernumerary, supermassive, and apocalyptic party.{fn-hail-and-rainbow}

A bold statement!{fn-disclaimer}

Another outrageous statement.{fn-disclaimer}

This approach works since attribute references are expanded before footnotes are parsed. The redundant text in the second disclaimer footnote is simply ignored.

You could even simplify the attribute names to fn-1, fn-2, etc. if brevity is your main concern. And once in document attributes, these definitions could even be moved to an included file.

## 61.2. Footnotes in Headings

If you use a footnote in a heading, you'll likely find that the footnote sequence is wrong. That's because headings (section titles and discrete headings) get converted out of document order for the purpose of generating IDs and setting up cross references.

To resolve this problem, you must assign an explicit ID and reftext to any heading that contains a footnote. For example:

```
See <<heading>>.
```

```
[[heading,Heading]]
```

```
-- Heading{blank}footnote:[This is a heading with a footnote]
```

Not only will this prevent the footnote from being processed out of document order (and thus assigned the wrong number in the sequence), it will also prevent the footnote number from reappearing in the text of an xref.

# Processing Your Content

While the AsciiDoc syntax is designed to be readable in raw form, the intended audience for that format are writers and editors. Readers aren't going to appreciate the raw text nearly as much. Aesthetics matter. You'll want to apply nice typography with font sizes that adhere to the golden ratio, colors, icons and images to give it the respect it deserves. That's where converters, backends, and themes come into play.

# Chapter 62. Selecting an Output Format

The Asciidoctor processor is typically used to parse an AsciiDoc document and convert it to a variety of formats, such as HTML, DocBook, or PDF. This section describes how to specify the output format.

The processor generates the output format using a converter. Each converter is mapped to the name of a backend. You specify the backend—and, in turn, the converter—using the `-b (--backend)` command line option or `backend` API option. If no backend is specified, the processor uses the converter for the default backend (`html5`).

Asciidoctor provides several built-in converters, which are mapped to the following backend names:

## **html (or html5)**

HTML 5, styled with CSS3 (default).

## **xhtml (or xhtml5)**

The XHTML variant of the output from `html5`.

## **docbook (or docbook5)**

DocBook 5.0 XML.

## **manpage**

Manual pages for Unix and Unix-like operating systems.

Asciidoctor also has several add-on converters, which can be plugged in by adding the appropriate library to the runtime path (e.g., `-r asciidoctor-pdf`). These converters are mapped to the following backend names:

## **pdf**

PDF, a portable document format. Requires the `asciidoctor-pdf` gem.

## **epub3**

EPUB3, a distribution and interchange format standard for digital publications and documents. Requires the `asciidoctor-epub3` gem.

## **latex**

LaTeX, a document preparation system for high-quality typesetting. Requires the `asciidoctor-latex` gem.

## **mallard**

Mallard 1.0 XML. Requires the `asciidoctor-mallard` gem (not yet released).

There are also converters available for HTML5 presentation systems such as reveal.js and Bespoke.js. Those converters are still in development and will be documented once releases become available.

# Chapter 63. HTML

Asciidoctor's default output format is HTML.

## html5

HTML 5 markup styled with CSS3.

## 63.1. Using the Command Line

In this section, we'll create a sample document, then process and convert it with Asciidoctor's `html5` converter.

1. Create an AsciiDoc file like the one below
2. Save the file as `mysample.adoc`

```
= My First Experience with the Dangers of Documentation
```

```
In my world, we don't have to worry about mutant, script-injecting warlocks.  
No.
```

```
We have something far worse.  
We're plagued by Wolpertingers.
```

```
== Origins
```

```
You may not be familiar with these https://en.wikipedia.org/wiki/Wolpertinger[ravenous  
beasts], but, trust me, they'll eat your shorts and suck the loops from your code.
```

To convert `mysample.adoc` to HTML from the command line:

1. Open a console
2. Switch to the directory that contains the `mysample.adoc` document
3. Call the Asciidoctor processor with the `asciidoctor` command, followed by the name of the document you want to convert

```
$ asciidoctor mysample.adoc
```

Remember, Asciidoctor's default converter is `html5`, so it isn't necessary to specify it with the `-b` command.

You won't see any messages printed to the console. If you type `ls` or view the directory in a file manager, there is a new file named `mysample.html`.

```
$ ls  
mysample.adoc  mysample.html
```

Asciidoctor derives the name of the output document from the name of the input document.

Open *mysample.html* in your web browser. Your document should look like the image below.

# My First Experience with the Dangers of Documentation

In my world, we don't have to worry about mutant, script-injecting warlocks. No. We have something far worse. We're plagued by Wolpertingers.

## Origins

You may not be familiar with these [ravenous beasts](#), but, trust me, they'll eat your shorts and suck the loops from your code.

The document's text, titles, and link is styled by the default Asciidoctor stylesheet, which is embedded in the HTML output. As a result, you could save *mysample.html* to any computer and it will look the same.

## 63.2. Using the Ruby API

Asciidoctor also includes a Ruby API that lets you generate an HTML document directly from a Ruby application.

```
require 'asciidoctor'

Asciidoctor.convert_file 'mysample.adoc'
```

Alternatively, you can capture the HTML output in a variable instead of writing it to a file.

```
html = Asciidoctor.convert_file 'mysample.adoc', to_file: false, header_footer: true
puts html
```

The convert methods also accept a map of options. Use of this map is described in [Ruby API Options](#).

## 63.3. Styling the HTML with CSS

Asciidoctor uses CSS for HTML document styling and JavaScript for generating document attributes such as a table of contents and footnotes. It comes bundled with a stylesheet, named *asciidoctor.css*. When you generate a document with the `html5` backend, the *asciidoctor.css* stylesheet is embedded into the HTML output by default (when the safe mode is less than `SECURE`).

You have the option of linking to the stylesheet instead of embedding it. This is the mandatory

behavior when the safe mode is SECURE. If your stylesheet is being linked when you expect it to be embedded, lower the safe mode (`safe` is the recommend value).

To have your document link to the stylesheet, set the `linkcss` attribute in the document's header.

```
= My First Experience with the Dangers of Documentation  
:linkcss:
```

In my world, we don't have to worry about mutant, script-injecting warlocks.

No.

We have something far worse.

We're plagued by Wolpertingers.

```
-- Origins
```

You may not be familiar with these <https://en.wikipedia.org/wiki/Wolpertinger>[ravenous beasts], but, trust me, they'll eat your shorts and suck the loops from your code.

You can also set `linkcss` with the CLI.

```
$ asciidoctor -a linkcss mysample.adoc
```

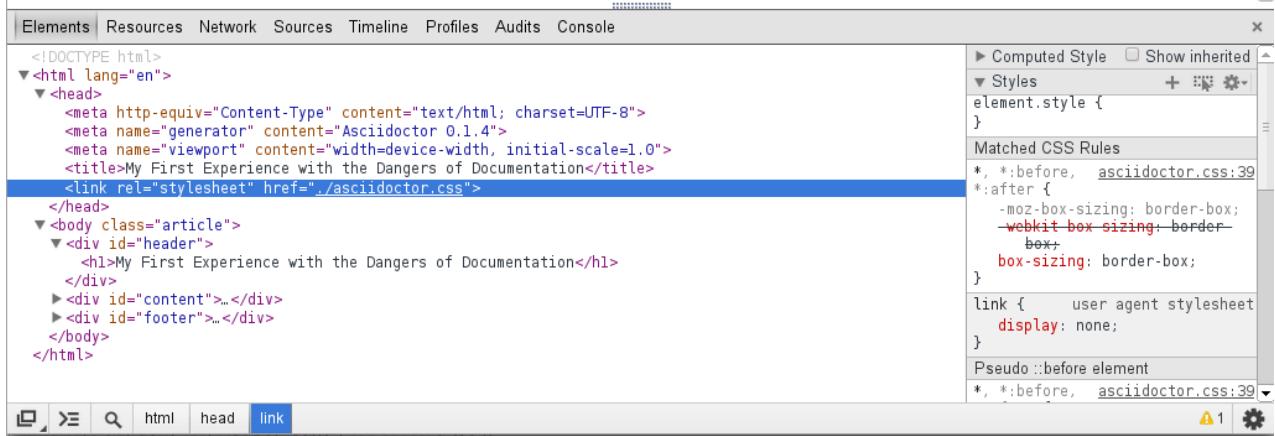
Now, when you view the directory, you should see the file `asciidoc.css` in addition to `mysample.adoc` and `mysample.html`. The `linkcss` attribute automatically copies `asciidoc.css` to the output directory. Additionally, you can inspect `mysample.html` in your browser and see `<link rel="stylesheet" href=".asciidoc.css">` inside the `<head>` tags.

# My First Experience with the Dangers of Documentation

In my world, we don't have to worry about mutant, script-injecting warlocks. No. We have something far worse. We're plagued by Wolpertingers.

## Origins

You may not be familiar with these [ravenous beasts](#), but, trust me, they'll eat your shorts and suck the loops from your code.



If you don't want any styles applied to the HTML output of your document, unset the `stylesheet` attribute.

```
$ asciidoctor -a stylesheet! mysample.adoc
```

One of Asciidoc's strengths is the ease in which you can swap the default stylesheet for an [alternative Asciidoc theme](#) or use your own [custom stylesheet](#).

## 63.4. Managing Images

Images are not embedded in the HTML output by default. If you have image references in your document, you'll have to save the image files in the same directory as your converted document.

As an alternative, you can embed the images directly into the document by setting the `data-uri` document attribute.

```
$ asciidoctor -a data-uri mysample.adoc
```

```
= My First Experience with the Dangers of Documentation
```

```
:imagesdir: myimages
```

```
:data-uri:
```

In my world, we don't have to worry about mutant, script-injecting warlocks.

No.

We have something far worse.

We're plagued by Wolpertingers.

-- Origins

```
[.left.text-center]
```

```
image::wolpertinger.jpg[Wolpertinger]
```

You may not be familiar with these <https://en.wikipedia.org/wiki/Wolpertinger> [ravenous beasts], but, trust me, they'll eat your shorts and suck the loops from your code.

## My First Experience with the Dangers of Documentation

In my world, we don't have to worry about mutant, script-injecting warlocks. No. We have something far worse. We're plagued by Wolpertingers.

### Origins



You may not be familiar with these [ravenous beasts](#), but, trust me, they'll eat your shorts and suck the loops from your code.

The screenshot shows the Chrome DevTools Elements tab. The DOM tree on the left shows the following structure:

```
<h2 id="_origins">Origins</h2>
  <div class="sectionbody">
    <div class="imageblock left text-center">
      <div class="content">
        
      </div>
    </div>
  </div>
```

The CSS panel on the right shows a single rule for the image element:

```
element.style {
}
```

Matched CSS Rules

```
img { mysample-uri-data.html:64 }
```

If the target of one or more images in the document is a URI, you must also set the `allow-uri-read` attribute securely and run Asciidoctor in **SECURE** mode or less.

```
$ asciidoctor -a data-uri -a allow-uri-read mysample.adoc
```

The same is true when converting the document to PDF using Asciidoctor PDF, even if the `allow-uri-read` attribute is not set (since the behavior is implied).

## 63.5. CodeRay and Pygments Stylesheets

Asciidoctor will also embed the theme stylesheet for the CodeRay or Pygments syntax highlighter.

### *CodeRay*

If the `source-highlighter` attribute is `coderay` and the `coderay-css` attribute is `class`, the CodeRay stylesheet is:

- *embedded* by default
- *copied* to the file `asciidoctor-coderay.css` inside the `stylesdir` folder within the output directory if `linkcss` is set

### *Pygments*

If the `source-highlighter` attribute is `pygments` and the `pygments-css` attribute is `class`, the Pygments stylesheet is:

- *embedded* by default
- *copied* to the file `asciidoctor-pygments.css` inside the `stylesdir` folder within the output directory if `linkcss` is set

# Chapter 64. XHTML

To convert AsciiDoc to XHTML, set the backend to `xhtml15`.

*Produce XHTML using the built-in HTML converter*

```
asciidoc -b xhtml15 document.adoc
```

To produce XHTML instead of HTML when using converter templates, set the `htmlsyntax` attribute to `xml` in addition to the backend option:

*Produce XHTML using custom templates*

```
asciidoc -T /path/to/templates -b slides -a htmlsyntax=xml document.adoc
```

## Black Box Decoded: XHTML and `htmlsyntax`

XHTML output is a special mode of the built-in HTML5 converter. It is activated by prefixing the backend value with `x` (e.g., `xhtml` or `xhtml15`). This hint implicitly assigns the `htmlsyntax` attribute to the value `xml`, which normally has the value `html`.

For all other converters, the `htmlsyntax` attribute is not set explicitly. If you want a converter template that's written in Slim or Haml to output XHTML instead of the default HTML, you simply need to set the `htmlsyntax` attribute to `xml` explicitly. Asciidoctor will pass on this preference to the Slim or Haml template engine by setting the `:format` option to `:html5`.

# Chapter 65. DocBook

Asciidoctor can produce DocBook 5.0 output. Since the AsciiDoc syntax was designed with DocBook output in mind, the conversion is very good. There's a corresponding DocBook element for each markup in the AsciiDoc syntax.

To convert the `mysample.adoc` document to DocBook 5.0 format, call the processor with the backend flag set to `docbook`.

```
$ asciidoctor -b docbook mysample.adoc
```

A new XML document, named `mysample.xml`, will now be present in the current directory.

```
$ ls  
mysample.adoc  mysample.html  mysample.xml
```

Here's a snippet of the XML generated by the DocBook converter.

## *XML generated from AsciiDoc*

```
<?xml version="1.0" encoding="UTF-8"?>
<article xmlns="http://docbook.org/ns/docbook"
    xmlns:xl="http://www.w3.org/1999/xlink" version="5.0" xml:lang="en">
<info>
    <title>Hello, AsciiDoc!</title>
    <date>2013-09-03</date>
    <author>
        <personname>
            <firstname>Doc</firstname>
            <surname>Writer</surname>
        </personname>
        <email>doc@example.com</email>
    </author>
    <authorinitials>DW</authorinitials>
</info>
<simpara>
    An introduction to <link xl:href="http://asciidoc.org">AsciiDoc</link>.
</simpara>
<section xml:id="_first_section">
    <title>First Section</title>
    <itemizedlist>
        <listitem>
            <simpara>item 1</simpara>
        </listitem>
        <listitem>
            <simpara>item 2</simpara>
        </listitem>
    </itemizedlist>
</section>
</article>
```

If you're on Linux, you can view the DocBook file with [Yelp](#).

```
$ yelp mysample.xml
```

And of course, if you're using the Asciidoc Ruby API, you can generate a DocBook document directly from your application.

*Generate DocBook output from the API*

```
Asciidoctor.convert_file 'mysample.adoc', backend: 'docbook'
```

By default, the docbook converter produces DocBook 5.0 output that is compliant to the DocBook 5.0 specification.

A summary of the differences are as follows:

- XSD declarations are used on the document root instead of a DTD
- `<info>` elements for document info instead of `<articleinfo>` and `<bookinfo>`
- elements that hold the author's name are wrapped in a `<personname>` element
- the id for an element is defined using an `xml:id` attribute
- `<link>` is used for links instead of `<ulink>`
- the URL for a link is defined using the `xl:href` attribute

Refer to [What's new in DocBook v5.0?](#) for more details about how DocBook 5.0 differs from DocBook 4.5.

If you need to output DocBook 4.5, you may find the community-supported [DocBook 4.5 Converter](#) useful.

```
$ asciidoctor -b docbook45 mysample.adoc
```

# Chapter 66. Man Pages

One of the more specialized uses of AsciiDoc is to serve as a shorthand for generating man pages (short for manual pages) for Unix and Unix-like operating systems.

A man page is coded in the roff typesetting language. By adhering to a specific structure in the man page file, the `man` command can parse the content and present a formatted document in a textual (command line) pager. Manual pages provide a unified help catalog for all commands in the system. For a full description, see the [roff man page](#) (or type `man roff` or `man 7 man`).

Asciidoctor can produce roff-formatted man pages if the structure of the AsciiDoc document adheres to the `manpage` doctype structure (described later in this section).

To produce a roff-formatted man page from an AsciiDoc file that declares the `manpage` doctype, run:

```
$ asciidoctor -b manpage source.adoc
```

Note that the man page converter sets the output file name to `progname.1`, where `progname` is the name of the command and `1` is the volume number, as defined by the doctitle of the source document.

When converting to the man page format, Asciidoctor uppercases the titles of all level-0 and level-1 sections. This saves you from having to type section titles in all uppercase. It also makes the document portable to other output formats since this style is only needed in the man page output to align with conventions.



If you're use Ruby 2.4 or better, Asciidoctor will uppercase any letter in the title that is recognized by the Unicode specification as having an uppercase equivalent (which includes non-Latin letters). Prior to Ruby 2.4, Ruby could only uppercase Latin letters.

Asciidoctor can also produce HTML and PDF versions similar to the `man` output for viewing in other contexts. To see the man page in HTML instead, run:

```
$ asciidoctor source.adoc
```

Here is an example man page composed in AsciiDoc for the `eve` command:

```

= eve(1)
Andrew Stanton
v1.0.0
:doctype: manpage
:manmanual: EVE
:mansource: EVE
:man-linkstyle: pass:[blue R < >]

== Name

eve - analyzes an image to determine if it's a picture of a life form

== Synopsis

*eve* [_OPTION_]... _FILE_...

== Options

*-o, --out-file*=_OUT_FILE_::
  Write result to file _OUT_FILE_.

*-c, --capture*::
  Capture specimen if it's a picture of a life form.

== Exit status

*0*::
  Success.
  Image is a picture of a life form.

*1*::
  Failure.
  Image is not a picture of a life form.

== Resources

*Project web site:* http://eve.example.org

== Copying

Copyright (C) 2008 {author}. +
Free use of this software is granted under the terms of the MIT License.

