AsciiDoc

Version pre-spec, 2025-10-27

Table of Contents

About AsciiDoc

AsciiDoc is a lightweight, semantic markup language primarily designed for writing technical documentation. The language can be used to produce a variety of presentation-rich output formats, all from content encoded in a concise, human-readable, plain text format.

The AsciiDoc syntax is intuitive because it builds on well-established, plain text conventions for marking up and structuring text. Someone unfamiliar with AsciiDoc can probably guess the purpose of many of its syntax elements just by looking at them. That's because the elements of the syntax were carefully chosen to look like what they mean (a practice long employed by the tech industry).

The AsciiDoc language isn't coupled to the output format it produces. An AsciiDoc processor can parse and comprehend an AsciiDoc source document and convert the parsed document structure into one or more output formats, such as HTML, PDF, EPUB3, man(ual) page, or DocBook. The ability to produce multiple output formats is one of the main advantages of AsciiDoc. This capability enables it to be used in static site generators, IDEs, git tools and services, CI/CD systems, and other software.

AsciiDoc bridges the gap between ease of writing and the rigorous requirements of technical authoring and publishing. AsciiDoc only requires a text editor to read or write, thereby offering a low bar to getting started.

About this documentation

You're reading the user-facing documentation for the AsciiDoc language as it's implemented in Asciidoctor. This documentation does not cover how to set up and use Asciidoctor to process AsciiDoc content. You can find that documentation in the Asciidoctor section of this website.

This documentation has been submitted as the initial contribution for the AsciiDoc Language project at Eclipse. That project will use this documentation as the basis for drafting a specification for the AsciiDoc language. It will also be used as the draft of the user-facing guide for the AsciiDoc Language, which will also be maintained by that project.

Until the first version of the AsciiDoc Language Specification is ratified, AsciiDoc is defined by the Asciidoctor implementation. There is no other official definition of the language.

The documentation for AsciiDoc will remain on this site until the AsciiDoc Language project starts publishing its own documentation for the AsciiDoc Language.

Until then, let's get started!

Introduction

Document Structure

On this page, you'll learn about the overall structure of an AsciiDoc document. Don't worry about the details of the syntax at this point. That topic will be covered thoroughly later in the documentation. Right now, we're just aiming to get a sense of what makes up an AsciiDoc document.

Documents

AsciiDoc is a plain text writing format with no boilerplate enclosure or prologue. An AsciiDoc document may consist of only a single sentence (or even a single character, to be academic).

The example below is a valid AsciiDoc document. It contains a single paragraph that consists of a single sentence.

```
This is a basic AsciiDoc document.
```

Of course, you can have more content than a single sentence! What we want to emphasize here is that it's simple to get started.

An AsciiDoc document is a series of blocks stacked are stacked linewise. These blocks are typically offset from one another by an empty line. (While these empty lines aren't always required, we do recommend using them for readability.)

To expand the previous document from one paragraph to two, you'd separate the two paragraphs by an empty line:

```
This is a basic AsciiDoc document.

This document contains two paragraphs.
```

An AsciiDoc document may begin with a document header. Although the document header is optional, it's often used because it allows you to specify a document title as well as document-wide configuration and reusable text in the form of document attributes.

```
= Document Title
Author Name
:reproducible:

This is a basic AsciiDoc document by {author}.

This document contains two paragraphs.
It also has a header that specifies the document title and some attibutes.
```

Almost any combination of blocks constitutes a valid AsciiDoc document (with some structural requirements dictated by the document type). Documents can range from a single sentence to a multi-part book.

Lines

The line is a significant building block in AsciiDoc. A line is defined as text that's separated on either side by either a newline character or the boundary of the document. Many aspects of the syntax must occupy a whole line. That's why we say AsciiDoc is a line-oriented language.

For example, a section title must be on a line by itself. The same is true for an attribute entry, a block title, a block attribute list, a block macro, a list item, a block delimiter, and so forth.

Example 1. Example of a section title, which must occupy a single line

```
== Section Title
```

Example 2. Example of an attribute entry, which must also occupy at least one line

```
:name: value
```

Example 3. Example of an attribute entry that extends to two lines

```
:name: value \
more value
```

Empty lines can also be significant. For example, a single empty line separates the header from the body. Many blocks are also separated by an empty line, as you saw in the example earlier with two paragraphs.

In contrast, lines within paragraph content are insignificant. Keep these points in mind as you're learning about the AsciiDoc syntax.

Blocks

Blocks in an AsciiDoc document lay down the document structure. Some blocks may contain other blocks, so the document structure is inherently hierarchical (i.e., a tree structure). You can inspect this section structure, for example, by enabling the automatic table of contents. Examples of blocks include paragraphs, sections, lists, delimited blocks, tables, and block macros.

Blocks are easy to identify because they're usually offset from other blocks by an empty line (though not always required). Blocks always start on a new line, terminate at the end of a line, and are aligned to the left margin.

Every block can have one or more lines of block metadata. This metadata can be in the form of block attributes, a block anchor, or a block title. These metadata lines must be above and directly adjacent to the block itself.

Sections, non-verbatim delimited blocks, and AsciiDoc table cells may contain other blocks. Despite the fact that blocks form a hierarchy, even nested blocks start at the left margin. By requiring blocks to start at the left margin, it avoids the tedium of having to track and maintain levels of indentation. It also happens to make the content more reusable.

Text and inline elements

Surrounded by the markers, delimiters, and metadata lines is the text. The text is the main focus of a document and the reason the AsciiDoc syntax gives it so much room to breathe. Text is most often found in the lines of a block (e.g., paragraph), the block title (e.g., section title), and in list items, though there are other places where it can exist.

Text is subject to substitutions. Substitutions interpret markup as text formatting, replace macros with text or non-text elements, expand attribute references, and perform other sorts of text replacement.

Normal text is subject to all substitutions, unless specified otherwise. Verbatim text is subject to a minimal set of substitutions to allow it to be displayed in the output as it appears in the source. It's also possible to disable all substitutions in order to pass the text through to the output unmodified (i.e., raw). The parsing of text ends up being a mix of inline elements and other forms of transformations.

Encodings and AsciiDoc files

An AsciiDoc file is a text file that has the *.adoc* file extension (e.g., *document.adoc*). Most AsciiDoc processors assume the text in the file uses UTF-8 encoding. UTF-16 encodings are supported only if the file starts with a BOM.

An AsciiDoc processor can process AsciiDoc from a string (i.e., character sequence). However, most of the time you'll save your AsciiDoc documents to a file.

Key Concepts

This page introduces you to some of the concepts and terms you'll encounter as you learn about AsciiDoc. Each concept will be covered in more depth later in the documentation. Use this page as a way to start to familiarize yourself with the lingo.

Document

A document represents the top-level block element in AsciiDoc. It consists of an optional document header and either a) one or more sections preceded by an optional preamble or b) a sequence of top-level blocks only.

The document can be configured using a document header. The header is not a block itself, but contributes metadata to the document, such as the document title and document attributes.

Elements

An element is an identifiable, addressable, and composable chunk of content in a document. An

AsciiDoc document is merely a composition of all the elements it contains.

Elements are a hierarchy of types, where one element may be a specialization of a family of elements. For example, a sidebar block is a block element, so it shares the traits of all block elements, and also adds some of its own.

Elements include the document itself, sections, blocks, block macros, breaks, and inline phrases and macros.

A **block element** is stacked vertically (by line) above or below other block elements. Block elements are typically referred to simply as **blocks**. Blocks form the main tree structure of the document.

An **inline element** is a span of content within a block element or one of its attributes (e.g., a block title). Inline elements include formatted text (italic, bold, etc), inline macros, and element references. What fills in the gap between these elements is unsubstituted text. Inline elements are less structured than block elements as they are more geared towards substitutions than a tree structure.

Attributes

An attribute is a name/value pair used for storing and disclosing metadata in the AsciiDoc language. Attributes can be used to influence the syntax, control behavior, customize styles, activate or configure integrations, or store inline replacement content. Attributes truly set AsciiDoc apart from other lightweight markup languages.

An attribute is actually an abstract term. There are two concrete classifications of attributes: document attributes and element attributes.

Document attributes

Document attributes, as the name implies, are associated directly with the document. They are used to export information about the document at runtime, control behavior of the processor, and to store reusable values or phrases. Thus, they are a sort of two-way communication channel with the processor.

Document attributes can be referenced in the content using an attribute reference (wherever the attribute substitution is enabled). A document attribute can be defined either in the document using an attribute entry (typically in the document header) or from the API or CLI. Not all document attributes can be modified.

Element attributes

Element attributes are metadata on a specific element, like a block or an inline element. They are defined in an attribute list and only apply to that element. The placement of the attribute list depends on the element. The attribute name can either be a string (i.e., a named attribute) or an implicit numerical index (i.e., an unnamed, positional attribute).

Unlike document attributes, element attributes cannot be referenced directly from the content, on the document model. In other words, they cannot be resolved using an attribute reference. Element attributes enrich or configure the behavior of an element, such as to apply a role or set the width of an image. An element attribute is defined using an attribute list on an element, or an available

shorthand like a block title line.

Macros

As you read through this documentation, you'll frequently see references to the term macro. A macro is a syntax for representing non-text elements or syntax that expands into text using the provided metadata. See macro to learn more about the meaning of this term.

Here's an example of a block macro:

```
image::sunset.jpg[Sunset]
```

Here's an example of an inline macro:

```
Click the button with the image:star.png[Star] to favorite the project.
```

You can think of a macro like a function. A syntax of macro follows the form of a name, a target which is sometimes optional, and an attribute list consisting of zero or more element attributes enclosed in square brackets.

There are two variations of a macro: block and inline. In a block macro, the name and target are separated by two colons (`::`) and it must reside on a line by itself. In an inline macro, the name and target are separated by a single colon (`:`) and it can be alongside text and other inline elements. A block macro is always parsed, whereas an inline macro is only parsed where the macros substitution is enabled.

Preprocessor directives

There's another syntax in AsciiDoc that looks a lot like block macros, only they aren't. These are the preprocessor directives.

A preprocessor directive is a function that controls lines that are fed into the parser. A conditional preprocessor directive can configure lines to be included or excluded based on the presence of an attribute (ifdef, ifndef) or another arbitrary condition (ifeval). An include directive can add additional lines to the document taken from another document.

Preprocessor directives share common traits with a block macro. Like a block macro, a preprocessor directive must be on a line by itself. While the preprocessor directive can access document attributes, it's not otherwise aware of the context around it. It's only a line processor. Like a block macro, the include directive can have element attributes, though they only apply to the preprocessing operation itself.

Document Processing

AsciiDoc is specifically a writing format, not a publishing format. In other words, it's not WYSIWYG like when you write in a word processor. Instead, what you write is the AsciiDoc source. You then use an AsciiDoc processor, such as Asciidoctor, to convert the AsciiDoc source into a publishable

format. It's this output that you publish.

Converting the AsciiDoc source is an opportunity to interpret and embellish your content to get more out of it than what you put in. The work of converting the AsciiDoc source to another format is handled by a converter. While there is a strong relationship between the language and the converters, these two aspects are not explicitly coupled.

An AsciiDoc processor provides several built-in converters, including ones for making HTML and DocBook. To activate one of these converters, you set the backend on the document (default: html). The backend is a keyword that tells the processor which output format you want to make. The processor then selects the converter that makes that output format. For example, the HTML converter handles the html backend to make HTML output.

An AsciiDoc processor actually works in two steps. First, it parses the AsciiDoc document. This parsing produces a structured document that reflects the written structure and interprets all the meaningful markup. The processor then passes this structured document to the converter to transform it into the output format.

In short, the processor accepts a string (which may be read from a file), parses it into a structure document, then produces another string (which may be written to a file).

Normalization

When an AsciiDoc processor reads the AsciiDoc source, the first thing it does is normalize the lines. (This operation can be performed up front or as each line is visited).

Normalization consists of the following operations:

- Force the encoding to UTF-8 (An AsciiDoc processor always assumes the content is UTF-8 encoded)
- Strip trailing spaces from each line (including any end of line character)

This normalization is performed independent of any structured context. It doesn't matter if the line is part of a literal block or a regular paragraph. All lines get normalized.

Normalization is only applied in certain cases to the lines of an include file. Only include files that have a recognized AsciiDoc extension are normalized as described above. For all other files, only the trailing end of line character is removed. Include files can also have a different encoding, which is specified using the encoding attribute. If the encoding attribute is not specified, UTF-8 is assumed.

When the AsciiDoc processor brings the lines back together to produce the rendered document (HTML, DocBook, etc), it joins the lines on the line feed character (`\n`).

Blocks

Block elements form the main structure of an AsciiDoc document, starting with the document itself.

What is a block?

A block element (aka block) is a discrete, line-oriented chunk of content in an AsciiDoc document. Once parsed, that chunk of content becomes a block element in the parsed document model. Certain blocks may contain other blocks, so we say that blocks can be nested. The converter visits each block in turn, in document order, converting it to a corresponding chunk of output.

Block forms

How the boundaries of a block are defined in the AsciiDoc syntax varies. The boundaries of some blocks, like lists, paragraphs, and block macro, are implicit. Other blocks have boundaries that are explicitly marked using delimiters (i.e., delimited blocks). The main commonality is that a block is always line-oriented.

A *paragraph block* is defined as a discrete set of contiguous (non-empty) lines. A *delimited block* is bounded by delimiter lines. A *section block* (aka section) is defined by a section title that's prefixed by one or more equal signs. The section includes all content that follows the section title line until the next sibling or parent section title or the document boundary. A *list block* is defined by a group of sibling list items, each denoted by a marker. A *description list* block is defined by a sibling group of list items, each denoted by one or more terms. A *block macro* is defined by a single line that matches the block macro syntax. And the *document* is also a block.

A block (including its metadata lines) should always be bounded by an empty line or document boundary on either side.

Whether or not a block supports nested blocks depends on content model of the block (and what the syntax allows).

Content model

The content model of a block determines what kind of content the block can have (if any) and how that content is processed. The content models of blocks in AsciiDoc are as follows:

compound
: a block that may only contain other blocks (e.g., a section)

simple
: a block that's treated as contiguous lines of paragraph text (and subject to normal substitutions) (e.g., a paragraph block)

verbatim
: a block that holds verbatim text (displayed "as is") (and subject to verbatim substitutions) (e.g., a listing block)

raw

a block that holds unprocessed content passed directly through to the output with no substitutions applied (e.g., a passthrough block)

empty

a block that has no content (e.g., an image block)

table

a special content model reserved for tables that enforces a fixed structure

The content model is inferred for all built-in syntax (as determined by the context), but can be configured for custom blocks. Blocks may also support different content models under different circumstances. The circumstance is determined by the context and style, and in the case of a delimited block, the structural container as well.

Context

You may often hear a block referred to by a name, such as an example block, a sidebar block, an admonition block, or a section. That name is the block's context.

Let's consider the following normal section:

```
== Section Title

Content of section.
```

The context of this block is `section`. We often refer to this as a section (or section block), using the context as an adjective to describe the block. The writer does not have to specify the context in this case since it's implied by the syntax.

Every block has a context. The context is often implied by the syntax, but can be declared explicitly in certain cases. The context is what distinguishes one kind of block from another. You can think of the context as the block's type.

The context can be further modified using a block style to create a family of blocks that share a common type, as is the case with admonition blocks and sections. We'll cover that modifier shortly.

For blocks, the context is sometimes referred to as the block name. This comes up in particular when talking about custom blocks. The block name is just another layer of abstraction. All the built-in block names map to exactly one context. But a block extension can map an arbitrary block name to one or more contexts. Which context is ultimately used depends on what is returned from the extension's process method. In the end, it's the context that determines how the block is converted.

The context often determines the content model. For example, all sections implicitly have the compound content model because a section may only contain other blocks. All literal blocks implicitly have the verbatim content model because the purpose of this block is to present verbatim output.

Summary of built-in contexts

Here's a list of the contexts of all the built-in blocks in AsciiDoc.

In the Asciidoctor API, the contexts are represented as symbols. In Ruby, a symbol is a name prefixed with a colon (e.g., `:listing`). This documentation will sometimes use this notation when referring to the name of a context. However, this notation is not universal. Some processors, such as Asciidoctor.js, store the context as a string instead.