```

The manpage doctype has the following required parts:

## Document Header

A man page document header is mandatory. The title line contains the man page name followed immediately by the manual section number in round brackets. The title name should not contain white space. The manual section number is a single digit optionally followed by a single

character.

## The NAME Section

The first man page section is mandatory, must be titled “NAME” and must contain a single paragraph (usually a single line) consisting of a list of one or more comma separated command name(s) separated from the command purpose by a dash character. The dash must have at least one white space character on either side.

## The SYNOPSIS Section

The second man page section is mandatory and must be titled “SYNOPSIS”.

Subsequent sections are optional, but typical sections include “SEE ALSO”, “BUGS REPORTS”, “AUTHORS” and “COPYRIGHT”.

There are several built-in document attributes that affect only man pages. If used, they must be set in the document header.

*Built-in document attributes specific to the manpage doctype*

Attribute name	Description	Value (as parsed from example above)
mantitle	Alternative way to set the man page name.	ASCIIDOCTOR(1)
manvolnum	Manual section number.	1
manname	Alternative way to set the command name.	asciidoc
manpurpose	Alternative way to set the command purpose.	converts AsciiDoc source files
man-linkstyle	Style the links in the manpage output. A valid link format sequence.	blue R <>
mansource	The source to which the man page pertains. When producing DocBook, it becomes a DocBook <a href="#">refmiscinfo</a> attribute and appears in the footer.	Asciidoc
manversion	The version of the man page. Defaults to revnumber if not specified. When producing DocBook, it becomes a DocBook <a href="#">refmiscinfo</a> attribute and appears in the footer. Not used by Asciidoc.	1.5.4
manmanual	Manual name. When producing DocBook, it becomes a DocBook <a href="#">refmiscinfo</a> attribute and appears in the footer.	Asciidoc Manual

Refer to [the AsciiDoc source of the Asciidoc man page](#) to see a complete example. The man page

for Asciidoc is produced using the built-in man page converter in Asciidoc. The man pages for git are also produced from AsciiDoc documents, so you can use those as another example to follow.

# Chapter 67. PDFs

Conversion from AsciiDoc to PDF is made possible by a number of tools.

## Asciidoctor PDF

A native PDF converter for Asciidoctor (converts directly from AsciiDoc to PDF using Prawn).

Instructions for installing and using Asciidoctor PDF are documented in the project's [README](#). The tool provides built-in theming via a YAML configuration file, which is documented in the [theming guide](#).



Asciidoctor PDF is the preferred tool for converting to PDF and is fully supported by the Asciidoctor community.

## a2x

A DocBook toolchain frontend provided by that AsciiDoc Python project.

To use this tool, you should first convert to DocBook using Asciidoctor, then convert the DocBook to PDF using a2x. a2x accepts a DocBook file as input and can convert it to a PDF using either Apache FOP or dblatex. Instructions for using a2x are documented in the project's [man page](#).

## asciidoctor-fopub

A DocBook toolchain frontend similar to a2x, but which only requires Java to be installed on your machine.

Instructions for using asciidoctor-fopub are documented in the project's [README](#). To alter the look and feel of the PDF, it's necessary to pass XSL parameters or modify the XSLT. More information about customization can be found in [DocBook XSL: The Complete Guide](#).

# Chapter 68. Preview Your Content

## 68.1. Guard/Live Viewer

### Guard

AsciiDoc is a plugin for [Guard](#) that converts watched AsciiDoc files to the specified output format whenever they change using Asciidoctor.

Instructions for using the Guard/Live viewer are documented in the project's [README](#).

# Chapter 69. CLI Inputs and Outputs

So far, you've seen how to pass a single input file to the CLI for it to convert. You can also convert multiple files at once, read content from STDIN, write content to STDOUT, or write content to a different output file or directory. This section explains the various input and output options of the CLI.

## 69.1. Process Multiple Source Files from the CLI

You can pass multiple source files or a filename pattern to the Asciidoctor CLI and it will convert each file given in turn.

Let's assume there exist two AsciiDoc files in the current directory, *a.adoc* and *b.adoc*. You can pass both files to Asciidoctor using a single command, as follows:

```
$ asciidoctor a.adoc b.adoc
```

Asciidoctor will convert both files, transforming *a.adoc* to *a.html* and *b.adoc* to *b.html*.

To save some typing, you can use the glob operator (\*) to match all AsciiDoc files in the current directory using a single argument:

```
$ asciidoctor *.adoc
```

Your shell will automatically expand the pattern and interpret the command exactly as you had typed it above:

```
$ asciidoctor a.adoc b.adoc
```

You can pass all AsciiDoc files inside direct subfolders using the glob operator (\*) in place of the directory name:

```
$ asciidoctor */*.adoc
```

To match all files in the current directory and direct subfolders, combine both glob patterns:

```
$ asciidoctor *.adoc */*.adoc
```

Since the globs in this command rely on shell expansion, the command is not portable across platforms. To make it portable, you can allow the Asciidoctor CLI to expand the globs. To do so, instruct the shell to not expand the glob by quoting the pattern, as shown here:

```
$ asciidoctor '*.adoc' '**/*.adoc'
```

This time, the arguments `*.adoc` and `**/*.adoc` are passed directly to Asciidoctor instead of being expanded. Asciidoctor handles the glob matching in a manner that is portable across platforms.



If you process multiple nested AsciiDoc files at once and are also applying a custom stylesheet to them, you'll need to [manage the stylesheet's location](#).

But it gets better. The glob handling in Asciidoctor (which matches the rules of file globbing in Ruby) is likely more powerful than what your shell offers. For example, you can match AsciiDoc files in the current folder and in folders of any depth using the double glob operator (\*\*).

```
$ asciidoctor '**/*.adoc'
```

Most shells do not honor the double glob pattern. Thus, to ensure both portability and flexibility when specifying a glob pattern, always enclose the argument in quotes.

If you specify an output folder, the source folders are not preserved by default. To preserve the folders, specify which portion of the path to remove using the `-R` flag:

```
$ asciidoctor -D out -R src 'src/**/*.adoc'
```

In this case, the folder structure under the `src` folder will be replicated.

You can use the special value `.` to indicate you want the folder structure of the source and output to be mirrored exactly. For example:

```
$ asciidoctor -D out -R . '**/*.adoc'
```

If you find yourself pushing the boundaries of what can be achieved using the CLI to convert multiple AsciiDoc files (including processing resources like images), consider switching to a build tool like Maven or Gradle. Both tools provide excellent Asciidoctor integration. Refer to the [Asciidoctor Maven plugin](#) or [Asciidoctor Gradle plugin](#) for details.

## 69.2. Specifying an Output File

By default, the CLI writes the converted output to the same directory as the input file and with the same name as the input file, except with a file extension that matches the output format (e.g., replacing `.adoc` with `.html`). You can override this default and have the CLI write to a file and directory of your choice.

There are several circumstances when you'll want to specify a different output file:

- You want to write the output file to a different name, perhaps to append a qualifier such as a version string.

- You want to write the output file to a different directory.
- You are piping content to the CLI, but want to write the output to a file (in this case, an output file is required).



By specifying an output file, you implicitly overwrite the output directory as well. The output file will be resolved relative to the current working directory, not the directory of the input file.

To specify the output file, you'll use the `-o` option. For example, let's say we want to convert `mydoc.adoc` and write the output to a filename that includes the current date. You'd use:

```
$ asciidoctor -o mydoc-$(date +%Y-%m-%d).html mydoc.adoc
```

We could write it to another folder as well by prefixing the output file with a folder name:

```
$ asciidoctor -o build/mydoc-$(date +%Y-%m-%d).html mydoc.adoc
```

If you only want to specify the output directory, but let the filename be defaulted, use the `-D` option:

```
$ asciidoctor -D build mydoc.adoc
```

The `-D` option can also be used when processing multiple input files:

```
$ asciidoctor -D build *.adoc
```

If you are piping content to the CLI, the default is to write the output to STDOUT. If you want to write the output to a file in this case, you have to specify one:

```
$ cat mydoc.adoc | asciidoctor -o build/mydoc-$(date +%Y-%m-%d).html -
```

You'll learn more about piping content through the CLI in the next section.

## 69.3. Piping Content Through the CLI

Using the `-` flag, you can pipe content to the `asciidoctor` command. This flag tells Asciidoctor read the source from standard input (STDIN). For example:

```
$ echo 'content' | asciidoctor -
```



Any variation of STDIN will work.

This command is effectively the same as:

```
$ echo 'content' | asciidoctor -o --
```

When reading source from STDIN, Asciidoctor doesn't have a reference to an input file. Therefore, it sends the converted text to standard output (STDOUT) by default.

If, instead, you want to write the full document to an output file, you specify it using the `-o` flag. For example, the following command writes a standalone HTML document to `output.html` instead of STDOUT:

```
$ echo "content" | asciidoctor -o output.html -
```

When you pipe content to the `asciidoctor` command, it no longer has a concept of where the document is located. Therefore, relative references such as includes may not work as expected. To resolve this problem, you should specify an absolute base directory using the `-B` option:

```
$ echo "content" | asciidoctor -B /path/to/basedir -o output.html -
```

Alternately, you can set an artificial document directory by passing an absolute path to the `docdir` attribute:

```
$ echo "content" | asciidoctor -a docdir=/path/to/docdir -o output.html -
```

Try both approaches to determine which one suits your needs better.

When piping source from STDIN to STDOUT through the `asciidoctor` command, you often just want the converted body (i.e., embeddable HTML). To produce that variant, add the `-s` flag:

```
$ echo 'content' | asciidoctor -s -
```

Or perhaps you want to include the doctitle as well:

```
$ echo -e '= Document Title\n\ncontent' | asciidoctor -s -a showtitle -
```

# Chapter 70. Running Asciidoctor Securely

Asciidoctor provides security levels that control the read and write access of attributes, the `include` directive, macros, and scripts while a document is processing. Each level includes the restrictions enabled in the prior security level.

## UNSAFE

A safe mode level that disables any security features enforced by Asciidoctor. Ruby is still subject to its own restrictions.

This is the default safe mode for the CLI. Its integer value is `0`.

## SAFE

This safe mode level prevents access to files which reside outside of the parent directory of the source file. The `include` directive is enabled, but paths to `include` files must be within the parent directory. This mode allows assets (such as the stylesheet) to be embedded in the document.

Its integer value is `1`.

## SERVER

A safe mode level that disallows the document from setting attributes that would affect conversion of the document. This level trims `docfile` to its relative path and prevents the document from:

- setting `source-highlighter`, `doctype`, `docinfo` and `backend`
- seeing `docdir`

It allows `icons` and `linkless`.

Its integer value is `10`.

## SECURE

A safe mode level that disallows the document from attempting to read files from the file system and including their contents into the document. Additionally, it:

- disables icons
- disables the `include` directive
- data can not be retrieved from URIs
- prevents access to stylesheets and JavaScripts
- sets the backend to `html5`
- disables `docinfo` files
- disables `data-uri`
- disables interactive (`opts=interactive`) and inline (`opts=inline`) modes for SVGs
- disables `docdir` and `docfile`
- disables source highlighting

Asciidoctor extensions may still embed content into the document depending whether they honor the safe mode setting.

This is the default safe mode for the API. Its integer value is 20.

You can set Asciidoctor's safe mode level using the CLI or API.

## 70.1. Set the Safe Mode in the CLI

When Asciidoctor is invoked via the CLI, the safe mode is set to `UNSAFE` by default. You can change the safe level by executing one of the following commands in the CLI.

`-S, --safe-mode=SAFE_MODE`

Sets the safe mode level of the document according to the assigned level (`UNSAFE`, `SAFE`, `SERVER`, `SECURE`).

`--safe, asciidoctor-safe`

Sets the safe mode level to `SAFE`. Provided for compatibility with the python AsciiDoc `safe` command.

## 70.2. Set the Safe Mode in the API

The default safe level in the API is `SECURE`.

In the API, you can set the safe mode using a string, symbol or integer value. The value must be set in the document constructor using the `:safe` option.

```
result = Asciidoctor.convert_file('master.adoc', :safe => 'server')
```

or

```
result = Asciidoctor.convert_file('master.adoc', :safe => :server)
```

or

```
result = Asciidoctor.convert_file('master.adoc', :safe => 10)
```

## 70.3. Set Attributes Based on the Safe Mode

Asciidoctor provides access to the current safe mode through built-in attributes. You can use these attributes to enable or disable content based on the current safe mode of the processor.

The safe mode can be referenced by one of the following attributes:

- The value of the `safe-mode-name` attribute (e.g., `unsafe`, `safe`, etc.)

- The value of the `safe-mode-level` attribute (e.g., 0, 10, etc.)
- The presence of the `safe-mode-<name>` attribute, where `<name>` is the safe mode name.

The attributes in the next example define replacement text for features that are disabled in high security environments:

```
ifdef::safe-mode-secure[]  
Link to chapters instead of including them.  
endif::safe-mode-secure[]
```

This feature is particularly handy for displaying content on GitHub, where the safe mode is set to its most restrictive setting, secure.

# Customizing Your Output

Asciidoctor provides a default stylesheet and built-in converters so you can quickly process and convert your document, but it also lets you use custom stylesheets and converters. The Asciidoctor project includes alternative stylesheet themes from [the stylesheet factory](#) and [specialized backends](#). You can also create your own [themes](#) and [backends](#).

# Chapter 71. Applying a Theme

A custom stylesheet can be stored in the same directory as your document or in a separate directory. Like the default stylesheet, you can have the output document link to your custom stylesheet or embed it.

If the stylesheet is in the same directory as your document, you can apply it when converting your document to HTML from the CLI.

```
$ asciidoctor -a stylesheet=mystyles.css mysample.adoc
```

1. Save your custom stylesheet in the same directory as `mysample.adoc`
2. Call the `asciidoctor` processor
3. Set `-a (--attribute)` and `stylesheet`
4. Assign the stylesheet file's name to the `stylesheet` attribute
5. Enter your document file's name.

Alternatively, let's set the `stylesheet` attribute in the header of `mysample.adoc`.

```
= My First Experience with the Dangers of Documentation
:stylesheet: mystyles.css
```

In my world, we don't have to worry about mutant, script-injecting warlocks.

No.

We have something far worse.

We're plagued by Wolpertingers.

== Origins

You may not be familiar with these <https://en.wikipedia.org/wiki/Wolpertinger>[ravenous beasts], but, trust me, they'll eat your shorts and suck the loops from your code.

## My First Experience with the Dangers of Documentation

---

In my world, we don't have to worry about mutant, script-injecting warlocks. No. We have something far worse. We're plagued by Wolpertingers.

### Origins

You may not be familiar with these [ravenous beasts](#), but, trust me, they'll eat your shorts and suck the loops from your code.

When your document and the stylesheet are stored in different directories, you need to tell Asciidoctor where to look for the stylesheet in relation to your document. Asciidoctor uses the relative or absolute path you assign to the `stylesdir` attribute to find the stylesheet. Let's move `mystyles.css` into `mydocuments/mystylesheets/`. Our AsciiDoc document, `mysample.adoc`, is saved in the `mydocuments/` directory.

```
= My First Experience with the Dangers of Documentation
```

```
:stylesdir: mystylesheets/
```

```
:stylesheet: mystyles.css
```

In my world, we don't have to worry about mutant, script-injecting warlocks.

No.

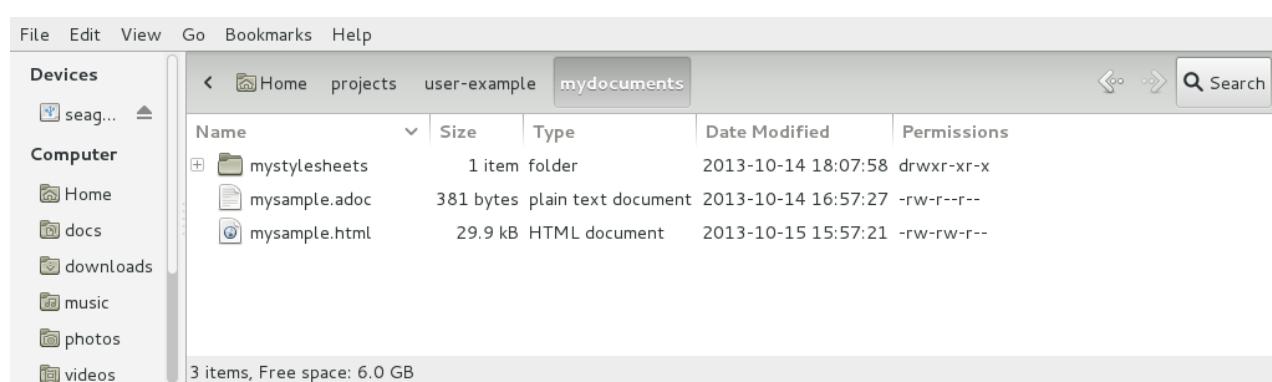
We have something far worse.

We're plagued by Wolpertingers.

```
== Origins
```

You may not be familiar with these <https://en.wikipedia.org/wiki/Wolpertinger>[ravenous beasts], but, trust me, they'll eat your shorts and suck the loops from your code.

After processing `mysample.adoc`, its HTML output (`mysample.html`), which includes the embedded `mystyles.css` stylesheet, is created in the `mydocuments/` directory.



You can also set `stylesdir` in the CLI.

```
$ asciidoctor -a stylesdir=mystylesheets/ -a stylesheet=mystyles.css mysample.adoc
```

If you don't want to embed the `mystyles.css` stylesheet into your HTML output, make sure to set `linkcss`.

= My First Experience with the Dangers of Documentation

:stylesdir: mystylesheets/

:stylesheet: mystyles.css

:linkcss:

In my world, we don't have to worry about mutant, script-injecting warlocks.

No.

We have something far worse.