Built-in contexts

Name	Purpose
admonition	One of five admonition blocks.
audio	An audio block.
colist	A callout list.
dlist	A description list.
document	The top-level document or the document in an AsciiDoc table cell
example	An example block.
floating_title	A discrete heading.
image	An image block.
list_item	An item in an ordered, unordered, or description list (only relevant inside a list or description list block). In a description list, this block is used to represent the term and the description.
listing	A listing block.
literal	A literal block.
olist	An ordered list.
open	An open block.
page_break	A page break.
paragraph	A paragraph.
pass	A passthrough block.
preamble	The preamble of the document.
quote	A quote block (aka blockquote).
section	A section. May also be a part, chapter, or special section.
sidebar	A sidebar block.
table	A table block.
table_cell	A table cell (only relevant inside a table block).
thematic_break	A thematic break (aka horizontal rule).
toc	A TOC block (to designate custom TOC placement).

Name	Purpose
ulist	An unordered list.
verse	A verse block.
video	A video block.

Each inline element also has a context, but those elements are not (yet) accessible from the parsed document model.

Additional contexts may be introduced through the use of the block, block macro, or inline macro extension points.

Contexts used by the converter

The context is what the converter uses to dispatch to a convert method. The style is then used by the converter to apply special behavior to blocks of the same family.

With two exceptions, there's a 1-to-1 mapping between the contexts and the handler methods of a converter. Those exceptions are the `list_item` and `table_cell` contexts, which are not mapped to a handler method. In the converter, these blocks must be accessed from their parent block.

Block style

The context does not always tell the whole story of a block's identity. Some blocks require specialization. That's where the block style comes into play.

Above some blocks, you may notice a name at the start of the block attribute list (e.g., `[source]` or `[verse]`). The first positional (unnamed) attribute in the block attribute list is used to declare the block style.

The declared block style is the value the author supplies. That value is then interpreted and resolved. The resolved block style, if non-empty, specializes the block's context. (It may instead, or in addition to, alter the block's context).

Consider the following example of a source block:

```
[source,ruby]
----
puts "Hello, World!"
----
```

The context of a source block is `listing` (as inferred from the block delimiters) and the style is `source` (as specified by the writer). We say that the style specializes the block as a source block. (Technically, the presence of a source language already implies the `source` style, but under the covers this is what's happening). The context of the block is still the same, but it has additional metadata to indicate that it requires special processing.

We also see the block style used for other purposes. The `appendix` block style (e.g., `[appendix]`) above the section title specializes the section as an appendix (a special section) and thus has special semantics and behavior. In the model, the section's style is dually stored as the `sectname`. One of the five admonition styles (e.g., `[TIP]`) above an example block transforms the example block into an admonition block with that name (i.e., label). In the model, the admonition style in lowercase is stored in the `name` attribute. A block style (e.g., `[circle]` or `[loweralpha]`) above an unordered or ordered list, respectively, alters the marker used in front of list items when displayed. A block style (e.g., `[qanda]` and `[horizontal]`) above a description list can either change its semantics or layout.

The declared block style can be used to change the context of a block, referred to as block masquerading. Consider the case of this alternate syntax for a listing block using the literal block delimiters.

```
[listing]
....
a > b
....
```

Since the declared block style matches the name of a context, the context of the block becomes `listing` and the resolved block style remains unset. That means the resolved block style differs from the declared block style. To learn more about how to change the context of a block using the declared block style, see Block Masquerading.

To get a complete picture of a block's identity, you must consider both the context and the style. The resolved style specializes the context to give it special behavior or semantics.

Block commonalities

Blocks are defined using some form of line-oriented syntax. Section blocks begin with a section title line. Delimited blocks are enclosed in a matching pair of delimiter lines. Paragraph blocks must be contiguous lines.

All blocks accommodate zero or more lines of metadata stacked linewise directly on top of the block. These lines populate the properties of the block, such as the ID, title, and options. These metadata lines are as follows:

- Zero or more block attribute lines (which populate the block's attributes)
- An optional block anchor line
- An optional block title line (many blocks also support a corresponding caption)
- An optional ID
- An optional set of roles
- An optional set of options

If the first line of a block begins with `[` and ends with `]`, that line will be interpreted as a block attribute line. It does not matter what text is contained between those brackets. For example, if the first description list term starts with `[` and defi-

> nition on the same line ends with], it will not appear to the parser as a description list entry, but rather as a block attribute line. To workaround this interpretation of the source, you need to move the trailing] (and whatever goes with it) to the next line.

For example, consider a sidebar block with a title and ID:

```
.Styles of music
[#music-styles]
****
Go off on a tangent to describe what a style of music is.
****
```

When it comes to processing content, blocks split off into different groups. These groups are primarily associated with the block's content model.

Paragraph blocks and verbatim blocks have an implicit and modifiable set of substitutions. Substitutions do not apply to compound blocks (i.e., blocks that may contain nested blocks).

Delimited Blocks

In AsciiDoc, a delimited block is a region of content that's bounded on either side by a pair of congruent linewise delimiters. A delimited block is used either to enclose other blocks (e.g.., multiple paragraphs) or set the content model of the content (e.g., verbatim). Delimited blocks are a subset of all block types in AsciiDoc.

Overview

Delimited blocks are defined using structural containers, which are the fixed set of recognized block enclosures in the AsciiDoc syntax. Here's the structural container for a literal block:

```
....
This text will be treated as verbatim content.
....
```

A structural container has an opening delimiter and a closing delimiter. The opening delimiter follows the block metadata, if present. Leading and trailing empty lines in a structure container are not considered significant and are automatically removed. The remaining lines define a block's content.

These enclosures not only define the boundaries of a block's content, but also imply a content model (e.g., verbatim content or a subtree). In certain cases, they provide a mechanism for nesting blocks. However, delimited blocks cannot be interleaved.

A delimited block has the unique ability of being able to be repurposed by the AsciiDoc syntax, through both built-in mappings and mappings for custom blocks defined by an extension. To understand how delimited blocks work, it's important to understand structural containers, their linewise

delimiters, their default contexts, and their expected content models, as well as block nesting and masquerading.

Linewise delimiters

A delimited block is characterized by a pair of congruent linewise delimiters. The opening and closing delimiter must match exactly, both in length and in sequence of characters. These delimiters, sometimes referred to as fences, enclose the content and explicitly mark its boundaries. Within the boundaries of a delimited block, you can enter any content or empty lines. The block doesn't end until the ending delimiter is found. The block metadata (block attribute and anchor lines) goes above the opening delimiter (thus outside the delimited region).

Here's an example of a delimited example block:

```
====
This is an example of an example block.
That's so meta.
====
```

Typically, the delimiter is written using the minimum allowable length (4 characters, with the exception of the open block, which currently has a fixed length of 2 characters). The length of the delimiter lines can be varied to accommodate nested blocks.

The valid set of delimiters for defining a delimited block, and the meaning they have, is defined by the available structural containers, covered next.

Structural containers

Structural containers are the fixed set of recognized block enclosures (delimited regions) defined by the AsciiDoc language. These enclosures provide a reusable building block in the AsciiDoc syntax. By evaluating the structural container and the block metadata, the processor will determine which kind of block to make.

Each structural container has an expected content model. For built-in blocks, it's the context of the block that determines the content model, though most built-in blocks adhere to the expected content model. Custom blocks have the ability to designate the content model. Even in these cases, though, the content model should be chosen to honor the semantics of the structural container.

Some structural containers are reused for different purposes, such as the structural container for the quote block being used for a verse block.

Summary of structural containers

The table below lists the structural containers, documenting the name, default context, and delimiter line for each.

Structural containers in AsciiDoc

Type	Default context	Content model (expected)	Minimum delimiter
comment	`n/a`	*n/a*	`////`
example	`:example`	compound	`====`
listing	`:listing`	verbatim	`----`
literal	`:literal`	verbatim	`....`
open	`:open`	compound	`--`
sidebar	`:sidebar`	compound	`****`
table	`:table`	table	`|===` `,===` `:===` `!===`
pass	`:pass`	raw	`++++`
quote	`:quote`	compound	`____`

You may notice the absence of the source block. That's because source is not a container type. Rather, it's a specialization of the listing (or literal) container as designated by the block style. The verse and admonition blocks are also noticeably absent. That's because they repurpose the structural containers for quote and example blocks, respectively.

The default context is assumed when an explicit block style is not present.

Currently, table is a specialized structural container that cannot be enlisted as a custom block.

Unlike other structural containers, a comment block is not preserved in the parsed document and therefore doesn't have a context or content model.

When creating a custom block, it's important to choose a structural container that provides the right content model. This allows a text editor to understand how to parse the block and provide a reasonable fallback when the extension is not loaded.

Structural containers are used to define delimited blocks. The structural container provides a default context and expected content model, but the actual context and content model is determined after considering the metadata on the block (specifically the declared block content).

Nesting blocks

Using delimited blocks, you can nest blocks inside of one other. (Blocks can also be nested inside sections, list items, and table cells, which is a separate topic).

First, the parent block must have the compound content model. The compound content model means that the block's content is a sequence of zero or more blocks.

When nesting a block that uses a different structural container from the parent, it's enough to ensure that the child block is entirely inside the parent block. Delimited blocks cannot be inter-

leaved.

```
====
Here's a sample AsciiDoc document:

----
= Document Title
Author Name

Content goes here.
----

The document header is useful, but not required.
====
```

When nesting a delimited block that uses the same structural container, it's necessary to vary the length of the delimiter lines (i.e., make the length of the delimiter lines for the child block different than the length of the delimiter lines for the parent block). Varying the delimiter line length allows the parser to distinguish one block from another.

```
====
Here are your options:

.Red Pill
[%collapsible]
======
Escape into the real world.
======

.Blue Pill
[%collapsible]
======
Live within the simulated reality without want or fear.
======
====
```

The delimiter length for the nested structural container can either be shorter or longer than the parent. That's a personal style choice.

Build a Basic Block

Build a delimited block

In this section, we'll create a delimited sidebar block. The delimiter for the sidebar style is four asterisks (****).

1. Enter the opening delimiter at the beginning of a new line and then press `Enter`.

```
Text in your document.

****
```

2. On the new line, enter your content, such as paragraphs, delimited blocks, directives, and macros. The delimited block's style will be applied to all of this content until the closing delimiter.

```
Text in your document.

****
This is content in a sidebar block.

image::name.png[]

This is more content in the sidebar block.
```

3. To end the delimited block, press `Enter` at the end of your last line of content. On the new line, type the closing delimiter.

```
Text in your document.

****
This is content in a sidebar block.

image::name.png[]

This is more content in the sidebar block.
****
```

That's it. You've built a delimited block.

Build a block from a paragraph

In some cases, you can style a block using the style's name. If the content is contiguous (not interrupted by empty lines or comment lines), you can assign the block style's name in an attribute list placed above the content. 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.
```

However, note that the lines of a styled paragraph are first parsed like a paragraph, then promoted to the specified block type. That means that line comments will be dropped, which can impact verbatim blocks such as a listing block. Thus, the delimited block form is preferred, especially when creating a verbatim block.

Summary of built-in blocks

The following table identifies the built-in block styles, their delimiter syntax, purposes, and the substitutions performed on their contents.

Block	Block Name	Delimiter	Purpose	Substitutions
Paragraph	n/a	n/a	Regular paragraph content (i.e., prose), offset on either side by an empty line. Must start flush to the left margin of the document. The block name can be used to convert the paragraph into most other blocks.	Normal
Literal paragraph	n/a	n/a	A special type of paragraph block for literal content (i.e., preformatted text). Must be indented from the left margin of the document by at least one space. Often used as a shorthand for a literal delimited block when the content does not contain any empty lines.	Verbatim
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. Will be colorized by the source highlighter if enabled on the document and a language is set.	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 *passthrough* or *table* blocks	Varies

Block	Block Name	Delimiter	Purpose	Substitutions
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. Will be colorized by the source highlighter if enabled on the document and a language is set.	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

Add a Title to a Block

You can assign a title to a block, whether it's styled using its style name or delimiters.

Block title syntax

A block title is defined on its own line directly above the block's attribute list, opening delimiter, or block content—which ever comes first. As shown in Example 4, the line must begin with a dot (.) and immediately be followed by the text of the title. The block title must only occupy a single line and thus cannot be wrapped.

Example 4. Block title syntax

```
.This is the title of a sidebar block
****
This is the content of the sidebar block.
****
```

The block title line should not be confused with an ordered list item that uses the . marker. A block title line has no space after the ., whereas the space after a list marker is required.

The next sections will show how to add titles to delimited blocks and blocks with attribute lists.

Add a title to a delimited block

Any delimited block can have a title. If the block doesn't have an attribute list, enter the title on a

new line directly above the opening delimiter. The delimited literal block in Example 5 is titled *Terminal Output*.

Example 5. Add a title to a delimited block

```
.Terminal Output ①
.... ②
From github.com:asciidoctor/asciidoctor
 * branch        main   -> FETCH_HEAD
Already up to date.
....
```

① The block title is entered on a new line. The title must begin with a dot (.). Don't put a space between the dot and the first character of the title.

② If you aren't applying attributes to a block, enter the opening delimiter on a new line directly after the title.

The result of Example 5 is displayed below.

Terminal Output

```
From github.com:asciidoctor/asciidoctor
 * branch        main   -> FETCH_HEAD
Already up to date.
```

In the next section, you'll see how a title is placed on a block that has an attribute list.

Add a title to a block with attributes

When you're applying attributes to a block, the title is placed on the line above the attribute list (or lists). Example 6 shows a delimited source code block that's titled *Specify GitLab CI stages*.

Example 6. Add a title to a delimited source code block

```
.Specify GitLab CI stages ①
[source,yaml] ②
----
image: node:16-buster
stages: [ init, verify, deploy ]
----
```

① The block title is entered on a new line.

② The block's attribute list is entered on a new line directly after the title.

The result of Example 6 is displayed below.

Specify GitLab CI stages

```yaml
image: node:16-buster
```

```
stages: [ init, verify, deploy ]
```

As shown in Example 7, a block's title is placed above the attribute list when a block isn't delimited.

Example 7. Add a title to a non-delimited block

```
.Mint
[sidebar]
Mint has visions of global conquest.
If you don't plant it in a container, it will take over your garden.
```

The result of Example 7 is displayed below.

Mint

Mint has visions of global conquest. If you don't plant it in a container, it will take over your garden.

You may notice that unlike the titles in the previous rendered listing and source block examples, the sidebar's title is centered and displayed inside the sidebar's background. How the title of a block is displayed depends on the converter and stylesheet you're applying to your AsciiDoc documents.

Captioned titles

Several block contexts support captioned titles. A **captioned title** is a title that's prefixed with a caption label and a number followed by a dot (e.g., `Table 1. Properties`).

The captioned title is only used if the corresponding caption attribute is set. Otherwise, the original title is displayed.

The following table lists the blocks that support captioned titles and the attributes that the converter uses to generate and control them.

Blocks that support captioned titles

Block context	Caption attribute	Counter attribute
appendix	`appendix-caption`	`appendix-number`
example	`example-caption`	`example-number`
image	`figure-caption`	`figure-number`
listing, source	`listing-caption`	`listing-number`
table	`table-caption`	`table-number`

All caption attributes are set by default except for the attribute for listing and source blocks (`listing-caption`). The number is sequential, computed automatically, and stored in a corresponding counter attribute.

Let's assume you've added a title to an example block as follows:

```
.Block that supports captioned title
====
Block content
====
```

The block title will be displayed with a caption label and number, as shown here:

Example 1. Block that supports captioned title

Block content

If you unset the `example-caption` attribute, the caption will not be prepended to the title.

Block that supports captioned title

Block content

The counter attribute (e.g., `example-number`) can be used to influence the start number for the first block with that context or the next number selected in the sequence for subsequent occurrences. However, this practice should be used judiciously.

The caption can be overridden using the `caption` attribute on the block. The value of the caption attribute replaces the entire caption, including the space that precedes the title.

Here's how to define a custom caption on a block:

```
.Block Title
[caption="Example {counter:my-example-number:A}: "]
====
Block content
====
```

Here's how the block will be displayed with the custom caption:

Example A: Block Title

Block content

Notice we've used a counter attribute in the value of the caption attribute to create a custom number sequence.

If you refer to a block with a custom caption using an xref, you may not get the result that you expect. Therefore, it's always best to define custom xreftext when you define a custom caption.

Assign an ID

You can assign an ID to any block using an attribute list. Once you've assigned an ID to a block, you can use that ID to link to it using a cross reference.

Block ID syntax

An ID is assigned to a block by prefixing the ID value with a hash (`#`) and placing it in the block's attribute list.