We're plagued by Wolpertingers.

## == Origins

You may not be familiar with these <https://en.wikipedia.org/wiki/Wolpertinger>[ravenous beasts], but, trust me, they'll eat your shorts and suck the loops from your code.

After your document is converted, notice that a copy of `mystyles.css` was not created in the `mydocuments/` directory. Unlike when you link to the default Asciidoctor stylesheet, any custom stylesheets you link to are not copied to the directory where your output is saved.

## Stylesheets and processing multiple nested documents

When you are [running Asciidoc once across a nested set of documents](#), it's currently not possible to specify a single relative path for the `stylesdir` attribute that would work for all of the documents. This is because the relative depth of the stylesheet's location differs for the documents in the subdirectories. One way to solve this problem is to maintain the path to `stylesdir` in each document.

Let's say you have three AsciiDoc documents saved in the following directory structure:

```
/mydocuments
  a.adoc
  b.adoc
/mynesteddocuments
  c.adoc
/mystylesheets
```

For `a.adoc` and `b.adoc`, set `stylesdir` to:

```
:stylesdir: mystylesheets
```

For `c.adoc`, set `stylesdir` to:

```
:stylesdir: ../mystylesheets
```

If you're serving your documents from a webserver, you can solve this problem by providing an absolute path to the stylesheet.

# Chapter 72. Stylesheet Factory

The [Asciidoctor stylesheet factory](#) is where themes are developed for styling your documents. Specifically, these stylesheets can be used to quickly customize the look and feel of HTML 5 documents generated by Asciidoctor.

To view the Asciidoctor themes in action, visit the [theme showcase](#).



The Asciidoctor stylesheet factory is only compatible with Asciidoctor 0.1.2 and greater.

## 72.1. Setting up the Factory

The stylesheets in the Asciidoctor stylesheet factory are built using [Compass](#), a CSS authoring framework that uses [Sass](#) to generate CSS files. The styles and components are generated by [Foundation 4](#), an awesome and flexible CSS component framework that ensures your stylesheet is cross-browser and mobile friendly.

### 72.1.1. Install the Gems

In order to build the stylesheets, you must download the Asciidoctor stylesheet factory repository and install the Compass and Foundation gems.

1. Download or clone the [Asciidoctor stylesheet factory repository](#).



It does not matter where you save the project on your system.

2. Make sure you have [Ruby and RubyGems](#) installed, and, ideally, [Bundler](#).
3. Run Bundler to install the Compass and Foundation gems.

```
$ bundle install
```

The previous `bundle` command is equivalent to the following two commands:



```
$ gem install compass --version 0.12.2  
$ gem install zurb-foundation --version 4.3.2
```

You don't need to execute these `gem install` commands if you use Bundler.

Once you have the gems installed, you can build the stylesheets.

### 72.1.2. Build the Stylesheets

To build the stylesheets:

1. Navigate to the Asciidoctor stylesheet factory directory on your system.
2. Run Compass's `compile` command.

```
$ compass compile
```

The stylesheets are compiled from the Sass source files in the `sass/` folder and written to the `stylesheets/` folder. You can reference the stylesheets in `stylesheets/` from your HTML file.

## 72.2. Creating a Theme

You can create your own themes to apply to your documents.

Themes go in the `sass/` folder. To create a new theme, let's call it `hipster`, start by creating two new files:

### `sass/hipster.scss`

- Imports the theme settings, which includes default variables and resets
- Imports the AsciiDoc components
- Defines any explicit customization

### `sass/settings/_hipster.scss`

- Sets variables that customize Foundation 4 and the AsciiDoc CSS components

Here's a minimal version of `sass/hipster.scss`:

```
@import "settings/hipster";  
@import "components/asciidoc";  
@import "components/awesome-icons";
```



You don't have to include the underscore prefix when importing files.



The `awesome-icons` component is only applicable to HTML generated by Asciidoctor > 0.1.2 with the `icons` attribute set to `font`.

You can add any explicit customization below the import lines.

The variables you can set in `sass/settings/_hipster.scss` are a combination of the [Foundation 4 built-in global settings](#) and [global settings and imports for the AsciiDoc components](#).

Once you've created your custom theme, it's time to apply it to your document.

## 72.3. Applying a Stylesheet

Let's practice applying a stylesheet to a simple AsciiDoc file.

1. Create an AsciiDoc file like the one below.
2. Save the file as *mysample.adoc*.

```
= Introduction to AsciiDoc
Doc Writer <doc@example.com>

A preface about http://asciidoc.org[AsciiDoc].

== First Section

* item 1
* item 2

[source,ruby]
puts "Hello, World!"
```

Next, you'll use the Asciidoctor processor to generate HTML and apply a stylesheet to it from the *stylesheets/* directory.

## 72.4. Generate an HTML Document

Now it's time to pick the stylesheet you want to apply to your content when it is rendered. In your file browser, navigate to the *stylesheets/* directory. Or, using a console, change to the *stylesheets/* directory and list the available stylesheets using the `ls` command.

```
$ ls
```

*Console output of ls command*

```
asciidoctor.css  foundation-lime.css    iconic.css      riak.css
colony.css       foundation-potion.css maker.css       rubygems.css
foundation.css   golo.css           readthedocs.css
```

Let's apply the *colony.css* stylesheet to the sample document.

1. Navigate to the directory where you saved *mysample.adoc*.
2. Call the `asciidoctor` processor.
3. Specify the stylesheet you want applied with the `stylesheet` attribute.
4. Tell the processor where the specified stylesheet is located with the `stylesdir` attribute.

```
$ asciidoctor -a stylesheet=colony.css -a stylesdir=../stylesheets mysample.adoc
```

Open a browser, navigate to *mysample.html* and checkout the result! If you inspect the *mysample.html* document, you should see that the stylesheet is embedded in the converted

document.

## Stylesheet images

The Golo, Maker, and Riak themes include image assets. To apply these themes to your document correctly, the applicable images must be copied into the same directory as the generated output.

For example, to apply the *maker.css* stylesheet to *mysample.adoc*:

1. Copy *body-bh.png* from the *images/maker/* directory into the output directory.
2. Call the **stylesheet** and **styledir** attributes.

```
$ asciidoctor -a stylesheet=maker.css -a stylesdir=../stylesheets  
mysample.adoc
```

Navigate to *mysample.html* in your browser. The *body-bh.png* image should add a graph paper-like background to your generated output.

## 72.5. External Preview

You may want to preview sample HTML files on another computer or device. To do that, you need to serve them through a web server. You can quickly serve HTML files in the root directory of the project using the following command:

*Using Python*

```
$ python -m SimpleHTTPServer 4242
```

or

*Using Ruby >= 1.9.3*

```
$ ruby -run -e httpd . -p 4242
```

# Chapter 73. Slide Decks

In addition to generating normal documents from AsciiDoc (e.g., a book or article), you can also use AsciiDoc to create HTML-based slide decks. The converters for creating HTML-based slides add some additional structure rules to the document for defining slides. However, nothing in this structure restricts you from still being able to publish the content as a normal document too.

The conversion from AsciiDoc to an HTML-based slide deck is handled by one of the following converters from the Asciidoctor ecosystem:

- [Asciidoctor Reveal.js](#) — Converts an AsciiDoc document into a Reveal.js presentation
- [Asciidoctor Bespoke](#) — Converts an AsciiDoc document into a Bespoke presentation

There's also a converter for making deck.js slide decks, but we discourage using it as deck.js itself is very outdated.

# Chapter 74. Custom Backends

## 74.1. Storing Multiple Templates

Custom templates can be stored in multiple directories. That means you can build on an existing backend by copying only the templates you want to modify. Then, just pass both the directory holding the original templates and the directory containing your customized templates when you invoke Asciidoctor.

In the CLI, multiple template directories are specified by using the `-T` option multiple times.

```
$ asciidoctor -T /path/to/original/templates -T /path/to/modified/templates  
mysample.adoc
```

In the API, multiple template directories are specified by passing an array to the `template_dirs` option:

```
Asciidoctor.convert_file 'mysample.adoc', safe: :safe  
  template_dirs: %w(/path/to/original/templates /path/to/modified/templates)
```

# Chapter 75. Using Asciidoctor with Other Languages

The built-in labels, messages, and syntax keywords in Asciidoctor are English by default. However, Asciidoctor is not restricted to working with English-only content. Asciidoctor can process the full range of the UTF-8 character set. That means you can write your document in any language, save the file with UTF-8 encoding, and expect Asciidoctor to convert the text properly. Furthermore, you can customize the built-in labels (e.g., “Appendix”) to match the language in which you are writing.

There are some caveats to know about:

- Currently, the official HTML and PDF converters only fully support left-to-right (and top-to-bottom) reading. Support for right-to-left (RTL) is being worked on. See [issue #1601](#) for details. In the interim, you can leverage the DocBook toolchain to get right-to-left support.
- Attributes that store dates and times (e.g., `docdatetime`) are always formatted like `2019-01-04 19:26:06 GMT+0000`.
- Message (aka logging) strings are always in English.
- Asciidoctor does not support the language conf files used by AsciiDoc Python. However, Asciidoctor does provide [a translation file](#) that can be used for a similar purpose.

## 75.1. Translating built-in labels

When converting to DocBook, you can rely on the DocBook toolchain to translate (most) built-in labels. To activate this feature, simply set the `lang` attribute to a valid country code (which defaults to `en` for English). For example:

```
$ asciidoctor -a lang=es -b docbook article.adoc
```

The list of supported languages, as well as additional language considerations for DocBook, are described in [DocBook XSL: The Complete Guide](#).

The `lang` attribute *does not* enable automatic translation of built-in labels when converting directly to HTML or PDF. It’s merely a hint to configure the DocBook toolchain. If you’re not using the DocBook toolchain for publishing, you must translate each built-in label yourself. One way is to set the following attributes in the document header or by passing the attributes via the API or CLI:

*Attributes that control built-in labels*

Attribute name	Used for	Default
<code>appendix-caption</code>	Appendix titles.	Appendix
<code>caution-caption</code>	CAUTION admonitions (when icons are not in use).	Caution

Attribute name	Used for	Default
chapter-label	Prefix added to chapter titles (i.e., level-1 section titles when doctype is book). ( <i>pdf converter only</i> )	Chapter
example-caption	Example titles.	Example
figure-caption	Automatically prefixed to figure titles.	Figure
important-caption	IMPORTANT admonitions (when icons are not in use).	Important
last-update-label	Label for when the document was last updated.	Last updated
listing-caption	The label for listing blocks. By default, listing blocks do not have captions. If you specify <code>listing-caption</code> , then you also turn on captions for listing blocks.	<i>not set</i>
manname-title	Label for the program name section in the man page.	NAME
note-caption	NOTE admonitions (when icons are not in use).	Note
preface-title	Title text used for the anonymous preface (when the <code>doctype</code> is book).	<i>not set</i>
table-caption	Automatically prefixed to table titles.	Table
tip-caption	TIP admonitions (when icons are not in use).	Tip
toc-title	Title of the table of contents.	Table of Contents
untitled-label	The document title, for documents that have only body content.	Untitled
version-label	The label preceding the revnumber in the document's byline.	Version
warning-caption	WARNING admonitions (when icons are not in use).	Warning

If you plan to support multiple languages, you'll want to define the attributes for each language inside a conditional preprocessor directive. For example:

```
ifeval::["{lang}" == "de"]
:caution-caption: Achtung
...
endif::[]
```

Of course, you're probably hoping this has already been done for you. Indeed, it has!

You can find an [AsciiDoc file](#) in the Asciidoctor repository that provides translations of these attributes for most major languages. The translations are defined using AsciiDoc attribute entries inside conditional preprocessor blocks, just as suggested above.

To use this file to translate the built-in labels according the the value of the `lang` attribute (just like the DocBook toolchain does), follow these steps:

1. Download the AsciiDoc file [attributes.adoc](#) from the Asciidoctor repository.
2. Put the file in the folder `locale` relative to your document.
3. Add the following line to the header of your AsciiDoc document:

```
include::locale/attributes.adoc[]
```

4. Set the language using the `lang` attribute. This attribute must be set before the include directive gets processed. For example:

```
-a lang=es
```

The built-in labels will now be translated automatically based on the value of the `lang` attribute.

There's an ongoing discussion about how to make language support even simpler ([issue #1129](#)). Input is welcome.

## 75.2. Translation

Asciidoctor (or DocBook) currently does not support translation of content out of the box. There's a proposal to integrate gettext ([discussion](#)), but suggestions are welcome.

# Publishing Your Content

# Chapter 76. Static Website Generators

## 76.1. Front Matter Added for Static Site Generators

Many static site generators (i.e., Jekyll, Middleman, Awestruct) rely on "front matter" added to the top of the document to determine how to convert the content. Front matter typically starts on the first line of a file and is bounded by block delimiters (e.g., `---`).

Here's an example of a document that contains front matter:

```
--- ①
layout: default ②
--- ①
= Document Title

content
```

① Front matter delimiters

② Front matter data

The static site generator removes these lines before passing the document to the AsciiDoc processor to be converted. Outside of the tool, however, these extra lines can throw off the processor.

If the `skip-front-matter` attribute is set via the CLI or API (e.g., `-a skip-front-matter`), Asciidoctor (as of 0.1.4) will recognize the front matter and consume it before parsing the document. Asciidoctor stores the content it removes in the `front-matter` attribute to make it available for integrations. Asciidoctor also removes front matter when reading [include files](#).



Awestruct can read front matter directly from AsciiDoc attributes defined in the document header, thus eliminating the need for this feature.

### 76.1.1. Configuring Attributes for Awestruct

Awestruct defines a set of default attributes that it passes to the API in its `/default-site.yml` file. One of the attributes in that configuration is `imagesdir`. The value there is set to `/images`. That means the value in your document is skipped due to the precedence rules.

Fortunately, there is one additional place you can override the attribute. This gives you the opportunity to set your own default and to flip the precedence order so that the document wins out. If an attribute value that is passed to the API ends with an `@` symbol, it makes that assignment have a lower precedence than an assignment in the document.

You can define attributes you want to pass to the API in the `_config/site.yml` file. Here's an example entry for Asciidoctor:

```
ascidioctor:  
:safe: safe  
:attributes:  
  imagesdir: /assets/images@  
  icons: font  
...
```



The second-level keys (safe and attributes, in this case) must have colons on both sides of the key name. The rest of the keys only have a colon after the key.

With this configuration added, you should observe that the `imagesdir` attribute in your document is now respected.

# Using Asciidoctor's API

In addition to the command line interface, Asciidoctor provides a Ruby API. The API is intended for integration with other software projects and is suitable for server-side applications, such as Rails, Sinatra and GitHub.

Asciidoctor also has a Java API that mirrors the Ruby API. The Java API calls through to the Ruby API using an embedded JRuby runtime. See the [AsciidoctorJ project](#) for more information.

# Chapter 77. Require the Library

To use Asciidoctor in your application, you first need to require the gem:

```
require 'asciidoctor'
```

This statement makes all of the [public APIs in Asciidoctor](#) available to your script or application. You are now ready to start processing AsciiDoc documents.

The main entry points in the Asciidoctor API are the static methods to load or convert AsciiDoc documents, which we'll cover the next two chapters.

# Chapter 78. Load and Convert Files

To parse a file into an `Asciidoctor::Document` object:

```
doc = Asciidoctor.load_file 'mysample.adoc'
```

You can get information about the document:

```
puts doc doctitle  
puts doc.attributes
```

More than likely, you will want to convert the document. To convert a file containing AsciiDoc markup to HTML 5, use:

```
Asciidoctor.convert_file 'mysample.adoc'
```

The command will output to the file *mysample.html* in the same directory.

You can convert the file to DocBook 5.0 by setting the `:backend` option to '`docbook`':

```
Asciidoctor.convert_file 'mysample.adoc', backend: 'docbook'
```

The command will output to the file *mysample.xml* in the same directory. If you're on Linux, you can view the file using [Yelp](#).

# Chapter 79. Load and Convert Strings

To parse an AsciiDoc-formatted string into a document object model, use:

```
doc = Asciidoctor.load '*This* is Asciidoctor.'
```

To convert the AsciiDoc-formatted string instead, use:

```
puts Asciidoctor.convert '*This* is Asciidoctor.'
```

When converting a string, Asciidoctor does not output a standalone document by default. Instead, it generates embeddable output. Let's learn why that is and how to control it.

## 79.1. Embeddable output

When you pass a string to `Asciidoctor.convert` to convert it to a backend format, such as HTML, this method only returns the converted content for those blocks. It does not include the shell around that content (i.e., the header and footer) that is needed to make a standalone document. Instead, it makes an *embeddable* document. This default was chosen to make Asciidoctor consistent with other lightweight markup processors like Markdown.

Here's what's included in an embeddable document:

- The document title, but only if the `showtitle` attribute is set (no attribution and revision information)
- The table of contents if the `toc` attribute is enabled (and not macro or preamble)
- The converted document body
- The footnotes unless the `nofootnotes` attribute is set

You can still generate a standalone document when converting a string. To go from an AsciiDoc string to a standalone output document, you need to explicitly set the `:header_footer` option to `true`.

```
puts Asciidoctor.convert '*This* is Asciidoctor.', header_footer: true
```

Now you'll get back a full HTML 5 file.

You can get the same behavior when converting a file by setting the `:header_footer` option to `false`. However, most of the time you'll want to generate a standalone document when converting a file (which is the default).

Another way to implicitly enable the shell is to output the result to a file. In this case, the default output behavior changes from embeddable to standalone.

```
puts Asciidoctor.convert '*This* is Asciidoctor.', to_file: 'out.html'
```

When converting a string, the TOC is only included by default when using the `:header_footer` option as shown above. However, you can force it to be included without the header and footer by setting the `toc` attribute with a value of `macro` and using the `toc::[]` macro in the string itself.