```
[#the-id-of-this-block]
====
Content of delimited example block
====
```

Let's go through some examples of assigning an ID to a block with several attributes, a title, and delimiters.

Assign an ID to a block with attributes

In this section, we'll assign an ID to this blockquote:

> Roads? Where we're going, we don't need roads.
>
> — Dr. Emmett Brown, Back to the Future

When the style attribute is explicitly assigned to a block, the style name is always placed in the first position of the attribute list. Then, the ID is attached directly to the end of the style name.

The blockquote with an assigned style and ID in Example 8 shows this order of attributes.

Example 8. Assign a style and ID to a block

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

Since Example 8 is a blockquote it should have some attribution and citation information. In Example 9, let's attribute this quote to its speaker and original context using the positional attribution attributes that are built into the `quote` style.

Example 9. Assign a style, ID, and positional attributes to a block

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

Except when the `role` and `options` attributes are assigned values using their shorthand syntax (`.` and `%`, respectively), all other block attributes are typically separated by commas (`,`).

Block Masquerading

The declared block style (i.e., the first positional attribute of the block attribute list) can be used to modify the context of any paragraph and most structural containers. This practice is known as block masquerading (meaning to disguise one block as another).

When the context of a paragraph is changed using the declared block style, the block retains the simple content model. When masquerading the context of a structural container, only contexts that preserve the expected content model are permitted.

How it works

If the block style declared on a block matches the name of a context, it sets the context of the block to that value and the resolved block style will be left unset. If the declared block style does not match the name of a context, it will either specialize the context or set the context implicitly and also specialize that context. How the declared block style is handled for a custom block is up to the extension, though a similar process takes place.

Let's consider the case of using the declared block style to change the context of a structural container. In this case, we're using the declared block style to change a literal block to a listing block.

```
[listing]
....
a > b
....
```

Even though the default context for the structural container is `:literal`, the declared block style changes it to `:listing`. The resolved block style of the block remains unset.

The declared block style can also be used to transform a paragraph into a different kind of block. The block will still retain the simple content model. Let's consider the case of turning a normal paragraph into a sidebar.

```
[sidebar]
This sidebar is short, so a styled paragraph will do.
```

Finally, let's consider an admonition block. Declaring the NOTE block style on an example structural container transforms it to an admonition block and also sets the style of the block to NOTE.

```
[NOTE]
====
Remember the milk.
====
```

This technique also works for converting a paragraph into an admonition block.

```
[NOTE]
Remember the milk.
```

Where permitted, the declared block style can be used to specialize the context of the block, change the context of the block, or both.

Built-in permutations

The table below lists the structural containers whose context can be altered, and which contexts are valid, using the declared block style.

Type	Default context	Masquerading contexts
example	:example	admonition (designated by the NOTE, TIP, WARNING, CAUTION, or IMPORTANT style)
listing	:listing	literal
literal	:literal	listing (can be designated using the source style)
open	:open	abstract, admonition (designated by the NOTE, TIP, WARNING, CAUTION, or IMPORTANT style), comment, example, literal, listing (can be designated using the source style), partintro, pass, quote, sidebar, verse
pass	:pass	stem, latexmath, asciimath
sidebar	:sidebar	*n/a*
quote	:quote	verse

All the contexts that can be applied to an open block can also be applied to a paragraph. A paragraph also access the `normal` style, which can be applied to revert a literal paragraph to a normal paragraph.

Troubleshooting Blocks

Opening and closing delimiters

The opening and closing delimiters of a delimited block must be the same length. For example, a sidebar is specified by an opening delimiter of four asterisks (****). Its closing delimiter must also be four asterisks (****).

Here's a sidebar using valid delimiter lengths:

```
****
This is a valid delimited block.
It will be styled as a sidebar.
****
```

However, the delimiter lengths in the following delimited block are not equal in length and therefore invalid:

```
********
This is an invalid sidebar block because the delimiter lines are different lengths.
****
```

When an AsciiDoc processor encounters the previous example, it will put the remainder of the content in the document inside the delimited block. As far as the processor is concerned, the closing delimiter is just a line of content. However, the processor will issue a warning if a matching closing delimiter is never found.

If you want the processor to recognize a closing delimiter, it must be the same length as the opening delimiter.

Document Attributes

Each document holds a set of name-value pairs called document attributes. These attributes provide a means of configuring the AsciiDoc processor, declaring document metadata, and defining reusable content. This page introduces document attributes and answers some questions about the terminology used when referring to them.

What are document attributes?

Document attributes are effectively document-scoped variables for the AsciiDoc language. The AsciiDoc language defines a set of built-in attributes, and also allows the author (or extensions) to define additional document attributes, which may replace built-in attributes when permitted.

Built-in attributes either provide access to read-only information about the document and its environment or allow the author to configure behavior of the AsciiDoc processor for a whole document or select regions. Built-in attributes are effectively unordered. User-defined attribute serve as a powerful text replacement tool. User-defined attributes are stored in the order in which they are defined.

Here's a summary of some of the things document attributes are used for:

- Provide access to document information
- Define document metadata
- Turn on or turn off built-in features
- Configure built-in features
- Declare the location of assets, like images
- Store content for reuse throughout a document

Let's look closer at the different types of document attributes.

Types of document attributes

Document attributes fall into the following groups.

Built-in attributes

Built-in attributes add, configure, and control common features in a document. Many built-in attributes only take effect when defined in the document header with an attribute entry.

Boolean attributes are a subgroup of the built-in attribute. If a boolean attribute is defined, but not given a value (i.e., set), it's in the "on" state. If the attribute is not defined (i.e., not set), it's in the "off" state. In that regard, these attributes act as a switch. Their sole function is to turn on or turn off a feature.

User-defined attributes

A user-defined attribute is any attribute that the author sets that isn't reserved by the AsciiDoc language or an extension. Most of the time, user-defined attributes are used as a text replace-

ment tool. These attributes allow the author to define named, reusable content. Thus, instead of having to repeat text throughout the document, such as a product name, that text can be defined in an attribute and referenced by its name instead. This technique helps to keep the document DRY, which stands for "Don't Repeat Yourself".

What does defining a document attribute mean?

- have default values in the case of built-in attributes
- have no value in the case of boolean attributes and built-in attributes with default values
- have a single line value
- have a value that spans multiple, contiguous lines
- have a value that includes basic inline AsciiDoc syntax, such as:
  - attribute references
  - text formatting (if wrapped in a pass macro)
  - inline macros (if wrapped in a pass macro)

But there are certain limitations to be aware of. Document attributes cannot:

- have a value that includes AsciiDoc block content, such as:
  - lists
  - multiple paragraphs
  - blocks (tables, sidebars, examples, etc)
  - other whitespace-dependent markup

What does setting a document attribute mean?

- be set (turned on)

What does unsetting a document attribute mean?

- be unset (turned off) with a leading (preferred) or trailing `!` added to the name

Where are document attributes defined, set, and unset?

Document attributes can be declared in the:

- document header as an attribute entry
- document body as an attribute entry
- API via the `:attributes` option
- CLI via the `-a` option

- override locked attributes assigned from the command line

What does referencing a document attribute mean?

Referencing a document attribute means replacing an attribute name with that attribute's value. A document attribute can be referenced in the document using the syntax `{name}`, where `name` is the name of the attribute.

Where can document attributes be referenced?

A document attribute can be referenced anywhere in the document where the attributes substitution is applied. Generally speaking, the attributes substitution is applied to the value of an attribute entry, titles, paragraph text, list text, the value of an element attribute, and the target of a macro.

A document attribute can only be referenced after it has been defined.

Attribute Entries

What is an attribute entry?

Before you can use a document attribute in your document, you have to declare it. 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, followed by the optional value. Each attribute entry must be entered on its own line. An attribute entry starts with an opening colon (`:`), directly followed by the attribute's name, and then a closing colon (`:`). This **sets** — that is, turns on — the document attribute so you can use it in your document.

```
:name-of-an-attribute: ①
```

① The attribute's name is directly preceded with a opening colon (`:`) and directly followed by a closing colon (`:`).

In many cases, you explicitly assign a value to a document attribute by entering information after its name in the attribute entry. The value must be offset from the closing colon (`:`) by at least one space.

```
:name-of-an-attribute: value of the attribute ①
```

① An explicitly assigned value is offset from the closing colon (`:`) by at least one space. At the end of the value, press `Enter`.

Take note that header substitutions automatically get applied to the value by default. That means

you don't need to escape special characters such in an HTML tag. It also means you can reference the value of attributes which have already been defined when defining the value of an attribute. Attribute references in the value of an attribute entry are resolved immediately.

```
:url-org: https://example.org/projects
:url-project: {url-org}/project-name ①
```

① You can reuse the value of an attribute which has already been set using using an attribute reference in the value.

Some built-in attributes don't require a value to be explicitly assigned in an attribute entry because they're a boolean attribute or have an implied value.

```
:name-of-an-attribute: ①
```

① If you don't want to explicitly assign a value to the attribute, press `Enter` after the closing colon (`:`).

When set, the value of a built-in boolean attribute is always empty (i.e., an *empty string*). If you set a built-in attribute and leave its value empty, the AsciiDoc processor may infer a value at processing time.

Where can an attribute entry be declared?

An attribute entry is most often declared in the document header. For attributes that allow it (which includes general purpose attributes), the attribute entry can alternately be declared between blocks in the document body (i.e., the portion of the document below the header).

An attribute entry should not be declared inside the boundaries of a delimited block. When an attribute entry is declared inside a delimited block, the behavior is undefined. What's certain is that preprocessor directives (i.e., include directive, conditional directives) cannot see attributes defined inside a delimited block.

When an attribute is defined in the document header using an attribute entry, that's referred to as a header attribute. A header attribute is available to the entire document until it is unset. A header attribute is also accessible from the document metadata for use by built-in behavior, extensions, and other applications that need to consult its value (e.g., `source-highlighter`).

When an attribute is defined in the document body using an attribute entry, that's simply referred to as a document attribute. For any attribute defined in the body, the attribute is available from the point it is set until it is unset. Attributes defined in the body are not available via the document metadata.

Unless the attribute is locked, it can be unset or assigned a new value in the document header or body. However, note that unsetting or redefining a header attribute that controls behavior in the document body usually has no affect. See the Document Attributes Reference for where in a document each attribute can be set.

Defining document attributes without an attribute entry

Document attributes can also be declared (set with an optional value or unset) outside the document via the CLI and API. The attribute entry syntax is not used in these cases. Rather, they are declared using the provided option. For the API, attributes are declared using the `:attributes` option (which supports various entry formats). For the CLI, the attribute is declared using the `-a` option.

When an attribute is assigned a value outside of the document, the value is stored as is, meaning substitutions are not applied to it. That also means that the special characters and quote substitutions are not applied to the value of that attribute when it is referenced in the document. However, subsequent substitutions, such as the macro substitution, do get applied. This behavior is due to that fact that the attributes substitution is applied after the special characters and quote substitutions. In order to force these substitutions to be applied to the value of the attribute, you must alter the substitution order at the point of reference. Here's an example using the inline pass macro.

```
pass:a,q[{attribute-with-formatted-text}]
```

When an attribute is declared from the command line or API, it is implicitly a document header attribute. By default, the attribute becomes locked (i.e., hard set or unset) and thus cannot be changed by the document. This behavior can be changed by adding an `@` to the end of the attribute name or value (i.e., the soft set modifier). See Attribute Assignment Precedence for more information.

The one exception to this rule is the `sectnums` attribute, which can always be changed.

Attribute Entry Names and Values

Valid built-in names

Built-in attribute names are reserved and can't be re-purposed for user-defined attribute names. The built-in attribute names are listed in the Document Attributes Reference and Character Replacement Attributes Reference.

Valid user-defined names

User-defined attribute names must:

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

A user-defined attribute name cannot contain dots (.) or spaces. Although uppercase characters are permitted in an attribute name, the name is converted to lowercase before being stored. For example, `URL-REPO` and `URL-Repo` are treated as `url-repo` when a document is loaded or converted. A best practice is to only use lowercase letters in the name and avoid starting the name with a number.

Attribute value types and assignment methods

Depending on the attribute, its value may be an empty string, an integer such as 5 or 2, or a string of characters like your name or a URL. Attributes that accept string values may include references to other attributes and inline macros. Values can't contain complex, multi-line block elements such as tables or sidebars.

An attribute's value may be assigned by default when the value is left empty in an attribute entry or the value may be assigned explicitly by the user. The type of value an attribute accepts and whether it uses a default value, has multiple built-in values, accepts a user-defined value, or requires a value to be explicitly assigned depends on the attribute.

Built-in values

Many built-in attributes have one or more built-in values. One of these values may be designated as the attribute's default value. The AsciiDoc processor will fall back to this default value if you set the attribute but leave the value empty. Additionally, the processor automatically sets numerous built-in attributes at processing time and assigns them their default values unless you explicitly unset the attribute or assign it another value. For instance, the processor automatically sets all of the character replacement attributes.

If you want to use the non-default value of a built-in attribute, you need to set it and assign it an alternative value.

Empty string values

The value for built-in boolean attributes is always left empty in an attribute entry since these attributes only turn on or turn off a feature. During processing, the AsciiDoc processor assigns any activated boolean attributes an *empty string* value.

Explicit values

You must explicitly assign a value to an attribute when:

- it doesn't have a default value,
- you want to override the default value, or
- it's a user-defined attribute.

The type of explicit value a built-in attribute accepts depends on the attribute. User-defined attributes accept string values. Long explicit values can be wrapped.

Wrap Attribute Entry Values

Soft wrap attribute values

If the value of a document attribute is too long to fit on the screen, you can split the value across multiple lines with a line continuation to make it easier to read.

A **line continuation** consists of a space followed by a backslash character (\) at the end of the line. The line continuation must be placed on every line of a multiline value except for the last line.

Lines that follow a line continuation character may be indented, but that indentation will not be included in the value.

When the processor reads the attribute value, it folds the line continuation, the newline, and any ensuing indentation into a single space. In this case, we can say that the attribute value has soft wraps.

Let's assume we want to define an attribute named `description` that has a very long value. We can split this attribute up across multiple lines by placing a line continuation at the end of each line of the value except for the last.

Example 10. A multiline attribute value with soft wraps

```
:description: 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 the value neatly in the header \
so it remains readable to folks looking at the AsciiDoc source.
```

If the line continuation is missing, the processor will assume it has found the end of the value and will not include subsequent lines in the value of the attribute.

Hard wrap attribute values

You can force an attribute value to hard wrap by inserting a hard line break replacement in front of the line continuation. A hard line break replace is a space followed by a plus character (`+`).

As described in the previous section, the line continuation, newline, and ensuing indentation is normally replaced with a space. This would prevent the hard line break replacement from being recognized. However, the processor accounts for this scenario and leaves the newline intact.

Let's assume we want to define an attribute named `haiku` that requires hard line breaks. We can split this attribute up across multiple lines and preserve those line breaks by placing a hard line break replacement followed by a line continuation at the end of each line of the value except for the last.

Example 11. A multiline attribute value with hard wraps

```
: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 as hard line breaks.

Attribute Entry Substitutions

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, which replaces special characters followed by attribute references, is applied to the values of attribute entries, regardless of whether the

entries are defined in the header or in the document body. This is the same group that gets applied to metadata lines (author and revision information) in the document header.

That means that any inline formatting in an attribute value isn't interpreted because:

1. inline formatting is not applied when the AsciiDoc processor sets an attribute, and
2. inline formatting is not applied when an attribute is referenced since the relevant substitutions come before attributes are resolved.

Change substitutions when assigning a value

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 will be 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 we can apply the quotes substitution to the value of an attribute entry:

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

Internally, the value is stored as `MyApp<sup>2</sup>`. 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.

Substitutions for attributes defined outside the document

Unlike attribute entries, substitutions are **not** applied to the value of an attribute passed in to the AsciiDoc processor. An attribute can be passed into the AsciiDoc processor using the `-a` CLI option or the `:attributes` API option. When attributes are defined external to the document, the value must be prepared so it's ready to be referenced as is. If the value contains XML special characters, that means those characters must be pre-escaped. The exception would be if you intend for XML/HTML tags in the value to be preserved. If the value needs to reference other attributes, those values must be pre-replaced.

Let's consider the case when the value of an attribute defined external to the document contains an ampersand. In order to reference this attribute safely in the AsciiDoc document, the ampersand must be escaped:

```
$ asciidoctor -a equipment="a bat &amp;amp; ball" document.adoc
```

You can reference the attribute as follows:

```
To play, you'll need {equipment}.
```

If the attribute were to be defined in the document, this escaping would not be necessary.

```
:equipment: a bat & ball
```

That's because, in contrast, substitutions are applied to the value of an attribute entry.

Change substitutions when referencing an attribute

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 post-processing the attribute value.

Inline Attribute Entries

An attribute reference can be used to set or unset an attribute inline as an alternative to a dedicated attribute entry line. This mechanism allows you to set or unset an attribute in places where attribute entries lines are not permitted, such as in a normal table cell or a list item.

You're strongly discouraged from using inline attribute entries unless you understand their limitations or they are a last resort for fulfilling a use case. It's very likely that this functionality will be removed from the AsciiDoc language since its behavior is difficult to define.

Attributes can be defined inline using the following notation:

```
{set:name:value}
```

The value segment is optional. If absent, the value defaults to empty string. In that case, the notation is reduced to:

```
{set:name}
```

If you add a `!` character after the name to unset the attribute instead:

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

Here's an example that uses an inline attribute entry to set the `sourcedir` attribute to the value `src/main/java`.

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

This assignment is effectively the same as:

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

However, it's important to understand that inline attribute assignments are processed in a different phase than attribute entry lines. Inline attribute entries are processed when attribute references are replaced, as part of the attributes substitution. Therefore, the result of the assignment is only available to attribute references that follow it. These assignments are not visible in the document model after the document has been loaded.

Declare Built-In Attributes

An AsciiDoc processor has numerous attributes reserved for special purposes. **Built-in attributes** add, configure, and control common features in a document. Many built-in attributes only take

effect when defined in the document header with an attribute entry.

Use an attribute's default value

Many built-in attributes have a default value. When you want to activate a built-in attribute and assign it its default value, you can leave the value in the attribute entry empty.

For example, to turn on the Table of Contents for a document, you set the `toc` attribute using an attribute entry in the document header.

```
= Title of Document
:toc:
```

The default value of an activated attribute will be assigned at processing time, if:

1. it has a default value, and
2. the value in the attribute entry is left empty

In the example above, the default value of `auto` will be assigned to `toc` since the value was left empty in the attribute entry.

Override an attribute's default value

You may not want to use the default value of a built-in attribute. In the next example, we'll override the default value of an attribute that the AsciiDoc processor sets automatically. The built-in attribute `doctype` is automatically set and assigned a value of `article` at processing time. However, if you want to use AsciiDoc's book features, the `doctype` attribute needs to be assigned the `book` value.

```
= Title of My Document
:doctype: book ①
```

① Set `doctype` in the document header and assign it the value `book`. Explicit values must be offset from the closing colon (`:`) by at least one space.

To override an attribute's default value, you have to explicitly assign a value when you set the attribute. The value assigned to an attribute in the document header replaces the default value (assuming the attribute is not locked via the CLI or API).

Override a default asset directory value

You can also use the built-in asset directory attributes to customize the base path to images (default: `empty`), icons (default: `./images/icons`), stylesheets (default: `./stylesheets`) and JavaScript files (default: `./javascripts`).

Example 12. Replace the default values of the built-in asset directory attributes

```
= My Document
:imagesdir: ./images
```

```
:iconsdir: ./icons
:stylesdir: ./styles
:scriptsdir: ./js
```

The four built-in attributes in the example above have default values that are automatically set at processing time. However, in the example, they're being set and assigned explicit values in the document header. This explicit user-defined value replaces the default value (assuming the attribute is not locked via the CLI or API).

Set Boolean Attributes

A boolean attribute is a built-in attribute that acts like a toggle. Its sole function is to turn on a feature or behavior.

Boolean attribute entry syntax

A boolean attribute is set using an attribute entry in the header or body of a document. The value of a boolean value is always empty because boolean attributes in AsciiDoc only accept an *empty string* value. In AsciiDoc, an attribute that is set, but has an empty value, is interpreted as the *true* state and an attribute which is not set is interpreted as the *false* state. However, a processor may interpret a value of `true` as the true state as well.

```
:name-of-a-boolean-attribute: ①
```

① On a new line, type a colon (`:`), directly followed by the attribute's name and then another colon (`:`). After the closing colon, press `Enter`. The attribute is now set and its behavior will be applied to the document.

Declare a boolean attribute

Let's use an attribute entry to turn on the built-in boolean attribute named `sectanchors`. When `sectanchors` is set, it activates an anchor in front of a section title when a cursor hovers over it.

```
= Document Title
:sectanchors: ①
```

① The value of `sectanchors` is always left empty because it's a boolean attribute.

Declare Custom Attributes

When you find yourself typing the same text repeatedly, or text that often needs to be updated, consider creating your own attribute.

User-defined attribute names and values

A user-defined attribute must have a name and explicitly assigned value.

The attribute's name must:

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

The name cannot contain dots or spaces.

Although uppercase characters are permitted in an attribute name, the name is converted to lowercase before being stored. For example, `URL` and `Url` are treated as `url`. A best practice is to only use lowercase letters in the name and avoid starting the name with a number.

Attribute values can:

- be any inline content, and
- contain line breaks, but only if an explicit line continuation (`+`) is used.

Create a custom attribute and value

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

Example 13. Create a user-defined attribute and value

```
:disclaimer: Don't pet the wild Wolpertingers. If you let them into your system, we're
\ ①
not responsible for any loss of hair, chocolate, or purple socks.
:url-repo: https://github.com/asciidoctor/asciidoctor
```

① Long values can be soft wrapped using a backslash (`\`).

Now, you can reference these attributes throughout the document.

Unset Attributes

Document attributes—built-in, boolean, and custom—can be unset in the document header and document body.

Unset a document attribute in the header

Document attributes are unset by adding a bang symbol (`!`) directly in front of (preferred) or after the attribute's name. Like when setting an attribute in a document header, the attribute entry must be on its own line. Don't add a value to the entry.

```
= Title
:!name: ①
:name!: ②
```

① An attribute is unset when a `!` is prefixed to its name (preferred).

② An attribute is unset when a `!` is appended to its name.

Let's use an attribute entry to turn off the built-in boolean attribute named `sectids`. The AsciiDoc processor automatically sets `sectids` at processing time unless you unset it. The `sectids` attribute generates an ID for each section from the section's title.

Example 14. Unset a boolean attribute

```
= Document Title
:!sectids: ①
```

① On a new line, type a colon (`:`), directly followed by a bang symbol (`!`), the attribute's name, and then another colon (`:`). After the closing colon, press `Enter`. The attribute is now unset and its behavior won't be applied to the document.

Once an attribute is unset, its behavior is deactivated. When `sectids` is unset, the AsciiDoc processor will not generate IDs from section titles at processing time.

Let's unset the built-in attribute `example-caption`. This is an attribute that is set and assigned a default value of `Example` automatically by the AsciiDoc processor when you use an example block.

Example 15. Unset an automatically declared attribute

```
= Title
:!example-caption: ①
```

① Example blocks won't be labeled and numbered, e.g., Example 1, because the attribute controlling that behavior is unset with the leading `!`.

Unset a document attribute in the body

Custom document attributes and some built-in document attributes can be turned off in the body of the document using an attribute entry and the bang symbol (`!`) as described in the previous section. For example, let's say you set the section numbering attribute in the header of your document; however, you don't want the two sections midway through the document to be numbered. To disable the numbering on these two sections, you'd unset `sectnums` before the first section you didn't want numbered and then reset it when you wanted the numbering to start again.

```
= Title
:sectnums: ①

== Section Title

:!sectnums: ②
== Section Title

=== Section Title
```

```
:sectnums: ③
== Section Title
```

① The `sectnums` attribute is set in the header to activate section numbering throughout the document.

② `sectnums` is unset by adding a `!` to it's name. The `!` can be placed either before or after the attribute's name. The attribute entry must be placed on its own line. All of the sections below where the attribute is unset will not be numbered.

③ `sectnums` is set and all subsequent sections will be numbered.

Reference Attributes

You'll likely want to insert the value of a user-defined or built-in document attribute in various locations throughout a document. To reference a document attribute for insertion, enclose the attribute's name in curly brackets (e.g., `{name-of-attribute}`). This inline element is called an **attribute reference**. The AsciiDoc processor replaces the attribute reference with the value of the attribute. To prevent this replacement, you can prefix the element with a backslash (e.g., `\{name-of-attribute}`).

Reference a custom attribute

Before you can reference a custom (i.e., user-defined) attribute in a document, it must first be declared using an attribute entry in the document header. In Example 16, we declare two user-defined attributes that we'll later be able to reference.

Example 16. Custom attributes set in the document header

```
= Ops Manual
:disclaimer: Don't pet the wild Wolpertingers. We're not responsible for any loss of
hair, chocolate, or purple socks.
:url-repo: https://github.com/asciidoctor/asciidoctor
```

Once you've set and assigned a value to a document attribute, you can reference that attribute throughout your document. In Example 17, the attribute `url-repo` is referenced twice and `disclaimer` is referenced once.

Example 17. Custom attributes referenced in the document body

```
Asciidoctor is {url-repo}[open source]. ①

WARNING: {disclaimer} ②
If you're missing a lime colored sock, file a ticket in
the {url-repo}/issues[Asciidoctor issue tracker]. ③
(Actually, please don't).
```

① Attribute references can be used in macros.

② Attribute references can be used in blocks, such as admonitions, and inline. Since there isn't an

empty line between the `disclaimer` reference and the next sentence, the sentence will be directly appended to the end of the attribute's value when it's processed.

③ The reference to the `url-repo` attribute is inserted to build the complete URL address, which is interpreted as a URL macro.

As you can see below, the attribute references are replaced with the corresponding attribute value when the document is processed.

Asciidoctor is open source.

Don't pet the wild Wolpertingers. We're not responsible for any loss of hair, chocolate, or purple socks. If you're missing a lime colored sock, file a ticket in the Asciidoctor issue tracker. Actually, please don't.

Reference a built-in attribute

A built-in document attribute (i.e., a document attribute which is automatically set by the processor) is referenced the same way as a custom (i.e., user-defined) document attribute. For instance, an AsciiDoc processor automatically sets these supported character replacement attributes. That means that you can reference them throughout your document without having to create an attribute entry in its header.

```
TIP: Wolpertingers don't like temperatures above 100{deg}C. ①
Our servers don't like them either.
```

① Reference the character replacement attribute `deg` by enclosing its name in a pair of curly brackets (`{` and `}`).

As you can see below, the attribute reference is replaced with the attribute's value when the document is processed.

Wolpertingers don't like temperatures above 100°C. Our servers don't like them either.

Escape an attribute reference

You may have a situation where a sequence of characters occurs in your content that matches the syntax of an AsciiDoc attribute reference, but is not, in fact, an AsciiDoc attribute reference. For example, if you're documenting path templating, you may need to reference a replaceable section of a URL path, which is also enclosed in curly braces (e.g., /items/{id}). In this case, you need a way to escape the attribute reference so the AsciiDoc processor knows to skip over it. Otherwise, the processor could warn about a missing attribute reference or perform an unexpected replacement. AsciiDoc provides several ways to escape an attribute reference.

Prefix with a backslash

You can escape an attribute reference by prefixing it with a backslash. When the processor encounters this syntax, it will remove the backslash and pass through the remainder of what looks to be an attribute reference as written.

In Example 18, the attribute reference is escaped using a backslash.

Example 18. An attribute reference escaped using a backslash

```
In the path /items/\{id}, id is a path parameter.
```

In the output of Example 18, we can see that the `{id}` expression in the path is preserved.

```
In the path /items/{id}, id is a path parameter.
```

Keep in mind that the backslash will only be recognized if the text between the curly braces is a valid attribute name. If the syntax that follows the backslash does not match an attribute reference, the backslash will not be removed during processing.

Enclose in a passthrough

You can also escape an attribute reference by enclosing it in an inline passthrough. In this case, the processor uses the normal substitution rules for the passthrough type you have chosen.

In Example 19, the attribute reference is escaped by enclosing it in an inline passthrough.

Example 19. An attribute reference escaped by enclosing it in an inline passthrough

```
In the path +/items/{id}+, id is a path parameter.
```

In the output of Example 19, we can see that the `{id}` expression in the path is preserved.

```
In the path /items/{id}, id is a path parameter.
```

When using an inline passthrough, you don't have to worry whether the curly braces form an attribute reference or not. All the text between the passthrough enclosure will get passed through to the output.

Alternative escape mechanisms

Attribute references are replaced by the attributes substitution. Therefore, wherever you can control substitutions, you can prevent attribute references from being replaced. This includes the inline pass macro as well as the subs attribute on a block. See using passthroughs to prevent substitutions for more details.

Handle Unresolved References

When you reference a missing attribute (e.g., `{does-not-exist}`), the AsciiDoc processor will leave the attribute reference behind. If you undefine an attribute on the same line as other text (e.g., `{set:attribute-no-more!}`), the processor will drop the whole line. You can tailor these behaviors using the `attribute-missing` and `attribute-undefined` attributes. You'll want to think about how you want the processor to handle these situations and configure the processor accordingly.

Missing attribute

The `attribute-missing` attribute controls how missing references are handled. By default, missing references are left behind so the integrity of the document is preserved and it's easy for the author to track down.

This attribute has four possible values:

`skip`
: leave the reference in place (default setting)

`drop`
: drop the reference, but not the line

`drop-line`
: drop the line on which the reference occurs (matches behavior of AsciiDoc.py)

`warn`
: print a warning about the missing attribute

The setting you might find of most interest is `warn`, which gives you a warning whenever the processor encounters an attribute reference that cannot be resolved, but otherwise leaves the line alone.

Consider the following line:

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

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

`attribute-missing` value	Result
`skip`	Hello, {name}!
`drop`	Hello, !
`drop-line`	
`warn`	`asciidoctor: WARNING: skipping reference to missing attribute: XYZ`

History

AsciiDoc.py always drops the line that contains a reference to a missing attribute (effectively `attribute-missing=drop-line`). This "feature" was a side effect of how the processor was implemented and not designed with the writer in mind. The

> behavior is frustrating for the writer because it's hard to detect where its occurring and can result in loss of important content. That's why Asciidoctor uses a different default behavior and, further, allows the behavior to be customized.

There are a few cases where 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 leave behind the same warning message in the converted document.

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

Forcing failure

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 CLI.

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

The processor will convert the entire document, but the application will complete with a non-zero exit status.

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. It's up to the application code to decide how and when to terminate the application.

Undefined attribute

The `attribute-undefined` attribute controls how an expression that undefines an attribute (e.g., `{set:name!}`) are handled. By default, the line containing the expression is dropped since the expression is intended to be a statement, not a content reference.

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.py)

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.

Attribute Assignment Precedence

Default attribute value precedence

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

1. An attribute defined using the API or CLI
2. An attribute defined in the document
3. 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.

Altering the 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 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
```

Alternately, 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
```

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.

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

1. An attribute passed to the API or CLI whose value does not end in `@`
2. An attribute defined in the document
3. An attribute passed to the API or CLI whose value or name ends in `@`
4. The default value of the attribute, if applicable

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

Counters

Counters are used to store and display ad-hoc sequences of numbers or Latin characters.

Counters are a poorly defined feature in AsciiDoc and should be avoided if possible. If you do use counters, you should only used them for the most rudimentary use cases, such as making a sequence in a list, table column, or prose. You should **not** use counters to build IDs (i.e., references) or reference text. Using counters across the boundaries of a reference will very likely result in unexpected behavior.

A counter is implemented as a specialized document attribute. You declare and display a counter using an attribute reference, where 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 resolved. The term **increment** means to advance the attribute value 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 (e.g., a → b). 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.

If you want to use a counter value in a section title, you should define it first using an attribute reference.

```
:seq1: {counter:seq1}
== Section {seq1}

The sequence in this section is {seq1}.

:seq1: {counter:seq1}
== Section {seq1}

The sequence in this section is {seq1}.
```

Here's the resulting output:

Section 1

The sequence in this section is 1.

Section 2

The sequence in this section is 2.