## 79.2. Convert inline markup only

If you only want the inline markup to be returned, set the `:doctype` option to '`inline`':

```
puts Asciidoctor.convert '*This* is Asciidoctor.', :doctype => 'inline'
```

In this mode, Asciidoctor will only process the first block (e.g., paragraph) in the document and ignore the rest.

## 79.3. Convert to DocBook

You can produce DocBook 5.0 by setting the `backend` option to '`docbook`'. Since embeddable DocBook isn't that useful, we also enable the shell (i.e., header and footer) by setting the `:header_footer` option to `true`.

```
puts Asciidoctor.convert '*This* is Asciidoctor.', :header_footer => true,  
:backend => 'docbook'
```

Let's say you don't like some (or all) of the output that Asciidoctor produces. That's no problem. You can change it. And yes, any of it! Let's find out how to customize the output Asciidoctor spits out in the next chapter.

# Chapter 80. Generate an HTML TOC

Asciidoctor's HTML 5 converter has built-in support for generating a TOC. This TOC generator can also be used as a general purpose API. This logic is available via the `outline` method on the HTML5 converter.

The `outline` method accepts a Document object and an optional Hash of options, and returns an HTML string. It can be resolved and invoked from anywhere using the following:

```
html_toc = (Asciidoctor::Converter.for 'html5').outline document
```

The method can also be executed from inside a converter template (e.g., Slim, Haml, or ERB). When using a composite converter, this call will be run through the converter chain.

```
= converter.convert document, 'outline'
```

The method is also available through the Document API:

```
html_doc = document.converter.convert document, 'outline'
```

The `outline` method accepts the following options:

## **sectnumlevels**

the number of section levels to number (defaults to the value of the `sectnumlevels` attribute).

## **toclevels**

the depth of the toc (defaults to the value of the `toclevels` attribute)

# Chapter 81. Provide Custom Templates

Asciidoctor allows you to override the converter methods used to convert almost any individual AsciiDoc element. If you provide a directory of Tilt-compatible templates, named in such a way that Asciidoctor can figure out which template goes with which element, Asciidoctor will use the templates in this directory instead of its built-in templates for any elements for which it finds a matching template. It will fallback to its default templates for everything else.

```
puts Asciidoctor.convert '*This* is Asciidoctor.', :header_footer => true,  
:template_dir => 'templates'
```

The Document and Section templates should begin with `document` and `section`, respectively. The file extension is used by Tilt to determine which view framework it will use to use to interpret the template. For instance, if you want to write the template in ERB, you'd name these two templates `document.html.erb` and `section.html.erb`. The first file extension, `.html`, maps to the converter and the second, `.erb` maps to the template engine. To use Haml as the template engine, you'd name the templates `document.html.haml` and `section.html.haml`.

Templates for block elements, like a paragraph or sidebar, are named after the block they handle. For instance, to override the default paragraph template with an ERB template, put a file named `paragraph.html.erb` in the template directory you pass to the `Document` constructor using the `:template_dir` option.

For more usage examples, see the [asciidoctor-backends](#), the O'Reilly [htmlbook](#) project and the (massive) [test suite](#).

# Extensions

Extensions have proven to be central to the success of AsciiDoc because they open up the language to new use cases. However, the way extensions are implemented in AsciiDoc Python presents a number of problems:

- They are challenging to write because they work at such a low-level (read as: nasty regular expressions).
- They are fragile since they often rely on system commands to do anything significant.
- They are hard to distribute due to the lack of integration with a formal distribution system.

Asciidoctor addresses these issues by introducing a proper extension API that offers a superset of the extension points that AsciiDoc Python provides. As a result, extensions in Asciidoctor are easier to write, more powerful, and simpler to distribute.

The goal for Asciidoctor has always been to allow extensions to be written using the full power of a programming language (whether it be Ruby, Java, Groovy or JavaScript), similar to what we've done with the backend (conversion) mechanism. That way, you don't have to shave yaks to get the functionality you want, and you can distribute the extension using defacto-standard packaging mechanisms like RubyGems or JARs.



The extension API in Asciidoctor is stable with the exception of inline macros. Since inline content is not parsed until the convert phase, the inline macro processor must return converted text (e.g., HTML) rather than an AST node. Once Asciidoctor is changed to [process inline content during the parse phase](#), the inline macro processor will need to return an inline node. When that switch occurs, there will either be some sort of adapter or required migration for inline macro processors, but that has yet to be determined.

# Chapter 82. Extension Points

Here are the extension points that are available in Asciidoctor 0.1.4.

## Preprocessor

Processes the raw source lines before they are passed to the parser.

## TreeProcessor

Processes the Asciidoctor::Document (AST) once parsing is complete.

## Postprocessor

Processes the output after the document has been converted, but before it's written to disk.

## DocinfoProcessor

Adds additional content to the header or footer regions of the generated document.

## Block processor

Processes a block of content marked with a custom block style (i.e., `[custom]`). (similar to an AsciiDoc filter)

## Block macro processor

Registers a custom block macro and processes it (e.g., `gist::12345[]`).

## Inline macro processor

Registers a custom inline macro and processes it (e.g., `[:Save:]`).

## Include processor

Processes the `include::<filename>[]` directive.

These extensions are registered per document using a callback that feels sort of like a DSL:

```
Asciidoctor::Extensions.register do |document|
  preprocessor FrontMatterPreprocessor
  tree_processor ShellSessionTreeProcessor
  postprocessor CopyrightFooterPostprocessor
  docinfo_processor TrackingCodeDocinfoProcessor if document.basebackend? 'html'
  block ShoutBlock
  block_macro GistBlockMacro if document.basebackend? 'html'
  inline_macro ManInlineMacro
  include_processor UriIncludeProcessor
end
```



Extension classes must be defined outside of the register block. Once an extension class is registered, it is frozen, preventing further modification. If you define an extension class inside the register block, it will result in an error on subsequent invocations.

You can register more than one processor of each type, though you can only have one processor per custom block or macro. Each registered class is instantiated when the Asciidoctor::Document is created.



There is currently no extension point for processing a built-in block, such as a normal paragraph. Look for that feature in a future Asciidoctor release.

# Chapter 83. Example Extensions

Below are several examples of extensions and how they are registered.

## 83.1. Preprocessor Example

### Purpose

Skim off front matter from the top of the document that gets used by site generators like Jekyll and Awestruct.

*sample-with-front-matter.adoc*

```
---
tags: [announcement, website]
---
= Document Title

content

[subs="attributes,specialcharacters"]
.Captured front matter
....
---
{front-matter}
---
....
```

## *FrontMatterPreprocessor*

```
require 'asciidoc'
require 'asciidoc/extensions'

class FrontMatterPreprocessor < Asciidoctor::Extensions::Preprocessor
  def process document, reader
    lines = reader.lines # get raw lines
    return reader if lines.empty?
    front_matter = []
    if lines.first.chomp == '---'
      original_lines = lines.dup
      lines.shift
      while !lines.empty? && lines.first.chomp != '---'
        front_matter << lines.shift
      end

      if (first = lines.first).nil? || first.chomp != '---'
        lines = original_lines
      else
        lines.shift
        document.attributes['front-matter'] = front_matter.join.chomp
        # advance the reader by the number of lines taken
        (front_matter.length + 2).times { reader.advance }
      end
    end
    reader
  end
end
```

## *Usage*

```
Asciidoctor::Extensions.register do
  preprocessor FrontMatterPreprocessor
end

Asciidoctor.convert_file 'sample-with-front-matter.adoc', :safe => :safe
```

## 83.2. Tree Processor Example

### Purpose

Detect literal blocks that contain shell commands, strip the prompt character and style the command using CSS in such a way that the prompt character cannot be selected (as seen on help.github.com).

*sample-with-shell-session.adoc*

```
$ echo "Hello, World!"  
> Hello, World!  
  
$ gem install asciidoctor
```

*ShellSessionTreeProcessor*

```
class ShellSessionTreeProcessor < Asciidoctor::Extensions::TreeProcessor  
  def process document  
    return unless document.blocks?  
    process_blocks document  
    nil  
  end  
  
  def process_blocks node  
    node.blocks.each_with_index do |block, i|  
      if block.context == :literal &&  
        ((first_line = block.lines.first).start_with? '$ ') ||  
        (first_line.start_with? '> ')  
        node.blocks[i] = convert_to_terminal_listing block  
      else  
        process_blocks block if block.blocks?  
      end  
    end  
  end  
  
  def convert_to_terminal_listing block  
    attrs = block.attributes  
    attrs['role'] = 'terminal'  
    prompt_attr = (attrs.has_key? 'prompt') ?  
      %( data-prompt="#{block.sub_specialchars attrs['prompt']}") : nil  
    lines = block.lines.map do |line|  
      line = block.sub_specialchars line.chomp  
      if line.start_with? '$ '  
        %(<span class="command"#{prompt_attr}>#{line[2..-1]}</span>)  
      elsif line.start_with? '> '  
        %(<span class="output">#{line[5..-1]}</span>)  
      else  
        line  
      end  
    end  
    create_listing_block block.document, lines * EOL, attrs, subs: nil  
  end  
end
```

## Usage

```
Asciidoctor::Extensions.register do
  treeprocessor ShellSessionTreeProcessor
end

Asciidoctor.convert_file 'sample-with-shell-session.adoc', :safe => :safe
```

## 83.3. Postprocessor Example

### Purpose

Insert copyright text in the footer.

### *CopyrightFooterPostprocessor*

```
class CopyrightFooterPostprocessor < Asciidoctor::Extensions::Postprocessor
  def process document, output
    content = (document.attr 'copyright') || 'Copyright Acme, Inc.'
    if document.basebackend? 'html'
      replacement = %(<div id="footer-text">\n<br>\n#{content}\n</div>)
      output = output.sub(/<div id="footer-text">(.*?)</div>/m, replacement)
    elsif document.basebackend? 'docbook'
      replacement = %(<simpara>#{content}</simpara>\n\\1)
      output = output.sub(/(<\\/(?:article|book)>)/, replacement)
    end
    output
  end
end
```

## Usage

```
Asciidoctor::Extensions.register do
  postprocessor CopyrightFooterPostprocessor
end

Asciidoctor.convert_file 'sample-with-copyright-footer.adoc', :safe => :safe
```

## 83.4. Docinfo Processor Example

### Purpose

Appends the Google Analytics tracking code to the bottom of an HTML document.

## *GoogleAnalyticsDocinfoProcessor*

```
class GoogleAnalyticsDocinfoProcessor < Asciidoctor::Extensions::DocinfoProcessor
  use_dsl
  at_location :footer
  def process document
    return unless (ga_account_id = document.attr 'google-analytics-account')
    %(<script>
(function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
(i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.createElement(o),
m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m)
})(window,document,'script','https://www.google-analytics.com/analytics.js','ga');
ga('create', '#{ga_account_id}', 'auto');
ga('send', 'pageview');
</script>
    end
  end
```

## *Usage*

```
Asciidoctor::Extensions.register do
  docinfo_processor GoogleAnalyticsDocinfoProcessor
end

Asciidoctor.convert_file 'sample.adoc', :safe => :safe,
                        :attributes => 'UA-ABCXYZ123'
```

## 83.5. Block Processor Example

### Purpose

Register a custom block style named **shout** that uppercases all the words and converts periods to exclamation points.

*sample-with-shout-block.adoc*

```
[shout]
The time is now. Get a move on.
```

## *ShoutBlock*

```
require 'asciidoc'
require 'asciidoc/extensions'

class ShoutBlock < Asciidoctor::Extensions::BlockProcessor
  PeriodRx = /\.(?= |$)/

  use_dsl

  named :shout
  on_context :paragraph
  name_positional_attributes 'vol'
  parse_content_as :simple

  def process parent, reader, attrs
    volume = ((attrs.delete 'vol') || 1).to_i
    create_paragraph parent, (reader.lines.map {|l| l.upcase.gsub PeriodRx, '!' * volume }), attrs
  end
end
```

## *Usage*

```
Asciidoctor::Extensions.register do
  block ShoutBlock
end

Asciidoctor.convert_file 'sample-with-shout-block.adoc', :safe => :safe
```

## 83.6. Compound Block Processor Example

### Purpose

Register a custom block named `collapsiblelisting` that transforms a listing block into a compound block composed of the following:

- an example block with the collapsible option enabled
- the original listing block
- the listing block is promoted to a source block if a language is specified using the second positional attribute.

```
.Show JSON  
[collapsiblelisting.json]  
----  
{  
  "foo": "bar"  
}  
----
```

### *CollapsibleListingBlock*

```
class CollapsibleListingBlock < Asciidoctor::Extensions::BlockProcessor  
  enable_dsl  
  on_context :listing  
  positional_attributes 'language'  
  
  def process parent, reader, attrs  
    lang = attrs.delete 'language'  
    attrs['title'] ||= 'Show Listing'  
    example = create_example_block parent, [], attrs, content_model: :compound  
    example.set_option 'collapsible'  
    listing = create_listing_block example, reader.readlines, nil  
    if lang  
      listing.style = 'source'  
      listing.set_attr 'language', lang  
      listing.commit_subs  
    end  
    example << listing  
    example  
  end  
end  
  
Asciidoctor::Extensions.register do  
  block CollapsibleListingBlock, :collapsiblelisting  
end
```

### *Usage*

```
$ asciidoctor -r ./collapsiblelisting.rb sample-with-collapsiblelisting-block.adoc
```



This extension mimics the builtin `collapsible` option on the example block, but consolidates it to a single block. The purpose of this extension is to show how to assemble a compound block in an extenesion.

## 83.7. Block Macro Processor Example

## Purpose

Create a block macro named `gist` for embedding a gist.

*sample-with-gist-macro.adoc*

```
.My Gist  
gist::123456[]
```

*GistBlockMacro*

```
require 'asciidoctor'  
require 'asciidoctor/extensions'  
  
class GistBlockMacro < Asciidoctor::Extensions::BlockMacroProcessor  
  use_dsl  
  
  named :gist  
  
  def process parent, target, attrs  
    title_html = (attrs.has_key? 'title') ?  
      %(<div class="title">#{attrs['title']}</div>\n) : nil  
  
    html = %(<div class="openblock gist">  
      #{title_html}<div class="content">  
        <script src="https://gist.github.com/#{@target}.js"></script>  
      </div>  
    </div>)  
  
    create_pass_block parent, html, attrs, subs: nil  
  end  
end
```

*Usage*

```
Asciidoctor::Extensions.register do  
  block_macro GistBlockMacro if document.basebackend? 'html'  
end  
  
Asciidoctor.convert_file 'sample-with-gist.adoc', :safe => :safe
```

## 83.8. Inline Macro Processor Example

### Purpose

Create an inline macro named `man` that links to a man page.

See `man:gittutorial[7]` to get started.

### *ManpageInlineMacro*

```
require 'asciidoc'
require 'asciidoc/extensions'

class ManInlineMacro < Asciidoctor::Extensions::InlineMacroProcessor
  use_dsl

  named :man
  name_positional_attributes 'volnum'

  def process parent, target, attrs
    text = manname = target
    suffix = ''
    target = %{#{manname}.html}
    suffix = if (volnum = attrs['volnum'])
      "#{volnum}"
    else
      nil
    end
    parent.document.register :links, target
    %{#{(create_anchor parent, text, type: :link, target: target).convert}#{suffix}}
  end
end
```

### *Usage*

```
Asciidoctor::Extensions.register do
  inline_macro ManInlineMacro
end

Asciidoctor.convert_file 'sample-with-man-link.adoc', :safe => :safe
```

## 83.9. Include Processor Example

### Purpose

Include a file from a URI.



Asciidoctor supports including content from a URI out of the box if you set the `allow-uri-read` attribute (not available if the safe mode is `secure`).

### *sample-with-uri-include.adoc*

```
:source-highlighter: coderay

.Gemfile
[source,ruby]
-----
include::https://raw.githubusercontent.com/asciidoc/asciidoc/master/Gemfile[]
-----
```

### *UriIncludeProcessor*

```
require 'asciidoc'
require 'asciidoc/extensions'
require 'open-uri'

class UriIncludeProcessor < Asciidoc::Extensions::IncludeProcessor
  def handles? target
    (target.start_with? 'http://') or (target.start_with? 'https://')
  end

  def process doc, reader, target, attributes
    content = (open target).readlines
    reader.push_include content, target, target, 1, attributes
    reader
  end
end
```

### *Usage*

```
Asciidoc::Extensions.register do
  include_processor UriIncludeProcessor
end

Asciidoc.convert_file 'sample-with-uri-include.adoc', :safe => :safe
```

You can see plenty more extension examples in the [extensions lab](#).

# **Build Integrations and Implementations**

# Chapter 84. Java

[AsciidoctorJ](#) is the official library for running Asciidoctor on the JVM. Using AsciidoctorJ, you can convert AsciiDoc content or analyze the structure of a parsed AsciiDoc document from Java and other JVM languages. For details see [Getting Started with Asciidoctor in Java using AsciidoctorJ](#).

# Chapter 85. Gradle

The [Asciidoctor Gradle Plugin](#) is the official means of using Asciidoctor to convert all your AsciiDoc documentation using Gradle. For details see [Getting Started with the Asciidoctor Gradle Plugin](#).

# Chapter 86. Maven

The [Asciidoctor Maven Plugin](#) is the official way to convert your AsciiDoc documentation using Asciidoctor from an Apache Maven build. For details see [Installing and Using the Asciidoctor Maven Plugin](#).

# Chapter 87. Apache Ant

The `asciidoc-ant` task is an all-in-one solution for running Asciidoctor from Ant. It is based on AsciidoctorJ and JRuby, both of which are encapsulated in a single jar file.

Usage is:

```
<project xmlns:asciidoc="antlib:org.asciidoctor.ant">
...
<target name="doc">
    <taskdef uri="antlib:org.asciidoctor.ant"
        resource="org/asciidoc/ant/antlib.xml"
        classpath="lib/asciidoc-ant.jar"/>
    <asciidoc:convert sourceDirectory="src/asciidoc" outputDirectory="target"/>
</target>
...
</project>
```

For more details, see [asciidoc-ant on GitHub](#).

# Chapter 88. JavaDoc

Using the [Asciidoctor doclet](#), you can write your Javadoc comments using all the features of Asciidoctor including tables, lists, code examples with syntax highlighting, and pictures.

If you want to incorporate your Javadoc into another document instead, there is an experimental doclet that exports them to AsciiDoc files. See the [exportdoclet project](#) and the [related discussion thread](#) for details.

# Chapter 89. JavaScript

[Asciidoctor.js](#) is a JavaScript port of Asciidoctor, transpiled directly from the Ruby version. Asciidoctor.js brings the AsciiDoc syntax to the browser and the Node.js ecosystem. For details see [Installing and Using Asciidoctor.js](#).

# Conversions and Migrations

In this part, you'll learn how to migrate documents written in other documentation languages to AsciiDoc. We'll start with the simplest migration, switching from AsciiDoc Python, the original implementation of AsciiDoc, to Asciidoctor, the modern implementation. We'll then cover how to move to AsciiDoc from other documentation languages like DocBook, Markdown and HTML. In addition to differences in the syntax, many of the sections also suggest tools you can use to ease the migration.

# Chapter 90. Migrating from AsciiDoc Python

The purpose of this section is to help you migrate legacy AsciiDoc documents written for AsciiDoc Python to the modern AsciiDoc syntax supported by Asciidocor and to learn about equivalent tools and extensions.

The differences are minor, so most documents will need very few changes, if any. Once you've made the necessary changes, this section also describes how to take advantage of the new features provided in Asciidocor.



This section specifically covers migration from AsciiDoc Python 8 to Asciidocor 1.5.x. The content assumes you've already updated any content that is deprecated as of AsciiDoc Python 8.

## 90.1. Command Line Interface

Asciidocor was designed from the outset to be a (much faster) drop-in replacement for AsciiDoc Python. For most documents, you can simply replace the call to AsciiDoc Python:

```
$ asciidoc document.adoc
```

with an equivalent call to Asciidocor:

```
$ asciidoc -a compat-mode document.adoc
```

If you run into trouble, check out the differences below, namely [Deleted and Deprecated Syntax and Attributes](#).



Keep in mind you can also run Asciidocor on the JVM using [AsciidocorJ](#) or with JavaScript using [Asciidocor.js](#).

### 90.1.1. Help Topics

In both AsciiDoc Python and Asciidocor, the `--help` CLI option shows the command usage by default. They differ in how they handle the optional topic argument.

In AsciiDoc Python, the `--help syntax` topic shows a syntax cheatsheet and the `--help manpage` topic shows a plaintext version of the man page.

Asciidocor only supports the `manpage` help topic. However, it outputs the formatted man page rather than the plaintext version. Therefore, to view it, you need to pipe the result to the `man` command as follows:

```
$ asciidoc --help manpage | man /dev/stdin
```

or

```
$ asciidoctor --help manpage | man -l -
```

If you want to view the plaintext version, you can route the output through the `col` command as follows:

```
$ asciidoctor --help manpage | man -l - | col -bx
```

You can also view the man page for Asciidoctor online at [asciidoctor\(1\)](#).

To get help with the AsciiDoc syntax in Asciidoctor, refer to the [AsciiDoc Syntax Quick Reference](#).

### 90.1.2. Configuration Files

Asciidoctor does not use .conf files or filters, so `--conf-file`, `--dump-conf`, and `--filter` are not implemented. Asciidoctor provides an [extension API](#) that replaces the configuration-based extension and filter mechanisms in AsciiDoc Python.

### 90.1.3. Internationalization

AsciiDoc Python has built-in .conf files that are used to translate built-in labels. You load the .conf file for a given language by setting the `lang` attribute to a supported language code (e.g., `-a lang=<language code>`). In Asciidoctor, you must define the translations for these labels explicitly. See [Using Asciidoctor with Other Languages](#) for details.

### 90.1.4. Themes

AsciiDoc Python provides a theming mechanism that encapsulates CSS, JavaScript and images. The `--theme` option activates one of these themes, which is resolved from your home directory. In Asciidoctor, you control the theme using CSS (i.e., a stylesheet) only, which you can specify using `-a stylesheet=<stylesheet>`.

If you require more advanced theming, you can inject additional resources using a [docinfo file](#) or use a postprocessor extension.

### 90.1.5. Default HTML Backend

AsciiDoc Python uses XHTML 1.1 as its default output (the `xhtml11` backend), though it supports HTML5 output as well (the `html5` backend). Asciidoctor defaults to creating HTML5 output (the `html5` backend), which closely adheres to the backend by the same name in AsciiDoc Python. The web has moved forward since AsciiDoc Python was created, so the switch to HTML5 is recommended anyway.

### 90.1.6. Doctest

AsciiDoc Python `--doctest` runs units tests. See [Tests](#) for how to run the Asciidoctor unit tests.

Asciidoctor also has a [doctest tool](#) which you can use when creating custom HTML or XML-based converters.

## 90.2. Changed Syntax

These changes are not backward-compatible, but if you set the `compat-mode` attribute, Asciidoctor will accept the AsciiDoc Python syntax. For the long term, you should update to the Asciidoctor syntax. Consult the [Migration Guide](#) to get the full details and learn how to migrate smoothly.

*AsciiDoc syntax affected by compat mode*

Feature	AsciiDoc Python (or Asciidoctor in compat mode)	Asciidoctor (no compat mode)
<i>italic text</i>	'italic text' or <code>_italic text_</code>	<code>_italic text_</code>
<i>monospaced text</i>	<code>+monospaced text+</code>	<code>`monospaced text`</code>
<i>monospaced text</i> (literal)	<code>`{asciidoc-version}`</code>	<code>`+{asciidoc-version}+`</code>
<i>``double quotes''</i>	<code>``double quotes''</code> <i>not available in compat mode</i>	"`double quotes`" or insert the Unicode quote characters using your editor
<code>`single quotes'</code>	<code>`single quotes'</code> <i>not available in compat mode</i>	<code>``single quotes``</code> or insert the Unicode quote characters using your editor
Document title <sup>[1]</sup>	<code>Title</code> or <code>= Title</code> <code>=====</code>	<code>= Title</code>

<sup>[1]</sup> Asciidoctor accepts the two-line heading style to set the document title. However, by using it, you implicitly set `compat-mode`. If you want to use the new Asciidoctor syntax, make sure to use the single-line style for the document title or unset the `compat-mode` attribute explicitly.

The following changes are not affected by the `compat-mode` attribute:

*AsciiDoc syntax not affected by compat mode*

Feature	AsciiDoc Python	Asciidoctor
Underlined titles	Underline length must match title length +/- 2 characters.	Underline length must match title length +/- 1 character (Underlined titles are deprecated anyway. See <a href="#">Sections</a> .)
<code>ifeval::[]</code>	Evaluates any Python expression.	Evaluates simple logical expressions testing the value of attributes. See <a href="#">ifeval directive</a> .
Block delimiters	Delimiter lines do not have to match in length.	The length of start and end delimiter lines must match exactly.
AsciiDoc table cell	<code>a </code> or <code>asciidoc </code>	<code>a </code> only
Table cell separator	A Python regular expression.	One or more literal characters or <code>\t</code> for tab.

## 90.3. Deleted and Deprecated Syntax and Attributes

These are attributes that either no longer exist, work differently, or have better alternatives.

### *Deleted and deprecated syntax and attributes*

AsciiDoc Python	Asciidoc	Notes
<code>big, small, underline, overline, line- through, colors</code>	<i>deprecated</i>	Character attributes to apply formatting directly. Usually better to apply a role, then apply the formatting based on that role by using a stylesheet.
<code>halign, valign</code> for table cells	Column and cell specifiers	See <a href="#">Cell Formatting</a> .
<code>infile</code>	<i>not implemented</i>	Provides the name and directory of the current document. (Distinct from <code>docfile</code> , because <code>infile</code> may be an included document, and <code>docfile</code> is always the master document.)
<code>indir</code>		No Asciidoc equivalent.
<code>asciidoc</code>	<code>asciidoc</code>	AsciiDoc Python sets <code>asciidoc</code> to show that it is the current processor. Asciidoc sets <code>asciidoc</code> instead.
<code>toc2</code>	<code>toc</code>	Combined in a single attribute, see <a href="#">Table of Contents</a> .
<code>toc-placement</code>		
<code>toc-position</code>		
<code>options="pgwide"</code>	<i>not implemented</i>	DocBook attribute to make tables full page width, whatever the current indent. No Asciidoc equivalent.
<code>options="unbreakable"</code>		In Asciidoc, this only works for tables, not paragraphs.
<code>plaintext</code>	<i>not implemented</i>	AsciiDoc Python uses this to suppress inline substitutions and retain block indents when importing large blocks of plain text. Asciidoc deliberately does not implement it; the closest Asciidoc equivalent is a passthrough block.
<code>replacements2</code>	<code>post_replacements</code>	Renamed.
<code>presubs</code>	-	Not required.
<code>showcomments</code>	<i>not implemented</i>	In AsciiDoc Python, turns single line comments into DocBook <code>&lt;remark&gt;</code> elements. Asciidoc considers this an inappropriate use of comments. If you want to send remarks to the output, use an extension, or:
		<pre>ifdef::showcomments+basebackend-docbook[] ++++ &lt;remark&gt;Your comment here&lt;/remark&gt; ++++ endif::[]</pre>

AsciiDoc Python	Asciidoctor	Notes
<code>specialwords</code>	<i>not implemented</i>	In AsciiDoc Python, applies special formatting to named text. In Asciidoctor this could be implemented using an extension.
<code>tabsize</code> (in-document and include directive)	in-document only	AsciiDoc Python replaces tabs with spaces in all text, using a default tab size of 8. Asciidoctor only replaces tabs with spaces in verbatim content blocks (listing, literal, etc), and the attribute has no default. In other words, tabs are not expanded in verbatim content blocks unless this attribute is set on the block or the document. For all other text, Asciidoctor tabs are fixed at 4 spaces by the CSS. See <a href="#">Normalize Block Indentation</a> for more detail.

## 90.4. Default HTML Stylesheet

You'll notice that the AsciiDoc Python and Asciidoctor stylesheets look quite different. However, they are compatible (for the most part) since the formatting is based on the same HTML structure and CSS classes. If you happen to prefer the AsciiDoc Python stylesheet, you can use it by copying it from the AsciiDoc Python *stylesheets* directory and instructing Asciidoctor to apply it using:

```
$ asciidoctor -a stylesheet=asciidoc.css document.adoc
```

 Keep in mind that the default stylesheet in Asciidoctor is just that, a default. If you don't like its appearance, you can either customize it or choose another stylesheet. You can find a collection of alternative themes in the [Asciidoctor Stylesheet Factory](#).

 Unlike AsciiDoc Python, Asciidoctor loads some resources from a CDN. It's possible to configure Asciidoctor to load all resources from local files. For instance, you can unset the `webfonts` attribute so that the generated HTML does not use fonts from Google Fonts. There are similar attributes to control how additional resources are resolved.

## 90.5. Mathematical Expressions

Both AsciiDoc Python and Asciidoctor can convert embedded LaTeX and AsciiMath expressions (e.g., `asciimath:[expression]`, `latexmath:[expression]`, etc), but with Asciidoctor you need to activate STEM support first using the `stem` attribute (see [Activating stem support](#)).

For block content, AsciiDoc Python uses a `[latex]` style delimited block. In Asciidoctor, use a `stem` passthrough block instead. See [Block Stem Content](#).

## 90.6. AsciiDoc Python Extensions

The extension mechanism is completely different in Asciidoctor, but the ‘standard’ extensions have been re-implemented, so they should work with minor changes.

### *Standard extensions*

AsciiDoc Python	Asciidoctor
source	<ul style="list-style-type: none"><li>• You can choose from a number of highlighters <a href="#">Syntax Highlighting Source Code</a>.</li><li>• Highlighters are built-in, not separately installed.</li><li>• <code>src_numbered</code>, <code>src_tab</code>, <code>args</code> are not implemented directly, but check the highlighter you are using for what features it has and how to configure them.</li></ul>
music	Not implemented.
latex (block macro)	Use a <code>stem</code> passthrough block <a href="#">Block Stem Content</a> .
graphviz	Incorporated into <a href="#">Asciidoctor Diagram</a> .

## 90.7. Custom Extensions

AsciiDoc Python custom extensions will not work with Asciidoctor because AsciiDoc Python extensions are essentially Python commands, and the Asciidoctor extensions are Ruby (or Java) classes. To re-write your extensions, see [Extensions](#).

## 90.8. Features Introduced by Asciidoctor

### 90.8.1. New Syntax

Asciidoctor has shorthand for id, role, style and options. See [Setting Attributes on an Element](#) for details.

The following longhand syntax in AsciiDoc Python:

```
[[id]]  
[style,role="role",options="option1,option2"]
```

can be written using the shorthand supported by Asciidoctor:

```
[style#id.role%option1%option2]
```

The longhand forms still work, but you should use the new forms for future compatibility, convenience and readability.

## 90.8.2. Enhancements

There are lots of new features and improvements Asciidoc. These are some of the more interesting ones when migrating:

- [Partial includes](#)
- [Additional safe modes](#)
- [Icon-based fonts and inline icons](#)
- [Asciidoc Diagram](#)

A detailed list of the improvements is shown in [Differences between Asciidoc and AsciiDoc Python](#).

## 90.8.3. Recommended Practices

See the [AsciiDoc Style Guide and Recommended Practices](#) for ways to make your documents clearer and more consistent.

# Chapter 91. Convert DocBook XML to AsciiDoc

One of the things Asciidoctor excels at is converting AsciiDoc source into valid and well-formed DocBook XML content.

What if you're in the position where you need to go the other way: migrate all your legacy DocBook XML content to AsciiDoc? The prescription (□) you need to get rid of your DocBook pains could be [DocBookRx](#).

DocBookRx is an early version of a DocBook to AsciiDoc converter written in Ruby. This converter is far from perfect at the moment, but it improves with each document it converts.

The plan is to evolve it into a robust library for performing this conversion in a reliable way. You can read more about this initiative in the [README](#).

The best thing about this tool is all the active users who are putting it through its paces. The more advanced the DocBook XML this tool tackles, and the more feedback we receive, the better the tool will become. Use it today to escape from XML hell!

# Chapter 92. Convert Markdown to AsciiDoc

Asciidoctor recognizes a fair amount of Markdown syntax, thus allowing you to migrate from Markdown to AsciiDoc gradually. See [Markdown Compatibility](#) to learn what syntax is shared. The syntax you must change is listed in [A selection of AsciiDoc language features compared to Markdown](#).

# Chapter 93. Convert Confluence XHTML to AsciiDoc

You can convert Atlassian Confluence XHTML pages to Asciidoc using this [Groovy](#) script.

The script calls [Pandoc](#) to convert single or multiple HTML files exported from Confluence to AsciiDoc files. You'll need Pandoc installed before running this script.



If you have trouble running this script, you can use the Pandoc command referenced inside the script to convert XHTML files to AsciiDoc manually.

*convert.groovy*

```
link:https://gist.githubusercontent.com/melix/6020336/raw/059d83a3dae933de71d585c3f6b229a3c62fa857/convert.groovy\[\]
```

The script is designed to be run locally on HTML files or directories containing HTML files exported from Confluence.

*Usage*

1. Save the script contents to a `convert.groovy` file in a working directory.
2. Make the file executable according to your specific OS requirements.
3. Place individual files, or a directory containing files into the working directory.
4. Run `groovy convert filename.html` to convert a single file.
5. Confirm the output file meets requirements
6. Recurse through a directory by using this command pattern: `groovy convert directory/*.html`

This script was created by Cédric Champeau ([melix](#)). You can find the source of this script hosted at this [GitHub Gist](#).

# Chapter 94. Convert MS Word to AsciiDoc

See [Migrating to AsciiDoc from MS Word](#).

# Resources

# Chapter 95. Copyright and License

Copyright © 2012-2018 Dan Allen, Sarah White, and Ryan Waldron. Free use of this software is granted under the terms of the MIT License.

See the [LICENSE](#) file for details.

# Chapter 96. Authors

**Asciidoctor** was written by [Dan Allen](#), [Sarah White](#), [Ryan Waldron](#), [Jason Porter](#), [Nick Hengeveld](#) and [other contributors](#).

The initial code from which Asciidoctor emerged was written by [Nick Hengeveld](#) to process the git man pages for the [Git project site](#). Refer to the commit history of [asciidoc.rb](#) to view the initial contributions.

**AsciiDoc** was written by Stuart Rackham and has received contributions from many other individuals.

# Chapter 97. Troubleshooting

## 1. Part way through the document, the blocks stop rendering correctly. What went wrong?

When content is not rendered as you expect in the later part of a document, it's usually due to a delimited block missing its closing delimiter line. The most devious culprit is the [open block](#). An open block doesn't have any special styling, but its contents have the same restrictions as other delimited blocks, i.e. it can not contain section titles.

To solve this problem, first look for missing delimiter lines. [Syntax highlighting in your text editor](#) can help with this. Also look at the rendered output to see if the block styles are extending past where you intended. When working with open blocks, you may need to add custom styles (such as a red border) to the class `openblock` so you can see its boundaries.

## 2. Why don't URLs containing underscores (\_) or carets (^) work after they're converted?

This problem occurs because the markup parser interprets parts of the URL (i.e., the link target) as valid [text formatting markup](#). Most lightweight markup languages have this issue because they don't use a grammar-based parser. Asciidoctor plans to handle URLs more carefully in the future (see [issue #281](#)), which may be solved by moving to a grammar-based parser (see [issue #61](#)). Thankfully, there are many ways to include URLs of arbitrary complexity using the AsciiDoc passthrough mechanisms.