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.

```
.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

Element Attributes

Element attributes are a powerful means of controlling the built-in settings of individual block and inline elements in the AsciiDoc syntax. They can also be used to add supplemental information, such as citation metadata and fallback content, to certain elements.

What are element attributes?

Element attributes define the built-in and user-defined settings and metadata that can be applied to an individual block element or inline element in a document (including macros). Although the include directive is not technically an element, element attributes can also be defined on an include directive.

Element attributes may be positional (value only) or named (key/value pair). Some built-in and extension elements will map a positional attribute to a named attribute. Each element recognizes a predefined set of positional and/or named element attributes. Authors may define any number of custom element attributes for passing information to an extension or document analyzer.

Like document attributes, there's no strict schema for element attributes, or for the value of the `options` element attribute. There's a core set of reserved attributes shared by all block elements and most inline elements, which includes id, role, opts, and title. Certain elements may reserve additional attributes and option values. For example, the source block reserves the `lang` attribute to set the source language and the `linenums` option to enable line numbers. The link macro reserves the `window` attribute to change the target window of a link and the `nofollow` option to prevent crawlers from following it. Otherwise, the schema for element attributes is open-ended, thus allowing extensions to use them for their own purpose.

Element attributes are commonly used for the following purposes:

- Declare the ID of an element
- Turn on or turn off an individual element's built-in features
- Configure the built-in features of an individual element
- Apply user-defined information, such as citation metadata, fallback text, link text, and target content, to an individual element
- Apply user-defined roles and behaviors to an individual element

Unlike document attributes, element attributes are defined directly on the element to which they apply using an attribute list.

Attribute lists

Attributes can be assigned to block and inline elements using an **attribute list** (often abbreviated as attrlist).

Example 20. Anatomy of an attribute list

```
first-positional,second-positional,named="value of named"
```

Entries in an attribute list are separated by commas, excluding commas inside quotes. The syntax used for an attribute list entry determines whether it's a positional or named attribute. The space after the comma separating entries is optional. To learn more about how the attribute list is parsed, see Positional and Named Attributes.

For **block elements**, the attribute list is placed inside one or more block attribute lines. A block attribute line is any line of text above the start of a block (e.g., the opening delimiter or simple content) that begins with `[` and ends with `]`. This line can be interspersed with other block metadata lines, such as the block title. The text enclosed in the `[` and `]` boundaries is assumed to be a valid attribute list and the line is automatically consumed. If the text cannot be parsed, an error message will be emitted to the log.

Example 21. A block attribute line

```
[style,second-positional,named="value of named"]
```

The opening line of a paragraph may inadvertently match the syntax of a block attribute line. If this happens, append `{empty}` to the end of the line to disrupt the syntax match.

For **block and inline macros**, the attribute list is placed between the square brackets of the macro. The text in an attribute list of a block macro never needs to be escaped. For an inline macro, it may be necessary to escape the text in the attribute list to avoid prematurely ending the macro or unwanted substitutions.

Example 22. A block macro with an attribute list

```
name::target[first-positional,second-positional,named="value of named"]
```

For **formatted text**, the attribute list is placed in the square brackets in front of the text enclosure. However, formatted text only supports a restricted form of the attribute list. Specifically, it does not support named attributes, only the attribute shorthand syntax.

Example 23. Formatted text with an attribute list

```
[#idname.rolename]*text with id and role*
```

Attribute lists:

- apply to blocks, macros, and inline elements,
- can contain positional and named attributes, and
- take precedence over document attributes if the element supports the override.

As mentioned in the previous section, the schema for element attributes is open-ended. Any positional or named attributes that are not recognized will be stored on the element, but will not have an impact on the behavior or output. Extensions may use this auxiliary information to influence their behavior and/or customize the output.

Positional and Named Attributes

This page breaks down the difference between positional and named attributes on an element and the rules for parsing an attribute list.

Positional attribute

Entries in an attribute list that only consist of a value are referred to as positional attributes. The position is the 1-based index of the entry once all named attributes have been removed (so they may be interspersed).

The positional attribute may be dually assigned to an implicit attribute name if the block or macro defines a mapping for positional attributes. Here are some examples of those mappings:

- `icon:` 1 ⇒ size
- `image:` and `image::` 1 ⇒ alt (text), 2 ⇒ width, 3 ⇒ height
- Delimited blocks: 1 ⇒ block style and attribute shorthand
- Other inline quoted text: 1 ⇒ attribute shorthand
- `link:` and `xref:` 1 ⇒ text
- Custom blocks and macros can also specify positional attributes

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 the same as the first, but written out in longhand form.

Block style and attribute shorthand

The first positional attribute on all blocks (including sections) is special. It's used to define the block style. It also supports a shorthand notation for defining the ID, role, and options attributes. This shorthand notation can also be used on formatted text, even though formatted text doesn't technically support attributes.

To be clear, the shorthand notation is allowed in two places:

- The first positional attribute in block attribute line (i.e., the location of the block style)
- The text inside the brackets of formatted text (which is otherwise treated as the role)

The attribute shorthand is inspired by the HAML and Slim template languages as a way of saving the author some typing. Instead of having to use the longhand form of a name attribute, it's possible to compress the assignment to a value prefixed by a special marker. The markers are mapped as follows:

- `#` - ID
- `.` - role
- `%` - option

Each shorthand entry is placed directly adjacent to previous one, starting immediately after the optional block style. The order of the entries does not matter, except for the style, which must come first.

Here's an example that shows how to set an ID on a section using this shorthand notation:

```
[#custom-id]
== Section with Custom ID
```

The shorthand entry must follow the block style, if present. Here's an example that shows how to set an ID on an appendix section using this shorthand notation:

```
[appendix#custom-id]
== Appendix with Custom ID
```

Here's an example of a block that uses the shorthand notation to set the ID, a role, and an option for a list. Specifically, this syntax sets the ID to `rules`, adds the role `prominent`, and sets the option `incremental`.

```
[#rules.prominent%incremental]
* Work hard
* Play hard
* Be happy
```

A block can have multiple roles and options, so these shorthand entries may be repeated. Here's an example that shows how to set several options on a table. Specifically, this syntax sets the `header`, `footer`, and `autowidth` options.

```
[%header%footer%autowidth]
|===
|Header A |Header B
|Footer A |Footer B
|===
```

This shorthand notation also appears on formatted text. Here's an example that shows how to set the ID and add a role to a strong phrase. Specifically, this syntax sets the ID to `free-world` and adds

the `goals` role.

```
[#free-world.goals]*free the world*
```

Formatted text does not support a style, so the first and only positional attribute is always the shorthand notation.

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 the value contains the **same** quote character used to enclose the value, the quote character in the value must be escaped by prefixing it with a backslash (e.g., `value="the song \"Dark Horse\""`).

If enclosing quotes are used, they are dropped from the parsed value and the preceding backslash is dropped from any escaped quotes.

Unset a named attribute

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

Attribute list parsing

The source text that's used to define attributes for an element is referred to as an **attrlist**. An attrlist is always enclosed in a pair of square brackets. This applies for block attributes as well as attributes on a block or inline macro. The processor splits the attrlist into individual attribute entries, determines whether each entry is a positional or named attribute, parses the entry accordingly, and assigns the result as an attribute on the node.

The rules for what defines the boundaries of an individual attribute, and whether the attribute is positional or named, are defined below. In these rules, `name` consists of a word character (letter or numeral) followed by any number of word or `-` characters (e.g., `see-also`).

- Attribute references are expanded before the attrlist is parsed (i.e., the attributes substitution is applied).
- Parsing an attribute proceeds from the beginning of the attribute list string or after a previously identified delimiter (`,`).
  - The first character of an attribute list cannot be a tab or space. For subsequent attributes, any leading space or tab characters are skipped.
- If a valid attribute name is found, and it is followed by an equals sign (=), then the parser recognizes this as a named attribute. The text after the equals sign (=) and up to the next comma or end of list is taken as the attribute value. Space and tab characters around the equals sign (=) and at the end of the value are ignored.
- Otherwise, this is a positional attribute with a value that ends at the next delimiter or end of list.

Any space or tab characters at the boundaries of the value are ignored.

- To parse the attribute value:
  - If the first character is not a quote, the string is read until the next delimiter or end of string.
  - If the first character is a double quote (i.e., `"`), then the string is read until the next unescaped double quote or, if there is no closing double quote, the next delimiter. If there is a closing double quote, the enclosing double quote characters are removed and escaped double quote characters are unescaped; if not, the initial double quote is retained.
  - If the next character is a single quote (i.e., `'`), then the string is read until the next unescaped single quote or, if there is no closing single quote, the next delimiter. If there is a closing single quote, the enclosing single quote characters are removed and escaped single quote characters are unescaped; if not, the initial single quote is retained. If there is a closing single quote, and the first character is not an escaped single quote, substitutions are performed on the value as described in Substitutions.

When to escape a closing square bracket

Since the terminal of an attrlist is a closing square bracket, it's sometimes necessary to escape a closing square bracket if it appears in the value of an attribute.

In line-oriented syntax such as a block attribute list, a block macro, and an include directive, you do not have to escape closing square brackets that appear in the attrlist itself. That's because the parser already knows to look for the closing square bracket at the end of the line.

If a closing square bracket appears in the attrlist of an inline element, such as an inline macro, it usually has to be escaped using a backslash or by using the character reference `]`. There are some exceptions to this rule, such as a link macro in a footnote, which are influenced by the substitution order.

Substitutions

Recall that attribute references are expanded before the attrlist is parsed. Therefore, it's not necessary to force substitutions to be applied to a value if you're only interested in applying the attributes substitution. The attributes substitution has already been applied at this point.

If the attribute name (in the case of a positional attribute) or value (in the case of a named attribute) is enclosed in single quotes (e.g., `citetitle='Processed by https://asciidoctor.org'`), and the attribute is defined in an attrlist on a block, then the normal substitution group is applied to the value at assignment time. No special processing is performed, aside from the expansion of attribute references, if the value is not enclosed in quotes or is enclosed in double quotes.

If the value contains the same quote character used to enclose the value, escape the quote character in the value by prefixing it with a backslash (e.g., `citetitle='A \'use case\' diagram, generated by https://plantuml.com'`).

ID Attribute

You can assign an identifier (i.e., unique name) to a block or inline element using the `id` attribute. The `id` attribute is a named attribute. Its purpose is to identify the element when linking, scripting, or styling. Thus, the identifier can only be used once in a document.

An ID:

1. provides an internal link or cross reference anchor for an element
2. can be used for adding additional styling to specific elements (e.g., via a CSS ID selector)

You can assign an ID to blocks using the shorthand hash (`#`) syntax, longhand (`id=`) syntax, or the anchor (`[[]]`) syntax. You can assign an ID to inline elements using the shorthand hash (`#`) syntax or by adding an anchor adjacent to the inline element using the anchor (`[[]]`) syntax. You can assign an ID to a table cell by using an anchor (`[[]]`) at the start of the cell. Likewise, you can assign an ID to a list item by using an anchor (`[[]]`) at the start of the principal text.

Valid ID characters

AsciiDoc does not restrict the set of characters that can be used for an ID when the ID is defined using the named `id` attribute. All the language requires in this case is that the value be non-empty. When the ID is defined using the shorthand hash syntax or the anchor syntax, the acceptable characters is more limited (for example, spaces are not permitted). Regardless, it's not advisable to exploit the ability to use any characters the AsciiDoc syntax allows. The reason to be cautious is because the ID is passed through to the output, and not all output formats afford the same latitude. For example, XML is far more restrictive about which characters are permitted in an ID value.

To ensure portability of your IDs, it's best to conform to a universal standard. The standard we recommend following is a Name value as defined by the XML specification. At a high level, the first character of a Name must be a letter, colon, or underscore and the optional following characters must be a letter, colon, underscore, hyphen, period, or digit. You should not use any space characters in an ID. Starting the ID with a digit is less likely to be problematic, but still best to avoid. It's best to use lowercase letters whenever possible as this solves portability problem when using case-insensitive platforms.

When the AsciiDoc processor auto-generates IDs for section titles and discrete headings, it adheres to this standard.

Here are examples of valid IDs (according to the recommendations above):

```
install
data-structures
error-handling
subject-and-body
unset_an_attribute
```

Here are examples of invalid IDs:

```
install the gem
3 blind mice
-about-the-author
```

Block assignment

You can assign an ID to a block using the shorthand syntax, the longhand syntax, or a block anchor.

In the shorthand syntax, you prefix the name with a hash (`#`) in the first position attribute.

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

In the longhand syntax, you use a standard named attribute.

```
[id=goals]
* Goal 1
* Goal 2
```

In the block anchor syntax, you surround the name with double square brackets:

```
[[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.movie#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., `[[attributes:id:::class-name.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.

Example 24. Quoted text with ID assignment using shorthand syntax

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

Example 25. General text with preceding ID assignment using inline anchor syntax

```
[[free_the_world]]free the world
```

Use an ID as an anchor

An anchor (aka ID) can be defined almost anywhere in the document, including on a 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., `[[attributes:id:::idname]]`) or using the shorthand ID syntax (e.g., `[#idname]`) at the start of an attribute list. The shorthand form is the preferred syntax.

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.

On block element

To reference a block element, you must assign an ID to that block. You can define an ID using the shorthand syntax:

Example 26. Assign an ID to a paragraph using shorthand syntax

```
[#notice]
This paragraph gets a lot of attention.
```

or you can define it using the block anchor syntax:

Example 27. Assign an ID to a paragraph using block anchor syntax

```
[[notice]]
This paragraph gets a lot of attention.
```

As an inline anchor

You can also define an anchor anywhere in content that receives normal substitutions (specifically the macros substitution). You can enclose the ID in double square brackets:

Example 28. Define an inline anchor

```
[[bookmark-a]]Inline anchors make arbitrary content referenceable.
```

or using the shorthand ID syntax.

Example 29. Define an inline anchor using shorthand syntax

```
[#bookmark-b]#Inline anchors can be applied to a phrase like this one.#
```

On a list item

In addition to being able to define anchors on sections and blocks, anchors can be defined inline wherever you can type normal text (anchors are a macros 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:

Example 30. **Invalid** *position for an anchor ID in front of a list item*

```
[[anchor-point]]* list item with invalid anchor
```

Instead, you would put it at the start of the text of the list item:

Example 31. Define an inline anchor on a list item

```
* First item
* [[step2]]Second item
* Third item
```

For a description list, the anchor must be placed at the start of the term:

Example 32. Define an inline anchor on a description list item

```
[[cpu,CPU]]Central Processing Unit (CPU)::
The brain of the computer.

[[hard-drive]]Hard drive::
Permanent storage for operating system and/or user files.
```

You can add multiple anchors to a list item or description list term. However, only the first anchor is registered for use as an xref within the document. The remaining anchors are auxiliary and are used for making deep links (i.e., accessible from a URL fragment).

On a table cell

You can assign an ID to a table cell by placing an inline anchor at the start of the cell.

Example 33. Assigning an ID to a table cell using an inline anchor

```
|===
|[[my_cell]]The table cell I want to jump to.
|===
```

On an inline image

You cannot currently define an ID on an inline image. Instead you need to place an inline anchor adjacent to it.

Example 34. Placing an inline anchor adjacent to an inline image using shorthand

```
[[tiger-image]]image:tiger.png[Image of a tiger]
```

Instead of the shorthand form, you can use the macro `anchor` to achieve the same goal.

Example 35. Placing an inline anchor adjacent to an inline image using a macro

```
anchor:tiger-image[]image:tiger.png[Image of a tiger]
```

Add additional anchors to a section

To add additional anchors to a section (with or without an autogenerated ID), place the anchors in front of the title (without any spaces).

Example 36. Add additional anchors to a section using inline anchors

```
[#version-4_9]
=== [[current]][[latest]]Version 4.9
```

You cannot use inline anchors in a section title to make internal references to that section. The processor will flag these as possible invalid references. These additional anchors are only intended for making deep links using an alternate ID.

Remember that inline anchors are discovered wherever the macros substitution is applied (e.g., paragraph text). If text content doesn't belong somewhere, neither does an inline anchor point.

Customize automatic xreftext

It's possible to customize the text that will be used in the cross reference link (called `xreflabel`). If not defined, the AsciiDoc processor 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).

Example 37. 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[]
```

Role Attribute

You can assign one or more roles to blocks and most inline elements using the `role` attribute. The `role` attribute is a named attribute. Even though the attribute name is singular, it may contain multiple (space-separated) roles. Roles may also be defined using a shorthand (dot-prefixed) syntax.

A role:

1. adds additional semantics to an element
2. can be used to apply additional styling to a group of elements (e.g., via a CSS class selector)
3. may activate additional behavior if recognized by the converter

The `role` attribute in AsciiDoc always get mapped to the `class` attribute in the HTML output. In other words, role names are synonymous with HTML class names, thus allowing output elements to be identified and styled in CSS using class selectors (e.g., `sidebarblock.role1`).

Assign roles to blocks

You can assign roles to blocks using the shorthand dot (`.`) syntax or the longhand (`role=`) syntax.

Shorthand role syntax for blocks

To assign a role to a block, prefix the value with a dot (`.`) in style style position of an attribute list. The dot implicitly sets the `role` attribute.

Example 38. Sidebar block with a role assigned using the shorthand dot

```
[.rolename]
****
This is a sidebar with a role assigned to it, rolename.
****
```

You can assign multiple roles to a block by prefixing each value with a dot (`.`).

Example 39. Sidebar with two roles assigned using the shorthand dot

```
[.role1.role2]
****
This is a sidebar with two roles assigned to it, role1 and role2.
****
```

The role values are turned into a space-separated list of values, `role1 role2`.

Formal role syntax for blocks

You can define the roles using a named attribute instead, which is the longhand syntax for adding roles to an element. When using this syntax, add the attribute name `role` followed by the equals sign (`=`) then the role name or names to any position in the block attribute list.

Example 40. Sidebar block with a role assigned using the formal syntax

```
[role=rolename]
****
This is a sidebar with one role assigned to it, rolename.
****
```

Separate multiple role values using spaces. Since the value has spaces, it's easier to read if enclosed in quotes, though the quotes are not strictly required.

Example 41. Sidebar with two roles assigned using the formal syntax

```
[role="role1 role2"]
****
This is a sidebar with two roles assigned to it, role1 and role2.
****
```

In this form, the value of the role attribute is already in the right form to be passed through to the output. No additional processing is done on it.

This longhand syntax can also be used on inline macros, but it cannot be used with formatted (aka quoted) text.

Assign roles to formatted inline elements

You can assign roles to inline elements that are enclosed in formatting syntax, such as bold (`*`), italic (`_`), and monospace (`` ` ``). To assign a role to an inline element that's enclosed in formatting syntax block, prefix the value with a dot (`.`) in an attribute list.

Example 42. Inline role assignments using shorthand syntax

```
This sentence contains [.application]*bold inline content* that's assigned a role.