### Solution A

The simplest way to get a link to behave is to assign it to an attribute.

```
= Document Title  
:link-with-underscores: https://asciidoctor.org/now_this__link_works.html
```

This URL has repeating underscores {link-with-underscores}.

Asciidoctor won't break links with underscores when they are assigned to an attribute because [inline formatting markup is substituted before attributes](#). The URL remains hidden while the rest of the document is being formatted (strong, emphasis, monospace, etc).

### *Solution B*

Another way to solve formatting glitches is to explicitly specify the formatting you want to have applied to a span of text. This can be done by using the [inline pass macro](#). If you want to display a URL, and have it preserved, put it inside the pass macro and enable the [macros substitution](#), which is what substitutes links.

```
This URL has repeating underscores  
pass:macros[https://asciidoc.org/now_this__link_works.html].
```

The pass macro removes the URL from the document, applies the [macros](#) substitution to the URL, and then restores the processed URL to its original location once the substitutions are complete on the whole document.

Alternatively, you can use `++` around the URL only. However, when you use this approach, Asciidoctor won't recognize it as a URL any more, so you have to use the explicit [link](#) prefix.

```
This URL has repeating underscores  
link:++https://asciidoc.org/now_this__link_works.html++[].
```

For more information, see [issue #625](#).

# Glossary

## admonition

a callout paragraph or block that has a label or icon indicating its priority.

## backend

a moniker for the expected output format; used as a key to select which converter to use; often used interchangeably with the name of a converter (i.e., the "html5" backend").

## block attribute

an attribute associated with a delimited block or paragraph; these attributes can affect processing of the block, and are available to block processors, but cannot be referenced using an attribute reference.

## built-in attribute

a document attribute that controls processing, integrations, styling, and localization.

## cross reference

a link from one location in the document to another location marked by an anchor.

## document attribute

an attribute associated with the document (node); in other words, an attribute in the global document attributes dictionary; the value of these attributes can be referenced using an attribute reference; if defined in the header, the document attribute is known as a header attribute.

## environment attribute

a dynamic document attribute that pertains to, or gives information about, the runtime environment.

## header attribute

a document attribute defined in the document header; visible from all nodes in the document; often required for global settings such as the source highlighter or icons mode.

## list continuation

a plus sign (+) on a line by itself that connects adjacent lines of text to a list item.

## macro attribute

an attribute associated with a block or inline macro; these attributes can affect processing of the macro, and are available to macro processors, but cannot be referenced using an attribute reference.

## predefined attribute

a document attribute defined for convenience; often used for inserting special content characters.

**quoted text**

text which is enclosed in special punctuation to give it emphasis or special meaning.

**user-defined attribute**

a document attribute defined by the content author; used for storing reusable content, and controlling conditional inclusion.

# Appendix A: Catalog of Document Attributes

This appendix catalogs all the recognized document attributes in Asciidoctor. It includes environment, built-in and predefined (aka character reference) attributes. Authors may define any number of additional attributes (aka user-defined attributes) for their own purposes.

## A.1. Environment Attributes

Asciidoctor automatically assigns values to various document attributes whenever a document is loaded or converted. These attributes, termed *environment attributes*, provide information about the runtime environment, such as how, where and when the document is being processed. Like other document attributes, environment attributes can be referenced wherever attribute references are permitted. It's recommended that you treat these attributes as read only.

### *Environment attributes*

Attribute	Description	Example Value
<code>asciidoc</code>	Set if the current processor is Asciidoctor.	
<code>asciidoc-version</code>	Asciidoctor version.	<code>2.0.10</code>
<code>backend</code>	Backend used to create the output file.	<code>html5</code>
<code>basebackend</code>	The backend value minus any trailing numbers. For example, if the backend is <code>docbook5</code> , the basebackend is <code>docbook</code> .	<code>html</code>
<code>docdate</code>	Last modified date of the source document. <sup>[1,2]</sup>	<code>2019-01-04</code>
<code>docdatetime</code>	Last modified date and time of the source document. <sup>[1,2]</sup>	<code>2019-01-04 19:26:06 UTC</code>
<code>docdir</code>	Full path of the directory that contains the source document.	<code>/home/user/docs</code>
<code>docfile</code>	Full path of the source document.	<code>/home/user/docs/userguide.adoc</code>
<code>docfilesuffix</code>	File extension of the source document, including the leading period. <i>Introduced in 1.5.6.</i>	<code>.adoc</code>
<code>docname</code>	Root name of the source document (no leading path or file extension).	<code>userguide</code>
<code>doctime</code>	Last modified time of the source document. <sup>[1,2]</sup>	<code>19:26:06 UTC</code>
<code>doctype</code>	Document type (article, book or manpage).	<code>article</code>
<code>docyear</code>	Year that the document was last modified. <sup>[1,2]</sup>	<code>2018</code>
<code>embedded</code>	Set if content is being converted to an embeddable document (body only).	
<code>filetype</code>	File extension of the output file name (without leading period).	<code>html</code>

Attribute	Description	Example Value
<code>htmlsyntax</code>	Syntax used when generating the HTML output (html or xhtml).	<code>html</code>
<code>localdate</code>	Date when the document was converted. <sup>[2]</sup>	<code>2019-02-17</code>
<code>localdatetime</code>	Date and time when the document was converted. <sup>[2]</sup>	<code>2019-02-17 19:31:05 UTC</code>
<code>localtime</code>	Time when the document was converted. <sup>[2]</sup>	<code>19:31:05 UTC</code>
<code>localyear</code>	Year when the document was converted. <sup>[2]</sup>	<code>2018</code>
<code>outdir</code>	Full path of the output directory.	<code>/home/user/docs/dist</code>
<code>outfile</code>	Full path of the output file.	<code>/home/user/docs/dist/usersguide.html</code>
<code>outfilesuffix</code>	File extension of the output file (starting with a period) as determined by the backend ( <code>.html</code> for <code>html</code> , <code>.xml</code> for <code>docbook</code> , etc.). (The value is not updated to match the file extension of the output file when one is specified explicitly). <i>Safe to modify</i> .	<code>.html</code>
<code>safe-mode-level</code>	Numeric value of the safe mode setting. (UNSAFE=0, SAFE=10, SERVER=10, SECURE=20).	<code>20</code>
<code>safe-mode-name</code>	Textual value of the safe mode setting.	<code>SERVER</code>
<code>safe-mode-unsafe</code>	Set if the safe mode is UNSAFE.	
<code>safe-mode-safe</code>	Set if the safe mode is SAFE.	
<code>safe-mode-server</code>	Set if the safe mode is SERVER.	
<code>safe-mode-secure</code>	Set if the safe mode is SECURE.	
<code>user-home</code>	Home directory of the current user. Resolves to <code>.</code> if the safe mode is SERVER or greater.	<code>/home/user</code>

<sup>[1]</sup> Only reflects the last modified time of the source document file. It does not consider the last modified time of files which are included.

<sup>[2]</sup> If the `SOURCE_DATE_EPOCH` environment variable is set, the value assigned to this attribute is built from a UTC date object that corresponds to the timestamp (as an integer) stored in that environment variable. This override offers one way to make the conversion reproducible. See <https://reproducible-builds.org/specs/source-date-epoch/> for more information about the `SOURCE_DATE_EPOCH` environment variable. Otherwise, the date is expressed in the local time zone, which is reported as a time zone offset (e.g., `-0600`) or UTC if the time zone offset is 0). To force the use of UTC, set the `TZ=UTC` environment variable when invoking Asciidoctor.

## A.2. Built-in Attributes

Built-in document attributes can be set anywhere in the document using an attribute entry line. However, there are some rules to keep in mind regarding the impact of that assignment.

- Many attributes can only be defined in the [document header](#) (or via the API or CLI). Otherwise, they won't have the desired impact. These are called *header attributes*. This requirement is marked in the table below.
- To set an attribute that does not accept a value, simply use an empty value (as indicated by `empty` in the table).
- If you set an attribute from the command line or API, it's defined for the whole document and cannot be changed in the body unless `@` is added to the end of the value. (The one exception to this rule is the `sectnums` attribute, which can always be changed).
- If you set an attribute in the body (anywhere after the document header), the attribute is visible from the point it is set until it is unset (assuming it is not locked as a result of being set from the command line or API).



Several attributes from AsciiDoc Python have been removed (or deprecated) in Asciidoctor and therefore are not included in this section. See [Migrating from AsciiDoc Python](#) if you are updating an older document.

#### Built-in document attributes

Attribute name	Description	Default value <sup>[1]</sup>	Allowed values	Header only	See also
<strong>Compliance</strong>					
attribute-missing	Controls how missing references are handled.	skip	skip, drop, drop-line or warn		<a href="#">[catch-a-missing-or-undefined-attribute]</a>
attribute-undefined	Controls how expressions that undefine an attribute are handled.	drop-line	drop or drop-line		<a href="#">[catch-a-missing-or-undefined-attribute]</a>
compat-mode	Controls whether the legacy parser is used to parse the document.	<i>not set</i> (Modern parser is used).	<i>empty</i> (Legacy parser is used).		<a href="#">Migrating from AsciiDoc Python</a>
experimental	Enable experimental extensions. The features behind this attribute are subject to change and may even be removed in a future version. Currently enables the UI macros (button, menu and kbd).	<i>not set</i>	<i>empty</i>	✓	<a href="#">User Interface Macros</a>

Attribute name	Description	Default value <sup>[1]</sup>	Allowed values	Header only	See also
reproducible	If set, prevents the last-updated date from being added to the HTML footer or DocBook info element. Alternatively, you can use the SOURCE_DATE_EPOCH environment variable to fix the value. Useful if you want to store the output in a source code control system as it prevents spurious changes every time you convert the document.	<i>not set</i>	<i>empty</i>	✓	

## Internationalization and numbering

appendix-caption	Sets the text used to prefix appendix titles.	Appendix	<i>any</i>		<a href="#">Appendix, Translating built-in labels</a>
appendix-number	Stores the number (aka letter) for the current appendix. <sup>[2]</sup>	A	<i>letter</i>		
appendix-refsig	Sets the signifier added to the title of appendices in cross reference text. Unset to disable.	Appendix	<i>any</i>		<a href="#">Book Parts and Chapters, Sections, Translating built-in labels</a>
caution-caption	Sets the text used to label CAUTION admonitions when icons are not enabled.	Caution	<i>any</i>		<a href="#">Admonition, Translating built-in labels</a>
chapter-number	Stores the number for the current chapter. <sup>[2]</sup>	2	<i>integer</i>		
chapter-refsig	Sets the signifier added to the title of numbered chapters in cross reference text. Unset to disable.	Chapter	<i>any</i>		<a href="#">Book Parts and Chapters, Sections, Translating built-in labels</a>

Attribute name	Description	Default value <sup>[1]</sup>	Allowed values	Header only	See also
chapter-label	Sets the prefix added to chapter titles (i.e., level-1 section titles when doctype is book). ( <i>pdf converter only</i> )	Chapter	<i>any</i>		<a href="#">Book Parts and Chapters, Sections, Translating built-in labels</a>
example-caption	Sets the text used to label example blocks.	Example	<i>any</i>		<a href="#">Translating built-in labels</a>
example-number	Stores the number for the current numbered example. <sup>[2]</sup>	5	<i>integer</i>		
figure-caption	Sets the text used to label figures.	Figure	<i>any</i>		<a href="#">Images, Translating built-in labels</a>
figure-number	Stores the number for the current numbered figure. <sup>[2]</sup>	3	<i>integer</i>		
important-caption	Sets the text used to label IMPORTANT admonitions when icons are not enabled.	Important	<i>any</i>		<a href="#">Admonition, Translating built-in labels</a>
lang	Set the value of the <code>lang</code> attribute on the root element of the output document.	en	Valid XML country code	✓	<a href="#">Translating built-in labels</a>
last-update-label	Text label for the “Last updated” time in the footer. Unsetting it will remove the label and time from the footer.	Last updated	<i>any</i>	✓	<a href="#">Footer docinfo files, Translating built-in labels</a>
listing-caption	Sets the text used to label listing blocks.	<i>not set</i>	<i>any</i>		<a href="#">Translating built-in labels</a>
listing-number	Stores the number for the current numbered listing. <sup>[2]</sup>	5	<i>integer</i>		
manname-title	Label for the program name section in the man page.	NAME	<i>any</i>	✓	<a href="#">Translating built-in labels</a>
nolang	Prevents the <code>lang</code> attribute from being added to root element of the output document.	<i>not set</i>	<i>empty</i>	✓	

Attribute name	Description	Default value <sup>[1]</sup>	Allowed values	Header only	See also
note-caption	Sets the text used to label NOTE admonitions when icons are not enabled.	Note	<i>any</i>		<a href="#">Admonition</a> , <a href="#">Translating built-in labels</a>
preface-title	Sets the title text for an anonymous preface when the doctype is book.	<i>not set</i>	<i>any</i>		<a href="#">Preface</a>
section-refsig	Sets the signifier added to the title of numbered sections in cross reference text. Unset to disable.	Section	<i>any</i>		<a href="#">Book Parts and Chapters</a> , <a href="#">Sections</a> , <a href="#">Translating built-in labels</a>
table-caption	Text of the label that is automatically prefixed to table titles. To turn off table caption labels and numbers, add the <a href="#">table-caption</a> attribute to the document header with an empty value.	Table	<i>any</i>		<a href="#">Translating built-in labels</a>
table-number	Stores the number for the current numbered table. <sup>[2]</sup>	5	<i>integer</i>		
tip-caption	Sets the text used to label TIP admonitions when icons are not enabled.	Tip	<i>any</i>		<a href="#">Admonition</a> , <a href="#">Translating built-in labels</a>
toc-title	Title for the table of contents.	Table of Contents	<i>any</i>		<a href="#">Table of Contents</a> , <a href="#">Translating built-in labels</a>
untitled-label	Used as the default document title if the document does not have a document title.	Untitled	<i>any</i>	✓	<a href="#">Translating built-in labels</a>
version-label	The label preceding the revnumber in a converted document's byline	Version	<i>any</i>	✓	<a href="#">Revision Number</a> , <a href="#">Date and Remark</a> , <a href="#">Translating built-in labels</a>
warning-caption	Sets the text used to label TIP admonitions when icons are not enabled.	Warning	<i>any</i>		<a href="#">Translating built-in labels</a>

Attribute name	Description	Default value <sup>[1]</sup>	Allowed values	Header only	See also
<strong>Header and metadata</strong>					
app-name	Application name (for mobile devices). If set, adds an <a href="#">application-name</a> meta element inside the HTML document head.	<i>not set</i>	<i>any</i>	✓	
author	Sets the document's main author. Can be set automatically via the author info line.	<i>not set</i>	<i>any</i>	✓	<a href="#">Header</a>
authorinitials	Sets the author's initials (e.g., JD). Derived automatically from the author attribute by default.	<i>not set</i>	<i>any</i>	✓	<a href="#">Header</a>
authors	Sets the document authors as a comma-separated list. Can be set automatically via the author info line. If set, adds an <a href="#">author</a> meta element inside the HTML document head.	<i>not set</i>	<i>any</i>	✓	<a href="#">Metadata</a>
copyright	If set, adds a <a href="#">copyright</a> meta element inside the HTML document head.	<i>not set</i>	<i>any</i>	✓	<a href="#">Metadata</a>
doctitle	Sets the document title. Set automatically to section title if document begins with level-0 section.	Based on content.	<i>any</i>	✓	<a href="#">Document Title</a>
description	If set, adds a <a href="#">description</a> meta element inside the HTML document head.	<i>not set</i>	<i>any</i>	✓	<a href="#">Metadata</a>
email	Sets the author's email address. Can be set automatically via the author info line. Can be any inline macro, such as a URL.	<i>not set</i>	<i>any</i>	✓	<a href="#">Header</a>
favicon	Adds a link to a favicon to the HTML <a href="#">&lt;head&gt;</a> .	<i>not set</i>	<i>any</i>	✓	<a href="#">Favicon</a>

Attribute name	Description	Default value <sup>[1]</sup>	Allowed values	Header only	See also
firstname	Sets the author's first name. Can be set automatically via the author info line.	<i>not set</i>	<i>any</i>	✓	<a href="#">Header</a>
front-matter	If <code>skip-front-matter</code> is set via the CLI or API, any YAML-style frontmatter skimmed from the top of the document is stored in this attribute.	Front matter content, if captured.	<i>any</i>	✓	<a href="#">Front Matter Added for Static Site Generators</a>
keywords	If set, adds a <code>keywords</code> meta element inside the HTML document head.	<i>not set</i>	<i>any</i>	✓	<a href="#">Metadata</a>
lastname	Sets the author's last name. Can be set automatically via the author info line.	<i>not set</i>	<i>any</i>	✓	<a href="#">Header</a>
middlename	Sets the author's middle name or initial. Can be set automatically via the author info line.	<i>not set</i>	<i>any</i>	✓	<a href="#">Header</a>
orgname	If set, add an <code>&lt;orgname&gt;</code> element with this value to the DocBook info element.	<i>not set</i>	<i>any</i>	✓	<a href="#">Metadata</a>
revdate	Sets the revision date. Can be set automatically via the revision info line.	<i>not set</i>	<i>any</i>	✓	<a href="#">Header</a>
revremark	Sets the revision description. Can be set automatically via the revision info line.	<i>not set</i>	<i>any</i>	✓	<a href="#">Header</a>
revnumber	Sets the revision number. Can be set automatically via the revision info line.	<i>not set</i>	<i>any</i>	✓	<a href="#">Header</a>