This sentence contains [.varname]`monospace text` that's assigned a role.
```

The HTML source code that is output from Example 42 is shown below.

Example 43. HTML source code produced by Example 42

```
<p>This sentence contains <strong class="application">bold inline content</strong>
that&#8217;s assigned a role.</p>
```

```
<p>This sentence contains <code class="varname">monospace text</code> that&#8217;s
assigned a role.</p>
</div>
```

As you can see from this output, roles in AsciiDoc are translated to CSS class names in HTML. Thus, roles are an ideal way to annotated elements in your document so you can use CSS to uniquely style them.

The role is often used on a phrase to represent semantics you might have expressed using a dedicated element in DocBook or DITA.

Options Attribute

The `options` attribute (often abbreviated as `opts`) is a versatile named attribute that can be assigned one or more values. It can be defined globally as document attribute as well as a block attribute on an individual block.

There is no strict schema for options. Any options which are not recognized are ignored.

Assign options to blocks

You can assign one or more options to a block using the shorthand or formal syntax for the `options` attribute.

Shorthand options syntax for blocks

To assign an option to a block, prefix the value with a percent sign (`%`) in an attribute list. The percent sign implicitly sets the `options` attribute.

Example 44. Sidebar block with an option assigned using the shorthand dot

```
[%option]
****
This is a sidebar with an option assigned to it, named option.
****
```

You can assign multiple options to a block by prefixing each value with a percent sign (`%`).

Example 45. Sidebar with two options assigned using the shorthand dot

```
[%option1%option2]
****
This is a sidebar with two options assigned to it, named option1 and option2.
****
```

For instance, consider a table with the three built-in option values, `header`, `footer`, and `autowidth`, assigned to it. Example 46 shows how the values are assigned using the shorthand notation.

Example 46. Table with three options assigned using the shorthand syntax

```
[%header%footer%autowidth,cols=2*~]
|===
|Cell A1 |Cell B1

|Cell A2 |Cell B2

|Cell A3 |Cell B3
|===
```

Formal options syntax for blocks

Explicitly set `options` or `opts`, followed by the equals sign (`=`), and then the value in an attribute list.

Example 47. Sidebar block with an option assigned using the formal syntax

```
[opts=option]
****
This is a sidebar with an option assigned to it, named option.
****
```

Separate multiple option values with commas (`,`).

Example 48. Sidebar with three options assigned using the formal syntax

```
[opts="option1,option2"]
****
This is a sidebar with two options assigned to it, option1 and option2.
****
```

Let's revisit the table in Example 46 that has the three built-in option values, `header`, `footer`, and `autowidth`, assigned to it using the shorthand notation (`%`). Instead of using the shorthand notation, Example 49 shows how the values are assigned using the formal syntax.

Example 49. Table with three options assigned using the formal syntax

```
[cols=2*~,opts="header,footer,autowidth"]
|===
|Cell A1 |Cell B1

|Cell A2 |Cell B2

|Cell A3 |Cell B3
|===
```

Using options with other attributes

Let's consider `options` when combined with other attributes. The following example shows how to structure an attribute list when you have style, role, and options attributes.

Example 50. Shorthand

```
[horizontal.properties%step] ① ② ③
property 1:: does stuff
property 2:: does different stuff
```

① The block style attribute, declared as `horizontal` in this example, is a positional attribute. A block style value is always placed at the start of the attribute list.

② `properties` is prefixed with a dot (`.`), signifying that it's assigned to the role attribute. The role and options attributes can be set in either order, i.e., `[horizontal%step.properties]`.

③ The percent sign (`%`) sets the `options` attribute and assigns the `step` value to it.

When you use the formal syntax, the positional and named attributes are separated by commas (`,`).

Example 51. Formal

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

① Like in the shorthand example, named attributes such as `role` and `options` can be set in any order in the attribute list once any positional attributes are set.

Document Header

An AsciiDoc document may begin with a document header. The document header encapsulates the document title, author and revision information, document-wide attributes, and other document metadata.

Document header structure

The optional document header is a series of contiguous lines at the start of the AsciiDoc source, after skipping any empty or comment lines. If a document has a header, *no content blocks are permitted above it*. In other words, the document must start with a document header if it has one.

The document header may not contain empty lines. The first empty line the processor encounters after the document header begins marks the end of the document header and the start of the document body.

A header typically begins with a Document Title. When a document title is specified, it may be immediately followed by one or two designated lines of content. These implicit content lines are used to assign Author Information and Revision Information to the document.

The header may contain the following elements as long as there aren't any empty lines between them:

- optional document title (a level-0 heading)
- optional author line or author and revision lines if the document title is present (should immediately follow the document title)
- optional document-wide attributes (built-in and user-defined) declared using attribute entries,
  - includes optional metadata, such as a description or keywords
- optional comment lines

Notice in Example 52 that there are no empty lines between any of the entries. In other words, the lines are contiguous.

Example 52. Common elements in a header

```
// this comment line is ignored
= Document Title ①
Kismet R. Lee <kismet@asciidoctor.org> ②
:description: The document's description. ③
:sectanchors: ④
:url-repo: https://my-git-repo.com ⑤
⑥
The document body starts here.
```

① Document title

② Author line

③ Attribute entry assigning metadata to a built-in document attribute

④ Attribute entry setting a built-in document attribute

⑤ Attribute entry assigning a value to a user-defined document attribute

⑥ The document body is separated from the document header by an empty line

There are a few attribute entries in Example 52. Each attribute entry, whether built-in or user-defined, must be entered on its own line. While attribute entries may be placed anywhere in the header, including above the document title, the preferred placement is below the title, if it's present. Since the document title is optional, it's possible for the header to only consist of attribute entries.

When does the document header end?

The first empty line in the document marks the end of the header. The next line after the first empty line that contains content is interpreted as the beginning of the document's body.

Example 53. Terminating a document header

```
= Document Title
Kismet R. Lee <kismet@asciidoctor.org>
:url-repo: https://my-git-repo.com
①
This is the first line of content in the document body. ②
```

① An empty line ends the document header.

② After the empty line, the next line with content starts the body of the document.

The first line of the document body can be any valid AsciiDoc content, such as a section heading, paragraph, table, include directive, image, etc. Any attributes defined below the first empty line are not part of the document header and will not be scoped to the entire document.

Header requirements per doctype

The header is optional when the `doctype` is `article` or `book`. A header is required when the document type is `manpage`. See the manpage doctype section for manual page (man page) requirements.

If you put content blocks above the document header when using the default article doctype, you will see the following warning:

```
level 0 sections can only be used when doctype is book
```

While this warning can be mitigated by changing the doctype to book, it may lead to a secondary warning about an invalid part. That's because the document title will be repurposed as a part title

and any lines that follow it as content blocks. If you're going to use the book doctype, you must structure your document to use Book Parts.

Header processing

The information in the document header is displayed by default when converting to a standalone document. If you don't want the header of a document to be displayed, set the `noheader` attribute in the document's header or via the 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 Skip Front Matter to learn more.

Document Title

A document title (aka doctitle) is defined in the document header, typically on the first line of the document. Like all elements of the document header, the document title is optional.

Title syntax

A document title is specified using a single equals sign (`=`), followed by a space, then the title text.

Example 54. Document with a title

```
= The Intrepid Chronicles

This adventure begins on a frigid morning.
```

In Example 54, notice the empty line between the document title and the first line of prose. That empty line is what separates the document header from the document body.

This adventure begins on a frigid morning.

Doctypes and titles

Technically, a document title is a level 0 section title (`=`). The `article` and `manpage` document types (`doctype`) can only have one level 0 section.

The `book` document type permits multiple level 0 section titles. When the `doctype` is `book`, the title of the level 0 section in the header is used as the document's title. Subsequent level 0 section titles in the document body are interpreted as part titles, unless labeled with a style.

Hide or show the document title

When converting a standalone document, the document title is shown by default. You can control whether the document title appears with the `showtitle` attribute. If you don't want the title to be shown, 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 isn't shown by default. To show the title in the embeddable document, set `showtitle` in the document header or via the CLI or API. The author and revision information isn't shown below the document title in the embeddable version of the document like it is in the standalone document, even when `showtitle` is set.

Reference the document title

The level 0 section title in a document's header, that is, its title, is automatically assigned to the document attribute `doctitle`. You can reference the `doctitle` attribute anywhere in your document and the document's title will be displayed.

Example 55. Reference the doctitle attribute

```
= The Intrepid Chronicles

{doctitle} begin on a frigid morning.
```

The Intrepid Chronicles

The Intrepid Chronicles begin on a frigid morning.

The `doctitle` attribute can also be explicitly set and assigned a value using an attribute entry in the header.

title attribute

By default, the text of the document title is used as the value of the HTML `<title>` element and main DocBook `<info>` element. You can override this behavior by setting the `title` attribute in the header with an attribute entry. If neither a level 0 section title or `doctitle` is specified in the header, but `title` is, its value is used as a fallback document title.

Subtitle

An optional subtitle can be appended to a document title.

The HTML 5 converter does not currently split the subtitle out from the document title when generating HTML from AsciiDoc. The document title is only partitioned into a main and subtitle in the output of the DocBook, EPUB 3, and PDF converters. However, the subtitle is still available via the API, so you could add support for it

> by extending the HTML 5 converter.

Subtitle syntax

When the document title contains a colon followed by a space (i.e, `: `), the text after the final colon-space sequence is treated as a subtitle.

Example 56. A document title and subtitle

```
= Main Title: Subtitle
```

The separator is searched for from the end of the text. Therefore, only the last occurrence of the separator (i.e, `: `) is used for partitioning the title.

Example 57. A document title that contains more than one colon-space sequence

```
= Main Title: Main Title Continued: Subtitle
```

Modify the title separator

You can change the title separator by specifying the `separator` block attribute explicitly above the document title. A space will automatically be appended to the separator value.

Example 58. Assign separator to the document title

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

You can also assign a separator using a document attribute `title-separator` in the header.

Example 59. Assign title-separator to the document title

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

`title-separator` can also be assigned via the CLI.

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

Partition the title using the API

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

Example 60. 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.

Example 61. Retrieving a partitioned document title with a custom separator

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

Author Information

Adding author information to your document is optional. A document's author information is assigned to multiple built-in attributes. These optional attributes can be set and assigned values using the author line or using attribute entries in a document's header.

Author and email attributes

author

The `author` attribute represents the author's full name. The attributes `firstname`, `middlename`, `lastname`, and `authorinitials` are automatically derived from the value of the `author` attribute. When assigned implicitly via the author line, the value includes all of the characters and words prior to the semicolon (`;`), angle bracket (`<`), or the end of the line. Note that when using the implicit author line, the full name can have a maximum of three space-separated names. If it has more, then the full name is assigned to the `firstname` attribute. You can adjoin names using an underscore (`_`) character.

email

The `email` attribute represents an email address or URL associated with the first author (`author`). When assigned via the author line, it's enclosed in a pair of angle brackets (`< >`). A URL can be used in place of the email address.

Name and initials attributes

firstname

The `firstname` attribute represents the first, forename, or given name of the author. The first space-separated name in the value of the `author` attribute is automatically assigned to `firstname`.

lastname

The `lastname` attribute represents the last, surname, or family name of the author. If `author` contains more than one space-separated name, the third name and any names after that are assigned to the `lastname` attribute.

middlename

The `lastname` attribute represents the middle name or initial of the author. If `author` contains

more than two space-separated names, the second name is assigned to the `middlename` attribute.

authorinitials

The first character of the `firstname`, `middlename`, and `lastname` attribute values are assigned to the `authorinitials` attribute. The value of the `authorinitials` attribute will consist of three characters or less depending on how many parts are in the author's name.

Multiple author attributes

author_<n>

An `author_<n>` attribute represents each additional author's full name, where `<n>` is the 1-based index of all of the authors listed on the author line (e.g., `author_2`, `author_3`). The attributes `firstname_<n>`, `middlename_<n>`, `lastname_<n>`, and `authorinitials_<n>` are automatically derived from `author_<n>`. Additional authors can only be assigned via the author line. Each author's full name includes all of the characters and words directly after a semicolon (`;`) but prior to the angle bracket (`<`), next semicolon (`;`), or the end of the line. The full name can have a maximum of three space-separated names. If it has more, then the full name is assigned to the `firstname_<n>` attribute. You can adjoin names using an underscore (`_`) character.

email_<n>

The `email_<n>` attribute represents an email address associated with each additional author (`author_<n>`). It's enclosed in a pair of angle brackets (`< >`) on the author line. A URL can be used in place of the email address.

firstname_<n>

The first space-separated name in the value of the `author_<n>` attribute is automatically assigned to `firstname_<n>`.

lastname_<n>

If `author_<n>` contains more than one space-separated name, the third name and any names after that are assigned to the `lastname_<n>` attribute.

middlename_<n>

If `author_<n>` contains more than two space-separated names, the second name is assigned to the `middlename_<n>` attribute.

authorinitials_<n>

The first character of the `firstname_<n>`, `middlename_<n>`, and `lastname_<n>` attribute values. The value of the `authorinitials_<n>` attribute will consist of three characters or less depending on how many parts are in the author's name.

Using the Author Line

The author attributes can be implicitly set and assigned values using the author line.

What's the author line?

The **author line** is directly after the document title line in the document header. When the content

on this line is structured correctly, the processor assigns the content to the built-in `author` and `email` attributes.

When can I use the author line?

In order for the processor to properly detect the author line and assign the content to the correct attributes, all of the following criteria must be met:

1. The header must contain a document title.
2. The author information must be entered on the line directly beneath the document title.
3. The author line must start with an author name.
4. The content in the author line must be placed in a specific order and separated with the correct syntax.

Example 62. Author line structure for single author

```
= Document Title
firstname middlename lastname <email>
```

The author's middle name is optional. An email following the author's last name is also optional. If included, the email address must be enclosed in a pair of angle brackets (`< >`).

The email can be replaced by a URL, though the value is still stored in the `email` attribute.

The author line also accepts multiple authors.

Assign an author and email

In Example 63, let's add an author and their email address using the author line. The author line must be placed on the line directly below the document title and start with an author's name.

Example 63. Add an author and email using the author line

```
= The Intrepid Chronicles
Kismet R. Lee <kismet@asciidoctor.org> ① ②
```

① Enter the author's name on the line below the document title.

② In a pair of angle brackets (`< >`), enter the author's email.

Remember, a middle name and email are optional. The processor assigns the content on the author line to the built-in attributes using word position, word count, and syntax.

The email can be replaced by a URL, though the value is still stored in the `email` attribute.

When the default stylesheet is applied, the author information is displayed on the byline. The **byline** displays the author information and the revision information directly beneath the docu-

ment's title.

The Intrepid Chronicles

Kismet R. Lee – kismet@asciidoctor.org

Using attribute references in the author line

The author line is not intended to support the 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 results. It's best to use the author line strictly as a shorthand for defining static author and email information.

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

Add Multiple Authors to a Document

The author line is the only way to assign more than one author to a document for display in the byline. Additionally, only the HTML 5 and Docbook converters can convert documents with multiple authors.

Multi-author syntax

The information for each author is concluded with a semicolon (`;`).

Example 64. Author line structure for multiple authors