Attribute name	Description	Default value <sup>[1]</sup>	Allowed values	Header only	See also
title	Sets the value of the <code>&lt;title&gt;</code> element in the HTML <code>&lt;head&gt;</code> or main DocBook <code>&lt;info&gt;</code> of the output document. Also used as a fallback when the document title is not specified. <i>Since this is a reserved attribute that has special behavior, you should avoid using it for any other purpose!</i>	<i>not set</i>	<i>any</i>	✓	<a href="#">Document Title</a>

## Section titles and table of contents

idprefix	Sets prefix used for auto-generated section IDs.	—	Valid XML ID start character.	✓	<a href="#">Auto-generated IDs</a>
idseparator	Sets word separator used in auto-generated section IDs.	—	Valid XML ID character.	✓	<a href="#">Auto-generated IDs</a>
leveloffset	Pushes the level of subsequent headings down, to make file inclusion more useful.	0	(+/-)0–5. (A leading + or - makes it relative).		<a href="#">Partitioning large documents and using leveloffset</a>
sectanchors	If set, adds an anchor in front of the section title when the mouse cursor hovers over it.	<i>not set</i> (No anchors).	<i>empty</i>	✓	<a href="#">Anchors</a>
sectids	If set, generates and assigns an ID to any section that does not have one.	<i>empty</i> (Assigns section ID if not specified).	<i>empty</i>	✓	<a href="#">Auto-generated IDs</a>
sectlinks	Turns section titles into self-referencing links.	<i>not set</i>	<i>empty</i>	✓	<a href="#">Links</a>
sectnums	If set, numbers sections to depth specified by sectnumlevels.	<i>not set</i> (Sections are not numbered).	<i>empty</i>	✓	<a href="#">Numbering</a>
sectnumlevels	controls the depth of section numbering	3	0–5	✓	<a href="#">Numbering depth</a>

Attribute name	Description	Default value <sup>[1]</sup>	Allowed values	Header only	See also
title-separator	The character used to separate the main title and subtitle in the document title.	:	any	✓	<a href="#">Subtitle Partitioning</a>
toc	Switches the table of contents on, and defines its location.	not set	auto, left, right, macro or preamble	✓	<a href="#">Table of Contents</a>
toclevels	Maximum section level to display.	2	1–5	✓	<a href="#">Table of Contents</a>
fragment	Hints to parser that document is a fragment and it should not enforce proper section nesting.	not set	empty		

### General content and formatting

asset-uri-scheme	Controls which protocol is used for assets hosted on a CDN.	https	empty, http or https	✓	
cache-uri	If set, cache content read from URIs.	not set	empty	✓	<a href="#">Caching URI Content</a>
data-uri	Embed graphics as data-uri elements in HTML elements so the file is completely self-contained.	not set (Images are linked, not embedded).	empty	✓	<a href="#">Managing Images</a>
docinfo	Read input from one or more DocBook info files.	not set	Comma-separated list of the following values: shared, private, shared-head, private-head, shared-footer or private-footer	✓	<a href="#">Naming docinfo files</a>
docinfodir	The location where docinfo files are resolved.	The base directory.	Directory	✓	<a href="#">Locating docinfo files</a>

Attribute name	Description	Default value <sup>[1]</sup>	Allowed values	Header only	See also
docinfosubs	The AsciiDoc substitutions that get applied to docinfo content.	attributes	Comma-separated list of substitution names. Set value to empty or <code>none</code> to disable substitutions.	✓	<a href="#">Attribute substitution in docinfo files</a>
doctype	Set the output document type.	article	article, book, inline or manpage	✓	<a href="#">Document Types</a>
eqnums	Controls automatic equation numbering on LaTeX blocks in HTML output (MathJax). If the value is AMS, only LaTeX content enclosed in an <code>\begin{equation}... \end{equation}</code> container will be numbered. If the value is all, then all LaTeX blocks will be numbered.	<i>not set</i> (Equation numbering is off)	<i>empty</i> (alias for AMS), AMS, all or none	✓	<a href="#">Equations and Formulas</a> , <a href="#">Equation numbering in MathJax</a>
hardbreaks	Preserve hard line breaks in the input.	<i>not set</i>	<i>empty</i>		<a href="#">Line Breaks</a>
hide-uri-scheme	Hides the URI scheme for all raw links.	<i>not set</i>	<i>empty</i>		<a href="#">URLs</a>
linkattrs	Parse attributes inside the link macro. Removed in Asciidoctor 1.5.7. Attributes are now parsed automatically if an equal sign is found after a comma (e.g., <code>[link text,window=_blank]</code> ).	<i>not set</i> (Do not parse).	<i>empty</i>		<a href="#">URLs</a>
media	Specifies the media type of the output, which may enable behavior specific to that media type.	<i>screen</i>	screen or print	✓	

Attribute name	Description	Default value <sup>[1]</sup>	Allowed values	Header only	See also
nofooter	Suppresses output of the footer.	<i>not set</i>	<i>empty</i>	✓	<a href="#">Footer docinfo files</a>
nofootnotes	Turn off display of footnotes.	<i>not set</i>	<i>empty</i>	✓	<a href="#">Footnotes</a>
noheader	Suppresses output of the header.	<i>not set</i>	<i>empty</i>	✓	<a href="#">Header</a>
outfilesuffix	File extension of the output file (starting with a period).  <small>(.html for html, .xml for docbook, etc).</small>	Determined by the backend	File extension		<a href="#">Navigating Between Source Files</a>
pagewidth	Page width, used to calculate the absolute width of tables in the DocBook output.	425	Number	✓	
relfileprefix	The path prefix to add to relative xrefs.	<i>empty</i>	Path segment		<a href="#">Navigating Between Source Files</a>
show-link-uri	Prints the URI of a link after the linked text. ( <i>pdf converter only</i> )	<i>not set</i>	<i>empty</i>	✓	
showtitle	If set, displays an embedded document's title. Mutually exclusive with the <code>notitle</code> attribute.	<i>not set</i>	<i>empty</i>	✓	<a href="#">Document Title</a>
stem	Enables mathematics processing or sets the processor used.	<i>not set</i>	<i>empty</i> (defaults to asciimath), asciimath or latexmath	✓	<a href="#">Inline Stem Content</a>
tabsize	If set, converts tabs to spaces in verbatim content blocks (e.g., listing, literal).	<i>not set</i>	0 or more	-	

Attribute name	Description	Default value <sup>[1]</sup>	Allowed values	Header only	See also
webfonts	Control whether webfonts are loaded, and which ones, when using the default stylesheet. The value populates the <code>family</code> query string parameter in the Google Fonts URL.	<i>empty</i> (use default fonts)	<i>empty</i> or a Google Fonts collection spec	✓	<a href="#">Applying a Theme</a> and issue #410
xmlns	The XML namespace to add to the DocBook 4.5 document. (The DocBook 5 document always declares a namespace).	<i>not set</i>	<i>empty</i> (alias for the DocBook namespace) or a valid XML namespace.	✓	<a href="#">DocBook</a>
xrefstyle	The formatting style to apply to cross reference text. <i>Introduced in 1.5.6.</i>	<i>not set</i>	full, short, or basic	✓	<a href="#">Customizing the Cross Reference Text</a>

## Images and icons

iconfont-cdn	Overrides the CDN used to resolve the Font Awesome stylesheet. Only used when <code>icons</code> attribute is set to <code>font</code> .	cdnjs	URI	✓	<a href="#">Icons</a>
iconfont-name	Overrides the name of the icon font stylesheet. Only used when <code>icons</code> attribute is set to <code>font</code> .	font-awesome	any	✓	<a href="#">Icons</a>
iconfont-remote	If set, allows use of a CDN for resolving the icon font. Only used when <code>icons</code> attribute is set to <code>font</code> .	<i>empty</i>	<i>empty</i>	✓	<a href="#">Icons</a>
icons	Chooses icons instead of text for admonitions.	<i>not set</i> (image)	font or image	✓	<a href="#">Icons</a>
iconsdir	Where icons are stored. Only used when <code>icons</code> attribute is set to <code>image</code> .	{imagesdir}/icons (or ./images/icons if imagesdir is not set)	Directory	✓	<a href="#">Icons</a>

Attribute name	Description	Default value <sup>[1]</sup>	Allowed values	Header only	See also
icontype	File type for image icons. Only used when <code>icons</code> attribute is set to <code>image</code> .	png	any, but typically jpg, tiff, etc.	✓	<a href="#">Icons</a>
imagesdir	Where image files are resolved.	<i>not set</i> (Same directory as document).	Directory		<a href="#">Images</a>

### Code highlighting and formatting

coderay-css	Controls whether CodeRay uses CSS classes or inline styles.	class	class or style	✓	<a href="#">CodeRay</a>
coderay-linenums-mode	Sets how CodeRay inserts line numbers into source listings.	table	table or inline	✓	<a href="#">CodeRay</a>
coderay-unavailable	If set, tells the processor not to attempt to load CodeRay.	<i>not set</i>	<i>empty</i>	✓	<a href="#">CodeRay</a>
highlightjsdir	Location of the highlight.js source code highlighter library.	<i>not set</i>	Directory	✓	<a href="#">highlight.js</a>
highlightjs-theme	Sets the name of the theme used by the highlight.js source code highlighter.	github	A style name recognized by highlight.js.	✓	<a href="#">highlight.js</a>
prettifydir	Location of the prettify source code highlighter library.	<i>not set</i> (Uses CDN).	Directory	✓	<a href="#">Syntax Highlighting Source Code</a>
prettify-theme	Sets the name of the theme used by the prettify source code highlighter.	prettify	A style name recognized by prettify.	✓	<a href="#">Syntax Highlighting Source Code</a>
prewrap	Wrap wide code listings. Sets the default behavior only; you can still switch off wrapping on specific listings.	<i>empty</i> (Code listing will wrap long lines, not scroll).	<i>empty</i>		<a href="#">To Wrap or to Scroll</a>
pygments-css	Controls whether Pygments uses CSS classes or inline styles.	class	class or style	✓	<a href="#">Pygments</a>

Attribute name	Description	Default value <sup>[1]</sup>	Allowed values	Header only	See also
pygments-linenums-mode	Sets how Pygments inserts line numbers into source listings.	table	table or inline	✓	<a href="#">Pygments</a>
pygments-style	Sets the name of the style used by the Pygments source code highlighter	default	A style name recognized by Pygments.	✓	<a href="#">Pygments</a>
pygments-unavailable	If set, tells the processor not to attempt to load Pygments.	not set	empty	✓	<a href="#">Pygments</a>
source-highlighter	Source code highlighter to use.	not set	coderay, highlightjs, prettify or pygments	✓	<a href="#">Syntax Highlighting Source Code</a>
source-indent	Normalize block indentation in code listings.	not set (Indentation is not modified).	Number		<a href="#">Normalize Block Indentation</a>
source-language	Set the default language for source code listings.	not set	Code language name in lowercase.		<a href="#">Syntax Highlighting Source Code</a>
source-linenums-option	Turns on line numbers option by default for source code listings. <i>Introduced in 1.5.6.</i>	not set	empty		<a href="#">Syntax Highlighting Source Code</a>

## HTML styling

copycss	If set, copy the CSS files to the output.	empty (File is copied if <a href="#">linkcss</a> is set).	Empty or the location of the custom stylesheet (if used)	✓	<a href="#">Applying a Theme</a>
css-signature	If set, assign the value to the <a href="#">id</a> attribute of the <code>&lt;body&gt;</code> element (HTML only). The preferred approach is to assign an ID to the document title.	not set	Valid XML ID	✓	

Attribute name	Description	Default value <sup>[1]</sup>	Allowed values	Header only	See also
linkcss	If set, link to the stylesheet instead of embedding it. Cannot be unset in SECURE safe mode.	<i>not set</i> (when safe mode < SECURE) <i>set</i> (when safe mode is SECURE)	<i>empty</i>	✓	<a href="#">Styling the HTML with CSS</a>
max-width	Constrain the maximum width of the document body. <b>Not recommended. Use custom CSS instead.</b>	<i>not set</i>	CSS length (e.g. 55em, 12cm, etc)	✓	
stylesdir	Location for resolving CSS stylesheets.	. (Same directory as document).	Directory	✓	<a href="#">Creating a Theme</a>
stylesheet	Name of a CSS stylesheet to replace the default one.	<i>not set</i> (The default stylesheet is used).	File name	✓	<a href="#">Applying a Theme</a>
toc-class	The CSS class on the table of contents container.	toc	Valid CSS class name	✓	<a href="#">Table of Contents</a>

#### Manpage attributes (relevant only when using the manpage doctype and/or converter)

mantitle	Metadata for manpage output.	Based on content.	<i>any</i>	✓	<a href="#">Man Pages</a>
manvolnum	Metadata for manpage output.	Based on content.	<i>any</i>	✓	<a href="#">Man Pages</a>
manname	Metadata for manpage output.	Based on content.	<i>any</i>	✓	<a href="#">Man Pages</a>
manpurpose	Metadata for manpage output.	Based on content.	<i>any</i>	✓	<a href="#">Man Pages</a>
man-linkstyle	Style the links in the manpage output.	blue R <>	Link format sequence	✓	<a href="#">Man Pages</a>
mansource	The source (e.g., application and version) to which the man page pertains.	<i>not set</i>	<i>any</i>	✓	<a href="#">Man Pages</a>
manmanual	Manual name displayed in the man page footer.	<i>not set</i>	<i>any</i>	✓	<a href="#">Man Pages</a>

#### Secure attributes (can only be set from the command line or API, typically for security reasons)

Attribute name	Description	Default value <sup>[1]</sup>	Allowed values	Header only	See also
allow-uri-read	If set, allows data to be read from URIs (via include directive, image macro when embedding images, etc.).	<i>not set</i>	<i>empty</i>	CLI or API	<a href="#">Include Content from a URI</a>
max-attribute-value-size	Limits the maximum size (in bytes) of a resolved attribute value. Since attributes can reference other attributes, it would be possible to create an output document disproportionately larger than the input document without this limit in place.	4096 (secure mode), <i>not set</i> (other modes)	0 or greater	CLI or API	
max-include-depth	Safety feature to curtail infinite include loops and to limit the opportunity to exploit nested includes to compound the size of the output document.	64	0 or greater	CLI or API	<a href="#">Include Directive</a>
skip-front-matter	If set, consume YAML-style frontmatter at the top of the document and store it in the <code>front-matter</code> attribute.	<i>not set</i>	<i>empty</i>	CLI or API	<a href="#">Front Matter Added for Static Site Generators</a>

<sup>[1]</sup> The default value isn't necessarily the value you will get by entering `{name}`. It may be the fallback value which Asciidoctor uses if the attribute is not defined. The effect is the same either way.

<sup>[2]</sup> The `-number` attributes are created and managed automatically by Asciidoctor for numbered blocks. They are only used if the corresponding `-caption` attribute is set (e.g., `listing-caption`) and the block has a title. In the current version of Asciidoctor, setting the `-number` attributes will influence the number used for subsequent numbered blocks of that type. However, you should not rely on this behavior as it may change in future versions.

## A.3. Predefined Attributes for Character Replacements

*Predefined attributes for character replacements* <sup>[1][2][3]</sup>

Attribute name	Replacement text	Appearance
<code>blank</code>	<code>nothing</code>	
<code>empty</code>	<code>nothing</code>	

Attribute name	Replacement text	Appearance
sp	<i>single space</i>	
nbspace	&nbsp;	
zwsp <sup>[4]</sup>	&nbsp;	
wj <sup>[5]</sup>	&nbsp;	
apos	&#39;	'
quot	&#34;	"
lsquo	&#8216;	'
rsquo	&#8217;	,
ldquo	&#8220;	"
rdquo	&#8221;	"
deg	&#176;	°
plus	&#43;	+
brvbar	&#166;	
vbar		
amp	&	&
lt	&lt;	<
gt	&gt;	>
startsb	[	[
endsb	]	]
caret	^	^
asterisk	*	*
tilde	~	~
backslash	\	\
backtick	`	`
two-colons	::	::
two-semicolons	;;	;;
cpp	C++	C++

<sup>[1]</sup> Some replacements are Unicode characters, whereas others are numeric character references (e.g., &#34;). These character references are used whenever the use of the Unicode character could interfere with the AsciiDoctor syntax or confuse the renderer (i.e., the browser). It's up to the converter to transform the reference into something the renderer understands (something both the man page and PDF converter handle).

<sup>[2]</sup> Asciidoctor does not prevent you from reassigning predefined attributes. However, it's best to treat them as read-only unless the output format requires the use of a different encoding scheme.

These attributes are an effective tool for decoupling content and presentation.

<sup><sup>[3]</sup></sup> Asciidoctor allows you to use any of the named character references (aka named entities) defined in HTML (e.g., &euro; resolves to &euro;). However, using named character references can cause problems when generating non-HTML output such as PDF because the lookup table needed to resolve these names may not be defined. Our recommendation is avoid using named character references<sup>[4]</sup> with the exception of those defined in XML (i.e., lt, gt, amp, quot and apos). Instead, use numeric character references (e.g., &#8364;).

<sup>[4]</sup> The Zero Width Space (ZWSP) is a code point in Unicode that shows where a long word can be split if necessary.

<sup>[5]</sup> The word joiner (WJ) is a code point in Unicode that prevents a line break at its position.

# Appendix B: CLI Options

## B.1. Security Settings

### **-B, --base-dir=DIR**

Base directory containing the document and resources. Defaults to the directory containing the source file, or the working directory if the source is read from a stream. When combined with the safe mode setting, can be used to chroot the execution of the program.

### **-S, --safe-mode=SAFE\_MODE**

Set safe mode level: *unsafe*, *safe*, *server* or *secure*. Disables potentially dangerous macros in source files, such as `include::[]`. If not set, the safe mode level defaults to *unsafe* when Asciidoctor is invoked using this script.