```
= Document Title
firstname middlename lastname <email>; firstname middlename lastname <email>
```

Directly after each author's last name or optional email, enter a semicolon (`;`) followed by a space, and then enter the next author's information.

Escape a trailing character reference

If an author name segment ends with a character reference (e.g., `®`), you must escape it from processing. One way to escape it is to add a trailing attribute reference (e.g., `{empty}`). If the character reference appears at the end of the last author name segment, you can use a second semicolon instead.

A better way of escaping the character reference is to replace it with an attribute reference (e.g., `{reg}`).

Even if the character reference is escaped, the segments of the author name will not be processed. Instead, the whole name will be assigned to the `author` and `firstname` attributes. This limitation may be lifted in the future.

List multiple authors on the author line

The author line in Example 65 lists the information for three authors. Each author's information is separated by a semicolon (`;`). Notice that the author *B. Steppenwolf* doesn't have an email, so the semicolon is placed at the end of their name.

Example 65. An author line with three authors and two email addresses

```
= The Intrepid Chronicles
Kismet R. Lee <kismet@asciidoctor.org>; B. Steppenwolf; Pax Draeke
<pax@asciidoctor.org>
```

The result of Example 65 is displayed below.

The information for each author can also be referenced in the document using their respective built-in attribute.

If an author name ends with with a character reference, you can preserve the semicolon in the character reference by adding a trailing attribute reference:

```
AsciiDoc&#174;{empty} WG; Another Author
```

Another solution entails moving the character reference to an attribute and inserting it using an attribute reference:

```
:reg: &#174;
AsciiDoc{reg} WG; Another Author
```

Even though the character reference is escaped, the segments of the author name will not be processed.

Assign Author and Email with Attribute Entries

Instead of using an author line, a single author's information can be set and assigned with attribute entries in the document header.

author and email attribute syntax

The built-in attributes `author` and `email` can be explicitly set and assigned values in the document header using attribute entries.

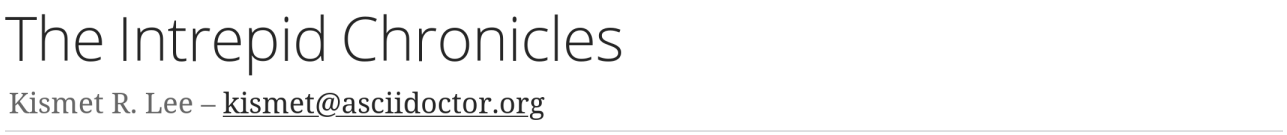
Example 66. Set author and email attributes

```
= The Intrepid Chronicles
:author: Kismet R. Lee ①
:email: kismet@asciidoctor.org ②
```

① The author's name is assigned to the built-in attribute `author`

② The author's email is assigned to the built-in attribute `email`

When the default stylesheet is applied, the author information assigned to these attributes is displayed on the byline. The result of Example 66 is displayed below.



You can't set the built-in attributes for multiple authors (e.g., `author_2`, `email_3`) using attribute entries. Multiple authors can only be set using the author line.

These attributes can also be referenced in the document.

Reference the Author Information

Referencing the author attributes

You can reference the built-in author attributes in your document regardless of whether they're set via the author line or attribute entries. In Example 67, the `author` and `email` attributes are assigned using attribute entries.

Example 67. Reference the author attributes

```
= The Intrepid Chronicles
:author: Kismet R. Lee
:email: kismet@asciidoctor.org

== About {author}

You can contact {firstname} at {email}.

P.S. Don't ask what the {middlename} stands for; it's a secret.
{authorinitials}
```

The result of Example 67 is displayed below.

The Intrepid Chronicles

Kismet R. Lee – kismet@asciidoctor.org

About Kismet R. Lee

You can contact Kismet at kismet@asciidoctor.org.

P.S. Don't ask what the R. stands for; it's a secret. KRL

Referencing information for multiple authors

The first author in an author line is assigned to the built-in attributes `author`, `email`, `firstname`, etc. Subsequent authors are assigned to the built-in author attributes, but the attribute names are appended with an underscore (_) and the numeric position of the author in the author line. For instance, the author *B. Steppenwolf* in Example 68 is the second author in the author line. The built-in attributes used to reference their information are appended with the number *2*, e.g., `author_2`, `email_2`, `lastname_2`, etc.

Example 68. Reference the built-in attributes for multiple authors

```
= The Intrepid Chronicles
Kismet R. Lee <kismet@asciidoctor.org>; B. Steppenwolf; Pax Draeke
<pax@asciidoctor.org>

.About {author_2}
Mr. {lastname_2} lives in the Rocky Mountains.

.About {author_3}
{firstname_3}, also known as {authorinitials_3}, loves to surf.

.About {author}
You can contact {firstname} at {email}.
```

The result of Example 68 is displayed below.

The Intrepid Chronicles

Kismet R. Lee – kismet@asciidoctor.org · B. Steppenwolf · Pax Draeke – pax@asciidoctor.org

About B. Steppenwolf

Mr. Steppenwolf lives in the Rocky Mountains.

About Pax Draeke

Pax, also known as PD, loves to surf.

About Kismet R. Lee

You can contact Kismet at kismet@asciidoctor.org.

Compound Author Names

When a name consists of multiple parts, such as a compound or composite surname, or a double middle name, the processor needs to be explicitly told which words should be assigned to a specific attribute.

Connecting compound author names

If the parts of an author's name aren't assigned to the correct built-in attributes, they may output the wrong information if they're referenced in the body of the document. For instance, if the name *Ann Marie Jenson* was entered on the author line or assigned to the attribute `author`, the processor would assign *Ann* to `firstname`, *Marie* to `middlename`, and *Jenson* to `lastname` based on the location and order of each word. This assignment would be incorrect because the author's first name is *Ann Marie*.

When part of an author's name consists of more than one word, use an underscore (_) between the words to connect them.

Example 69. Compound name syntax

```
= Document Title
firstname_firstname lastname; firstname middlename_middlename lastname
```

If the more than three space-separated names (or initials) are entered in the implicit author line, the entire line (including the email portion) will be used as the author's full name and first name. Thus, it's important to use the underscore separator to ensure there are no more than three space-separated names.

Compound names in the author line

In Example 70, the first author has a compound first name and the second author has a compound surname.

Example 70. Assign compound names in the author line

```
= Drum and Bass Breakbeats
Ann_Marie Jenson; Tomás López_del_Toro ① ②
```

① To signal to the processor that *Ann Marie* is the author's first name (instead of their first and middle names), type an underscore (_) between each part of the author's first name.

② The second author's last name consists of three words. Type an underscore (_) between each word of the author's last name.

The result of Example 70 is displayed below. Notice that the underscores (_) aren't displayed when the document is rendered.

Drum and Bass Breakbeats

Ann Marie Jenson · Tomás López del Toro

The underscore between each word in a compound name ensures that the parts of an author's name are assigned correctly to the corresponding built-in attributes. If you were to reference the first author's first name or the second author's last name in the document body, as shown in Example 71, the correct values would be displayed.

Example 71. Reference authors with compound names

```
= Drum and Bass Breakbeats
Ann_Marie Jenson; Tomás López_del_Toro

The first author's first name is {firstname}.

The second author's last name is {lastname_2}.
```

Like in the byline, the underscores (_) aren't displayed when the document is rendered.

Drum and Bass Breakbeats

Ann Marie Jenson · Tomás López del Toro

The first author's first name is Ann Marie.

The second author's last name is López del Toro.

Compound names in the author attribute

An underscore (_) should also be placed between each part of a compound name when the author is assigned using the `author` attribute.

Example 72. Assign a compound name using the author attribute

```
= Quantum Networks
:author: Mara_Moss Wirribi ①

== About {author}

{firstname} lives on the Bellarine Peninsula near Geelong, Australia. ②
```

① Assign the author's name to the `author` attribute. Enter an underscore (_) between each part of the author's first name. This ensures that their full first name is correct when it's automatically assigned to `firstname` by the processor.

② The built-in attribute `firstname` is referenced in the document's body. The author's first name is automatically extracted from the value of `author` and assigned to `firstname`.

The result of Example 72, displayed below, shows that the processor assigned the correct words to the built-in attribute `firstname` since the author's full first name, *Mara Moss*, is displayed where `firstname` was referenced.

Quantum Networks

Mara Moss Wirribi

About Mara Moss Wirribi

Mara Moss lives on the Bellarine Peninsula near Geelong, Australia.

Revision Information

A document's revision information is assigned to three built-in attributes: `revnumber`, `revdate` and `revremark`. These optional attributes can be set and assigned values using the revision line or using attribute entries in a document header.

Revision attributes

revnumber

The document's revision number or version is assigned to the built-in `revnumber` attribute. When assigned using the revision line, the version must contain at least one number, and, if it isn't followed by a date or remark, it must begin with the letter **v** (e.g., `v7.0.6`). Any letters or symbols preceding the number, including **v**, are dropped when the document is rendered. If `revnumber` is set with an attribute entry, it doesn't have to contain a number and the entire value is displayed

in the rendered document.

revdate

The date the revision was completed is assigned to the built-in `revdate` attribute. If the date is assigned using the revision line, it must be separated from the version by a comma (e.g., `78.1, 2020-10-10`). The date can contain letters, numbers, symbols, and attribute references.

revremark

Remarks about the revision of the document are assigned to the built-in `revremark` attribute. The remark must be separated by a colon (`:`) from the version or revision date when assigned using the revision line.

Using the Revision Line

The revision attributes can be set and assigned values using the revision line.

What's the revision line?

The **revision line** is the line directly after the author line in the document header. When the content on this line is structured correctly, the processor assigns the content to the built-in `revnumber`, `revdate` and `revremark` attributes.

When can I use the revision line?

In order for the processor to properly detect the revision line and assign the content to the correct attributes, all of the following criteria must be met:

1. The document header must contain a document title and an author line.
2. The revision information must be entered on the line directly beneath the author line.
3. The revision line must start with the revision number.
4. The revision number must contain at least one number, but a number doesn't have to be the first character in the version.
5. The values in the revision line must be placed in a specific order and separated with the correct syntax.

Example 73. Revision line structure

```
= Document Title
author <email>
revision number, revision date: revision remark
```

When using the revision line, the revision date and remark are optional.

- `v7.5` When the revision line only contains a revision number, prefix the number with a `v`.
- `7.5, 1-29-2020` When the revision line contains a version and a date, separate the version number from the date with a comma (`,`). A `v` prefix before the version number is optional.
- `7.5: A new analysis` When the revision line contains a version and a remark, separate the ver-

sion number from the remark with a colon (`:`). A `v` prefix before the version number is optional.

- `7.5, 1-29-2020: A new analysis` When the revision line contains a version, date, and a remark, separate the version number from the date with a comma (`,`) and separate the date from the remark with a colon (`:`). A `v` prefix before the version number is optional.

Assign revision information using the revision line

The revision line in Example 74 contains a revision number, date, and remark.

Example 74. Revision line with a version, date and remark

```
= The Intrepid Chronicles
Kismet Lee ①
2.9, October 31, 2021: Fall incarnation ② ③ ④
```

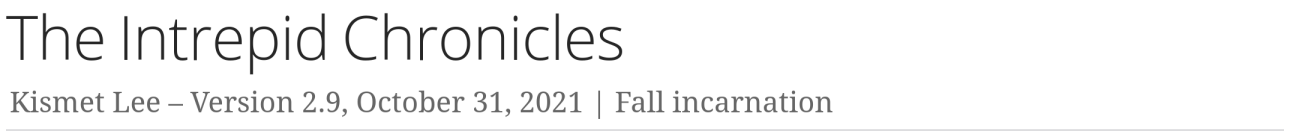
① The author line must be directly above the revision line.

② The revision line must begin with the revision number.

③ The date is separated from the version by a comma (`,`). The date can contain letters, numbers, symbols, and attribute references.

④ The remark is separated from the date by a colon (`:`).

When the default stylesheet is applied, the revision information is displayed on the same line as the author information. Note that the revision number is preceded with the word *Version*. This label is automatically added by the processor. It can be changed or turned off with the version-label attribute.



Let's look at another revision line. In Example 75, the version starts with a letter, the date is a reference to the attribute `docdate`, and there's a Unicode glyph in the remark.

Example 75. Revision line with a version prefix, attribute reference and Unicode glyph

```
= The Intrepid Chronicles
Kismet Lee
LPR55, {docdate}: A Special  Edition
```

The result of Example 75 is displayed below.

The Intrepid Chronicles

Kismet Lee – Version 55, 2020-10-25 | A Special 🎲 Edition

LPR was removed from the version because any letters or symbols that precede the revision number in the revision line are dropped. To display the letters or symbols in front of a revision number, set revnumber using an attribute entry.

Assign Revision Attributes with Attribute Entries

The revision information attributes can be set and assigned values with attribute entries.

When should I set revision attributes explicitly?

You should set the revision attributes explicitly when one of the following occurs:

- the document doesn't have an author line,
- the document doesn't have a revision number,
- you want the full value of the revision number—including any letter and symbol prefixes—to be displayed, or
- a revision attribute's value contains characters or elements that conflict with the revision line syntax.

Set the revision attributes

The attributes `revdate`, `revnumber` and `revremark` are set and assigned values in Example 76. The order of the attribute entries doesn't affect their order in the byline of a rendered document.

Example 76. Set the revision attributes in the document header

```
= The Intrepid Chronicles
:revdate: April 4, 2022
:revnumber: LPR55 ①
:revremark: The spring incarnation of {doctitle} ②
:version-label!: ③
```

① Any non-numeric characters that precede the version number aren't dropped when `revnumber` is set using an attribute entry.

② The value of `revremark` can contain attribute references.

③ The `version-label` attribute is unset so that the word *Version* isn't displayed in the byline.

The result of Example 76 is displayed below.

The Intrepid Chronicles

LPR55, April 4, 2022 | The spring incarnation of The Intrepid Chronicles

The word *Version* is absent from the rendered document's byline because the version-label attribute was unset.

Version Label Attribute

The `version-label` attribute controls the version label displayed before the revision number in the byline.

Change the version label in the byline

By default, `version-label` is assigned the value `Version`. This label can be changed by setting `version-label` and assigning it a new value in the document header.

Example 77. Assign a new label to version-label

```
= The Intrepid Chronicles
Kismet Lee
v3: An icy winter incarnation
:version-label: Edition
```

The result of Example 77 is displayed below.

The Intrepid Chronicles

Kismet Lee – Edition 3 | An icy winter incarnation

Notice that when `revnumber` is implicitly set using the revision line, any preceding letters are still removed even though `version-label` is explicitly assigned a value.

Unset the version label

You can remove the default version label from the byline by unsetting the `version-label` attribute. In an attribute entry, add a bang (`!`) to the attribute's name.

Example 78. Unset version-label

```
= The Intrepid Chronicles
Kismet Lee
v3: An icy winter incarnation
```

```
:!version-label:
```

The result of Example 78 is displayed below.

The Intrepid Chronicles

Kismet Lee – 3 | An icy winter incarnation

Reference the Revision Attributes

You can reference the revision information attributes in your document regardless of whether they're set via the revision line or attribute entries.

Reference revnumber

Remember, when `revnumber` is assigned via the revision line, any characters preceding the version number are dropped. For instance, the revision number in Example 79 is prefixed with a *v*.

Example 79. Revision line and revision attribute references

```
= The Intrepid Chronicles
Kismet Lee
v8.3, July 29, 2025: Summertime!

== Colophon

[%hardbreaks]
Revision number: {revnumber}
Revision date: {revdate}
Revision notes: {revremark}
```

The result of Example 79 below shows that the *v* in the version number has been removed when it's rendered in the byline and referenced in the document.

The Intrepid Chronicles

Kismet Lee – Version 8.3, July 29, 2025 | Summertime!

Colophon

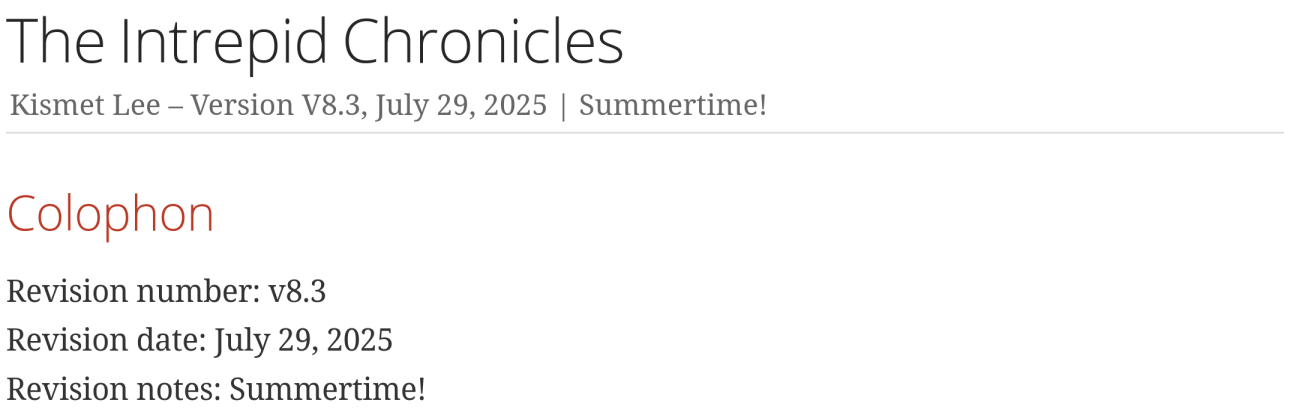
Revision number: 8.3
Revision date: July 29, 2025
Revision notes: Summertime!

To display the entire value of `revnumber` when it's referenced in the document, you must set and assign it a value using an attribute entry.

Example 80. Revision attribute entries and references

```
= The Intrepid Chronicles
Kismet Lee
:revnumber: v8.3
:revdate: July 29, 2025
:revremark: Summertime!