### **--safe**

Set safe mode level to *safe*. Enables include directives, but prevents access to ancestor paths of source file. Provided for compatibility with the asciidoc command. If not set, the safe mode level defaults to *unsafe* when Asciidoctor is invoked using this script.

## B.2. Document Settings

### **-a, --attribute=ATTRIBUTE**

Define, override or delete a document attribute. Command-line attributes take precedence over attributes defined in the source file unless the value ends with @.

ATTRIBUTE is normally formatted as a key-value pair, in the form *NAME=VALUE*. Alternate acceptable forms are *NAME* (where the *VALUE* defaults to an empty string), *NAME!* (unassigns the *NAME* attribute) and *NAME=VALUE@* (where *VALUE* does not override value of *NAME* attribute if it's already defined in the source document). Values containing spaces should be enclosed in quotes.

This option may be specified more than once.

### **-b, --backend=BACKEND**

Backend output file format: *html5*, *docbook5*, *docbook45* and *manpage* are supported out of the box. You can also use the backend alias names *html* (aliased to *html5*) or *docbook* (aliased to *docbook5*). Other values can be passed, but if Asciidoctor cannot resolve the backend to a converter, it will fail. Defaults to *html5*.

### **-d, --doctype=DOCTYPE**

Document type: *article*, *book*, *manpage* or *inline*. Sets the root element when using the *docbook* backend and the style class on the HTML body element when using the *html* backend. The *book* document type allows multiple level-0 section titles in a single document. The *manpage* document type enables parsing of metadata necessary to produce a man page. The *inline* document type allows the content of a single paragraph to be formatted and returned without wrapping it in a containing element. Defaults to *article*.

## B.3. Document Conversion

### **-D, --destination-dir=DIR**

Destination output directory. Defaults to the directory containing the source file, or the working directory if the source is read from a stream. If specified, the directory is resolved relative to the working directory.

### **-E, --template-engine=NAME**

Template engine to use for the custom converter templates. The gem with the same name as the engine will be loaded automatically. This name is also used to build the full path to the custom converter templates. If a template engine is not specified, it will be auto-detected based on the file extension of the custom converter templates found.

### **-e, --eruby**

Specifies the eRuby implementation to use for executing the custom converter templates written in ERB. Supported values are *erb* and *erubis*. Defaults to *erb*.

### **-I, --load-path=DIRECTORY**

Add the specified directory to the load path, so that *-r* can load extensions from outside the default Ruby load path. This option may be specified more than once.

### **-n, --section-numbers**

Auto-number section titles. Synonym for **--attribute sectnums**.

### **-o, --out-file=OUT\_FILE**

Write output to file *OUT\_FILE*. Defaults to the base name of the input file suffixed with *backend* extension. The file is resolved relative to the working directory. If the input is read from standard input or a named pipe (fifo), then the output file defaults to stdout. If *OUT\_FILE* is *-*, then the output file is written to standard output.

### **-R, --source-dir=DIR**

Source directory. Currently only used if the destination directory is also specified. Used to preserve the directory structure of files converted within this directory in the destination directory. If specified, the directory is resolved relative to the working directory.

### **-r, --require=LIBRARY**

Require the specified library before executing the processor, using the standard Ruby require. This option may be specified more than once.

### **-s, --no-header-footer**

Output an embeddable document, which excludes the header, the footer, and everything outside the body of the document. This option is useful for producing documents that can be inserted into an external template.

### **-T, --template-dir=DIR**

A directory containing custom converter templates that override one or more templates from the built-in set. (requires *tilt* gem)

If there is a subfolder that matches the engine name (if specified), that folder is appended to the template directory path. Similarly, if there is a subfolder in the resulting template directory that matches the name of the backend, that folder is appended to the template directory path.

This option may be specified more than once. Matching templates found in subsequent directories override ones previously discovered.

## B.4. Processing Information

### **--failure-level=LEVEL**

The minimum logging level that triggers a non-zero exit code (failure). If this option is not set (default: FATAL), the program exits with a status code zero even if warnings or errors have been logged.

### **-q, --quiet**

Silence warnings.

### **--trace**

Include backtrace information on errors. Not enabled by default.

### **-v, --verbose**

Verbosely print processing information and configuration file checks to stderr.

### **-t, --timings**

Display timings information (time to read, parse and convert).

## B.5. Program Information

### **-h, --help [TOPIC]**

Print the help message. Show the command usage if *TOPIC* is not specified (or not recognized). Dump the Asciidoctor man page (in troff/groff format) if *TOPIC* is *manpage*.

### **-V, --version**

Print program version number.

**-v** can also be used if no other flags or arguments are present.

# Appendix C: Ruby API Options

## Ruby API options

Name	Description	Default value	Allowed values	CLI equivalent
:attributes	Sets additional document attributes, which override equivalently-named attributes defined in the document unless the name or value ends with <code>@</code> . When using the Hash form, a <code>nil</code> value unsets the attribute and a <code>false</code> value soft unsets the attribute.	<i>not set</i>	Any number of <a href="#">built-in</a> or user-defined attributes in one of the following formats:  Hash: <code>{'name'=&gt;'value'}</code> Array: <code>['name=value']</code> String: <code>'name=value'</code>	<code>-a, --attributes</code>
:backend	Selects the converter to use (as registered with this keyword).	html5	html5, docbook5, docbook45, manpage, or any backend registered through an active converter	<code>-b, --backend</code>
:base_dir	Sets the base (aka working) directory containing the document and resources.	The directory of the source file, or the working directory if the source is read from a stream.	file path	<code>-B, --base-dir</code>

Name	Description	Default value	Allowed values	CLI equivalent
<code>:catalog_assets</code>	If true, tells the parser to capture images and links in the reference table. (Normally only IDs, footnotes and indexterms are included). The reference table is available via the <code>references</code> property on the <code>document</code> AST object. <i>(Experimental)</i> .	false	<i>boolean</i>	<i>n/a</i>
<code>:converter</code>	Specifies a user-supplied <code>Asciidoctor::Converter</code> instance, used in place of the converter that is automatically resolved from the backend value.	<i>not set</i>	<code>Asciidoctor::Converter</code> instance	<i>n/a (Can be emulated using a combination of <code>-r</code> and <code>-b</code>).</i>
<code>:doctype</code>	Sets the document type.	article	article, book, manpage, inline	<code>-d, --doctype</code>
<code>:eruby</code>	Specifies the eRuby implementation to use for executing the converter templates written in ERB.	erb	erb, erubis	<code>-e, --eruby</code>
<code>:extensions</code>	A Ruby block that registers (and possibly defines) Asciidoctor extensions for this instance of the processor.	<i>not set</i>	A Ruby block that conforms to the Asciidoctor extensions API (the same code that would be passed to the <code>Extensions.register</code> method).	<i>n/a (Can be emulated using <code>-r</code>).</i>

Name	Description	Default value	Allowed values	CLI equivalent
<code>:extensions_registry</code>	Overrides the extensions registry instance. Instead of providing a Ruby block containing extensions to register, this option lets you replace the extension registry itself, giving you complete control over how extensions are registered for this processor.	<i>not set</i>	Extensions::Registry instance	<i>n/a</i>
<code>:header_footer</code>	If true, add the document header and footer (i.e., framing) around the body content in the output. NOTE: The default value for this option is opposite of the default value for the CLI.	false	<i>boolean</i>	<code>-s, --no-header-footer</code>
<code>:mkdirs</code>	If true, the processor will create the necessary output directories if they don't yet exist.	false	<i>boolean</i>	<i>n/a (Implicitly enabled).</i>
<code>:parse</code>	If true, the source is parsed eagerly (i.e., as soon as the source is passed to the <code>load</code> or <code>load_file</code> API). If false, parsing is deferred until the <code>parse</code> method is explicitly invoked.	true	<i>boolean</i>	<i>n/a</i>

Name	Description	Default value	Allowed values	CLI equivalent
<code>:safe</code>	Sets the safe mode.	<code>:secure</code>	<code>:unsafe</code> , <code>:safe</code> , <code>:server</code> , <code>:secure</code>	<code>-S, --safe-mode</code>
<code>:sourcemap</code>	Keeps track of the file and line number for each parsed block. (Useful for tooling applications where the association between the converted output and the source file is important).	false	<i>boolean</i>	<i>n/a</i>
<code>:template_cache</code>	If true, enables the built-in cache used by the template converter when reading the source of template files. Only relevant if the <code>:template_dirs</code> option is specified.	true	<i>boolean</i>	<i>n/a</i>
<code>:template_dir</code>	Specifies a directory of Tilt-compatible templates to be used instead of the default built-in templates. <b>Deprecated.</b> <b>Please use <code>:template_dirs</code> instead.</b>	<i>not set</i>	file path	<code>-T, --template-dir</code>
<code>:template_dirs</code>	An array of directories containing Tilt-compatible converter templates to be used instead of the default built-in templates.	<i>not set</i>	An array of file paths	<code>-T, --template-dir</code>

Name	Description	Default value	Allowed values	CLI equivalent
<code>:template_engine</code>	Template engine to use for the custom converter templates. The gem with the same name as the engine will be loaded automatically. This name is also used to build the full path to the custom converter templates.	<i>auto</i> (Set based on the file extension of the custom converter templates found).	Template engine name (e.g., slim, haml, erb, etc.)	<code>-E, --template-engine</code>
<code>:template_engine_options</code>	Low-level options passed directly to the template engine.	<i>not set</i>	A nested Hash of options with the template engine name as the top-level key and the option name as the second-level key.	<i>n/a</i>
<code>:timings</code>	Capture time taken to read, parse, and convert document. <b>Internal use only.</b>	<i>not set</i>	<code>Asciidoctor::Timings</code> instance	<code>-t, --timings</code>
<code>:to_file</code>	The name of the output file to write, or true to use the default output file ( <code>docname + outfilesuffix</code> ).	<i>not set</i>	true, file path	<code>-o, --out-file</code>
<code>:to_dir</code>	Destination directory for output file(s), relative to <code>base_dir</code> .	The directory containing the source file, or the working directory if the source is read from a stream.	file path	<code>-D, --destination-dir</code>

# Appendix D: Application Messages

All warning, error, and failure messages emitted by the Asciidoctor CLI (i.e., `asciidoc`) are listed in the table below. These messages are written to the console via stderr (i.e., standard error). Most messages also apply to the API, though keep in mind they're written directly to stderr.

## *Placeholders*

### `<docname>`

represents the basename of the source file being processed (e.g., `sample.adoc`).

### `<file>`

represents a path to the input file or other referenced file.

### `<uri>`

represents a URI being referenced.

### `<x> or <y>`

placeholders for other contextual information in the message.

## *Categories*

### **ERROR**

Errors do not stop conversion, but the output document will almost certainly be wrong.

### **FAILURE**

Failures are fatal; no output document will be produced.

### **WARNING**

Warnings do not stop conversion, but they indicate possible problems, and the output may not be what you were expecting.

## *List of Application Messages*

Category	Message	Cause	See also
ERROR	input file <code>&lt;file&gt;</code> missing or cannot be read	Check that the file exists and that the filename is not misspelled.	<a href="#">Using the Command Line Interface</a>
ERROR	include file has illegal reference to ancestor of jail, auto-recovering	The safe mode is restricting access to an include file outside of the base directory.	<a href="#">Running Asciidoctor Securely</a>
ERROR	input file and output file cannot be the same: <code>&lt;file&gt;</code>	Choose a different output directory or filename.	
ERROR	partintro block can only be used when doctype is book and it's a child of a part section. Excluding block content.	Invalid book document structure.	<a href="#">Book Parts and Chapters</a>

Category	Message	Cause	See also
ERROR	unmatched macro: endif::<x>[]	endif::[] with no unclosed preceding ifdef::<x>[].	<a href="#">ifdef Directive</a>
ERROR	<docname> dropping cell because it exceeds specified number of columns		<a href="#">Tables</a>
ERROR	<docname> illegal block content outside of partintro block	Invalid book document structure.	<a href="#">Book Parts and Chapters</a>
ERROR	<docname> invalid part, must have at least one section (e.g., chapter, appendix, etc.)	Invalid book document structure.	<a href="#">Book Parts and Chapters</a>
ERROR	<docname> malformed manpage title	Invalid man page document structure.	<a href="#">Man Pages</a>
ERROR	<docname> malformed name section body	Invalid man page document structure.	<a href="#">Man Pages</a>
ERROR	<docname> maximum include depth of 64 exceeded	Does your file include itself, directly or indirectly?	
ERROR	<docname> mismatched macro: endif::<x>[], expected endif::<y>[]	ifdef/endif blocks must be strictly nested.	<a href="#">ifdef Directive</a>
ERROR	<docname> name section expected	Invalid man page document structure.	<a href="#">Man Pages</a>
ERROR	<docname> name section title must be at level 1	Invalid man page document structure.	<a href="#">Man Pages</a>
ERROR	<docname> only book doctypes can contain level 0 sections	Illegal use of a level-0 section when doctype is not book.	<a href="#">Sections</a>
ERROR	<docname> table missing leading separator, recovering automatically	Check for missing cell separator characters at the start of the line.	<a href="#">Tables</a>
FAILED	missing converter for backend '<x>'. Processing aborted. (RuntimeError)	You used -b with an invalid or missing backend.	
FAILED	'tilt' could not be loaded	You must have the tilt gem installed ( <code>gem install tilt</code> ) to use custom backend templates	
WARNING	abstract block cannot be used in a document without a title when doctype is book. Excluding block content.	Invalid book document structure.	<a href="#">Abstract</a>

Category	Message	Cause	See also
WARNING	cannot retrieve contents of <x> at URI: <uri> (allow-uri-read attribute not enabled)	Reading from a URI is only allowed in certain safe modes and the allow-uri-read attribute must be passed to the application.	<a href="#">Include Content from a URI</a>
WARNING	could not retrieve contents of <x> at URI: <uri>	Web address not found.	
WARNING	could not retrieve image data from URI: <uri>	Web address not found. Only occurs with <code>allow-uri-read</code> and <code>data-uri</code> . Check the URI.	<a href="#">Include Images by Full URL</a>
WARNING	dropping line containing reference to missing attribute: <x>	An attribute cannot be resolved and the <code>attribute-missing</code> attribute is set to <code>drop-line</code> .	<a href="#">[catch-a-missing-or-undefined-attribute]</a>
WARNING	file does not exist or cannot be read: <file>	You specified a stylesheet ( <code>-a stylesheet=&lt;file&gt;</code> ) but <file> does not exist or is not readable.	
WARNING	gem 'thread_safe' is not installed. This gem is recommended when registering custom converters.	You have registered a custom converter, and you have not installed the thread_safe gem.	
WARNING	gem 'thread_safe' is not installed. This gem is recommended when using custom backend templates.	You are using custom templates ( <code>-T &lt;template_dir&gt;</code> ), but you have not installed the thread_safe gem.	
WARNING	image to embed not found or not readable: <file>	You used <code>:data-uri:</code> but the file could not be found.	<a href="#">Managing Images</a>
WARNING	include file not readable: <file>	You do not have permission to access the file.	<a href="#">AsciiDoc vs non-AsciiDoc files</a>
WARNING	input path <file> is a <x>, not a file	The path is not a file (perhaps it is a socket or a block device).	<a href="#">AsciiDoc vs non-AsciiDoc files</a>
WARNING	optional gem 'asciimath' is not installed. Functionality disabled.	asciimath is one of the libraries used for equations.	<a href="#">Equations and Formulas</a>
WARNING	optional gem 'coderay' is not installed. Functionality disabled.	CodeRay is used for source code highlighting.	<a href="#">CodeRay</a>
WARNING	skipping reference to missing attribute: <x>	An attribute cannot be resolved and the <code>attribute-missing</code> attribute is set to <code>skip</code> .	<a href="#">[catch-a-missing-or-undefined-attribute]</a>
WARNING	tables must have at least one body row		<a href="#">Tables</a>

Category	Message	Cause	See also
WARNING	tag '<x>' not found in include file: <file>	You tried to include by tagged region, but the included document does not have that tag.	<a href="#">Select Portions of a Document to Include</a>
WARNING	<docname>: id assigned to <type> already in use: <id>	<id> is a duplicate ID, meaning it has already been assigned to a node of <type> (e.g., section, block, anchor). If you don't see the problem in <docname>, check that the duplicate ID isn't coming from a file which is being included.	
WARNING	<docname> callout list item index: expected <x> got <y>	Callouts are expected to be in numerical order, just like any ordered list.	<a href="#">Callouts</a>
WARNING	<docname> include <x> not readable: <y>	If <y> is a file, do you have read permissions for it? If it is a URI and <code>-a allow-uri-read</code> is set, does it exist?	
WARNING	<docname> include file not found: <file>	Probably a typo or missing file. If not, make sure you understand the search process.	<a href="#">Include Directive, File resolution</a>
WARNING	<docname> invalid empty <x> detected in style attribute	The first positional attribute in the block attributes could not be parsed.	<a href="#">Options</a>
WARNING	<docname> invalid style for <x> block: <y>	You have added a custom style to a block, but you haven't registered a custom block extension to handle it.	
WARNING	<docname> invalid style for paragraph: <x>	You have a line [xxx] before a paragraph, but xxx isn't one of the built-in styles.	<a href="#">Style</a>
WARNING	<docname> list item index: expected <x>, got <y>	You gave explicit numbers on an ordered list, but they were not sequential. Asciidoctor renames them for you, and gives this warning.	<a href="#">Ordered Lists</a>
WARNING	<docname> multiple ids detected in style attribute	Multiple IDs cannot be specified in the block style (e.g., [#cat#dog]).	<a href="#">Id</a>

Category	Message	Cause	See also
WARNING	<docname> no callouts refer to list item <x>	The callout is missing or not recognized. In source listings, is the callout the last thing on the line?	<a href="#">Callouts</a>
WARNING	<docname> section title out of sequence	Invalid document structure. Check section levels.	<a href="#">Sections</a>