== Colophon

[%hardbreaks]
Revision number: {revnumber}
Revision date: {revdate}
Revision notes: {revremark}
```

The entire value of the `revnumber` from Example 80 is displayed in the byline, including the default `version-label` value *Version*. When referenced in the document, the entire value of `revnumber` is displayed because it was set with an attribute entry.



If you don't want the default version label to be displayed in the byline, unset the version-label attribute.

Document Metadata

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

Description

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

```
= The Intrepid Chronicles
Kismet Lee; Lazarus Draeke
:description: A story chronicling the inexplicable \ ①
hazards and unique challenges a team must 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>` element.

Example 81. HTML output

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<meta name="generator" content="Asciidoctor 2.0.11">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="A story chronicling the inexplicable
hazards and unique challenges a team must vanquish on their journey
to finding an open source project's true power.">
<title>The Intrepid Chronicles</title>
<style>
```

Keywords

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

```
= The Intrepid Chronicles
Kismet Lee; Lazarus Draeke
:keywords: team, obstacles, journey, victory

This journey begins on a bleary Monday morning.
```

Example 82. HTML output

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<meta name="generator" content="Asciidoctor 2.0.11">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="keywords" content="team, obstacles, journey, victory">
```

```
<title>The Intrepid Chronicles</title>
<style>
```

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.

Document Header Reference

Attribute	Values	Converters	Notes
`author`, `author_<n>`	*user-defined*	all	
`authorinitials`, `authorinitials_<n>`	**Derived from `author`**, *user-defined*	all	
`description`	*user-defined*	html	
`docinfo`		html, docbook	
`doctitle`	**Derived from level 0 section title**, *user-defined*	all	
`email`, `email_<n>`	*user-defined*	all	
`firstname`, `firstname_<n>`	**Derived from `author`**, *user-defined*	all	
`keywords`	*user-defined*	html	
`lastname`, `lastname_<n>`	**Derived from `author`**, *user-defined*	all	
`middlename`, `middlename_<n>`	**Derived from `author`**, *user-defined*	all	
`no-header-footer`, `-s`	*empty*	all	
`noheader`	*empty*	all	
`nofooter`	*empty*	all	
`notitle`	*empty*	all	
`revdate`	*user-defined*	all	
`revnumber`	*user-defined*	all	
`revremark`	*user-defined*	all	
`showtitle`	*empty*	all	
`title`	**Derived from level 0 section title or `doctitle`**, *user-defined*	html, docbook	
`version-label`	**Version**, *user-defined*	html	

Document Type

The document type (aka doctype) declares the expected structure of an AsciiDoc document. AsciiDoc defines a fixed set of document types. Each document type provides a slight variation on the permitted structure of an AsciiDoc document to accommodate different uses cases.

The default doctype is `article`, which provides the foundation structure on which other doctypes build. The `book` doctype permits multiple level-0 sections that act as part sections. The `manpage` doctype provides an extended header for defining standard metadata of a manpage, such as the volume number, man, and purpose. The `inline` doctype is intended for embedded scenarios.

Document types

Article (`article`)

The default doctype. In DocBook, this includes the appendix, abstract, bibliography, glossary, and index sections. Unless you are making a book or a man page, you don't need to worry about the doctype. The default will suffice.

Book (`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 (`manpage`)

Used for producing a roff or HTML-formatted manual page (man 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 man page. See Generate Manual Pages from AsciiDoc for details on how structure a man page using AsciiDoc and generate it using Asciidoctor.

Inline (`inline`)

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.

Inline doctype rules

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:

```
https://asciidoctor.org[AsciiDoc] is a _lightweight_ markup language...
```

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

```
<a href="https://asciidoctor.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!

Sections

Section Titles and Levels

Sections partition the document into a content hierarchy. A section is an implicit enclosure. Each section begins with a title and ends at the next sibling section, ancestor section, or end of document. Nested section levels must be sequential. A section can be a child of a document or another section, but it cannot be the child of any other block (i.e., you cannot put a section inside of a delimited block or list).

Section level syntax

A section title marks the beginning of a section and also acts as the heading for that section. The section title must be prefixed with a section marker, which indicates the section level. The number of equal signs in the marker represents the section level using a 0-based index (e.g., two equal signs represents level 1). A section marker can range from two to six equal signs and must be followed by a space.

The section title line is interpreted as paragraph text if it's found inside of a non-section block unless it marked as a discrete heading.

In the HTML output, the section title is represented by a heading tag. The number of the heading tag is one more than the section level (e.g., section level 1 becomes an h2 tag). The section level ranges from 0-5. This limit was established primarily due to the fact that HTML only provides heading tags from h1 to h6 (making level 5 the upper limit).

Example 83. 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
```

The section titles are rendered as:

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

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
```

In addition to the equals sign marker used for defining 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.

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.

Activate Section Title Links

Turn section titles into 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.

Add § to section titles

When the `sectanchors` 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 (§) that floats to the left of the section title.

Autogenerate Section IDs

Sections and discrete headings support automatic ID generation. Unless you've assigned a custom ID to one of these blocks, or you've unset the `sectids` document attribute, the AsciiDoc processor will automatically generate and assign an ID for the block using the title. This page explains how the ID is derived and how to control this behavior.

How a section ID is computed

The AsciiDoc processor builds an ID from the title using the following order of events and rules:

- 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 (not their contents), and non-word characters (except for space, hyphen, and period) are removed.
- 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.

The generated ID can be expected to be safe to use in an HTML document. However, it's important to understand that the generated ID does not necessarily conform to an NT-Name, as required by the XML specification. If you intend to produce DocBook from your AsciiDoc document(s), and your section titles uses word characters that are not permitted in an XML ID, the onus is on you to either provide an explicit ID that is conforming, or encode those invalid ID characters using a character reference (e.g., `Ⅱ`).

With those rules in mind, given the following section title:

```
== Wiley & Sons, Inc.
```

the processor will produce the following ID:

```
_wiley_sons_inc
```

You can toggle ID autogeneration on and off using `sectids` and customize the ID prefix and word separator.

If the section title contains a forward looking xref (i.e., an xref to an element that comes later in document order), you must either assign a custom ID to the block or disable ID generation around the title. Otherwise, the AsciiDoc processor may warn that the reference is invalid. This happens because, in order to generate an ID, the processor must convert the title. This conversion happens before the processor has visited the target element. As a result, the processor is not able to lookup the reference and therefore must consider it invalid.

Disable automatic section ID generation

To disable autogeneration of section and discrete heading IDs, unset the `sectids` attribute.

```
:!sectids:
```

Custom IDs are still used even when automatic section IDs are disabled.

You can unset this attribute anywhere that attribute entries are permitted in the document. By doing so, you can disable ID generation for only certain sections and discrete headings.

```
== ID generation on

:!sectids:
== ID generation off
:sectids:
```

```
== ID generation on again
```

If you disable autogenerated section IDs, and you don't assign a custom ID to a section or discrete headings, you won't be able to create cross references to that element.

Change the ID Prefix and Separator

When an AsciiDoc processor auto-generates section IDs, it begins the value with an underscore and uses a hyphen between each word. These characters can be customized with the `idprefix` and `idseparator` attributes.

Change the ID prefix

By default, the AsciiDoc processor begins an auto-generated section ID with an underscore (_). This default can cause problems when referencing the ID in an xref (either within the same file or a deep link to another file). The leading underscore may get paired with an underscore somewhere else in the paragraph, thus resulting in unexpected text formatting. One workaround is to disrupt the match by prefixing the ID with `{empty}` (e.g., `{empty}_section_title`) or using an attribute to refer to the target. Instead, we strongly encourage you to customize the ID prefix.

You can change this prefix by setting the `idprefix` attribute and assigning it a new value. The value of `idprefix` 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 empty, 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.

Change the ID word separator

The default section ID word separator is an underscore (_). You can change the separator with 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.

Assign Custom IDs and Reference Text

You can assign a custom ID and optional reference text (i.e., label) to a section (see anchor). The custom ID is used in place of the autogenerated 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:

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

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

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

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

The ID and reference text can also be defined using the block anchor syntax:

```
[[tigers-subspecies,Subspecies]]
=== Subspecies of Tiger
```

When using the block anchor syntax, the ID must conform to the XML Name rules, which means the ID must start with a letter, an underscore, or a colon.

AsciiDoc allows all valid UTF-8 characters to be used in section IDs. 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.

Assign auxiliary IDs

A section title can only have a single primary ID. However, it's possible to register auxiliary IDs on a section title for referencing from the URL using inline anchors. This feature works regardless of whether you assign an explicit (primary) ID.

If possible, you should avoid adding inline anchors on a section title. However, if you need to be able to link to that section from a URL using alternate fragment identifiers, this is what you need to use.

Here's how to register auxiliary IDs using inline anchors when using an autogenerated ID:

Example 84. Register auxiliary IDs at the beginning of the section title

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

Example 85. Register auxiliary IDs at the end of the section title

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

Where you place the inline anchor is where the anchor will end up in the output. The beginning is the preferred location.

These additional anchor points don't interfere with the declaration of the primary ID, as shown in the next example.

Example 86. Register auxiliary IDs on a section title with an explicit ID

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

These auxiliary IDs are not registered with the referencing system. That means they cannot be used for referencing the section title within the document. They are only intended for assigning auxiliary fragment identifiers to the section title so it can be referenced the from the URL using a URL fragment (aka deep linking). Only the primary ID can be used for referencing the section title within the document.

Section Numbers

Turn on section numbers

Sections aren't numbered by default. However, you can enable this feature by setting the attribute `sectnums`.

```
= Title
:sectnums:
```

When `sectnums` is set, level 1 (`==`) through level 3 (`====`) section titles are prefixed with arabic numbers in the form of *1.*, *1.1.*, etc. Section numbers can be set and unset via the document header, CLI, and API. Once you've set `sectnums`, you can reduce or increase the section levels that get numbered in the whole document with the sectnumlevels attribute. You can also control whether a section is numbered on a section by section basis.

Toggle section numbers on or off per section

The `sectnums` attribute is a unique attribute. It's a **flexible attribute**, which means it can be set and unset midstream in a document, even if it is enabled through the API or CLI. This allows you to toggle numbering on and off throughout a document.

To turn off numbering for one or more sections, insert the attribute above the section where you want numbering to cease and unset it by adding an exclamation point to the end of its name. To turn section numbering back on midstream, reset the attribute above the section where numbering should resume.

```
= Title
:sectnums:

== Numbered Section

:sectnums!:

== Unnumbered Section

== Unnumbered Section

=== Unnumbered Section

:sectnums:

== Numbered Section
```

For regions of the document where section numbering is turned off, the section numbering will not be incremented. Given the above example, the sections will be numbered as follows:

```
1. Numbered Section

Unnumbered Section

Unnumbered Section

Unnumbered Section

2. Numbered Section
```

The section number does not increment in regions of the document where section numbers are turned off.

sectnums order of precedence

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

Specify the section levels that are numbered

When `sectnums` is set, level 1 (`==`) through level 3 (`====`) section titles are numbered by default. You can increase or reduce the section level limit by setting the `sectnumlevels` attribute and assigning it the section level you want it to number. The `sectnumlevels` attribute accepts a value of 0 through 5, and it can only be set in the document header.

```
= Title
:sectnums:
:sectnumlevels: 2 ①
```

① When the `sectnumlevels` attribute is assigned a value of `2`, level 3 through 5 section titles are not numbered.

When the `doctype` is `book`, 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 (`sectnums!`). However, if your document is a multi-part book with part numbering enabled, then you'd have to set `sectnumlevels` to `-1` to disable part numbering too (the equivalent of `partnums!`).

Section Styles for Articles and Books

AsciiDoc provides built-in styles for the specialized front matter and back matter sections found in journal articles, academic papers, and books. These styled sections are referred to as **special sections**. The document type, `article` or `book`, determines which section styles are available for use.

Book section styles

The following section styles are permitted in the `book` document type:

- abstract (becomes a chapter)
- colophon
- dedication
- acknowledgments
- preface
- partintro (must be first child of part)
- appendix
- glossary
- bibliography
- index

The following styles are implied by the location of the section in the document and are thus not special sections.

- part
- chapter

Article section styles

The following section styles are permitted in the `article` document type:

- abstract
- appendix
- glossary
- bibliography
- index

Hide Special Section Titles

If supported by the converter, the title of a special section, such as the Dedication, can be turned off by setting the `notitle` option (e.g., `%notitle` or `opts=notitle`) (previously `untitled`) on the section.

```
[dedication%notitle]
== Dedication

For S.S.T.--

thank you for the plague of archetypes.
```

Although the title is hidden in the output document, it still needs to be specified in the AsciiDoc source for the purpose of referencing. The title will be used as the reftext of a cross reference, just as with any section.

Number Special Sections

Sections that are assigned a built-in special style aren't numbered by default. To number regular sections as well as special sections, set `sectnums` and assign it a value of `all`.

```
= Title
:sectnums: all
```

The assignment of appendix numbers isn't affected by `sectnums` as their section title prefixes are controlled by the attribute appendix-caption. Book parts aren't numbered by `sectnums` either, instead, they're controlled by `partnums`.

Colophon

A colophon contains factual information about the book, particularly relating to its production. It may include information such as the ISBN, publishing house, edition and copyright dates, legal notices and disclaimers, typographic style, fonts and paper used, cover art and layout credits, binding method, and any other significant production notes.

The colophon (or colophons), if present, almost always occur at the very beginning (front matter) or end (back matter) of a book. However, they can also be placed anywhere in an AsciiDoc manuscript as a top-level section. In a printed book, the colophon is often found on the verso side of the title page.

Colophon section syntax

To use the `colophon` section style, the document type must be `book`. If the book does not have parts, the colophon must be a level 1 section (`==`).

```
[colophon]
== Colophon

The Asciidoctor Press, Ceres and Denver.

(C) 2020 by The Asciidoctor Press

Published in the Milky Way Galaxy.

This book is designed by Dagger Flush, Denver, Colorado.
The types are handset Volcano Dust and Papaya, designed by Leeloo.
Leeloo created the typefaces to soften the bluntness of documentation.

Built with Asciidoctor on Fedora 33.

Printing and binding by Ceres Lithographing, Inc., Ceres, Milky Way.
```

If the book has parts, the colophon must be a level 0 section (`=`).

```
[colophon]
= Colophon

The Asciidoctor Press, Ceres and Denver.

(C) 2020 by The Asciidoctor Press

Published in the Milky Way Galaxy.

This book is designed by Dagger Flush, Denver, Colorado.
The types are handset Volcano Dust and Papaya, designed by Leeloo.
Leeloo created the typefaces to soften the bluntness of documentation.
```

```
Built with Asciidoctor on Fedora 33.

Printing and binding by Ceres Lithographing, Inc., Ceres, Milky Way.
```

The colophon will only be numbered if the `sectnums` attribute has the value `all`.

Dedication

A dedication page is used to express gratitude.

Dedication section syntax

To use the `dedication` section style, the document type must be `book`. The dedication section must be a level 1 section (`==`), unless the book has parts.

```
[dedication]
== Dedication

For S.S.T.--

thank you for the plague of archetypes.
```

If the book has parts, the dedication section must be a level 0 section (`=`).

```
[dedication]
= Dedication

For S.S.T.--

thank you for the plague of archetypes.
```

Abstract (Section)

An abstract is a concise overview of an article. The abstract section style can be used in the `article` document type.

Abstract section syntax

The `abstract` section style must be set on the first section of the article, as seen in the example below:

```
= Article Title

[abstract]
== Abstract

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

```
== Section Title
```

If you want to style a paragraph or an open block as an abstract, instead of a whole section, see the abstract block style documentation.

Abstract (Block)

An abstract is a concise overview of a document. The `abstract` block style can be placed on an open block or paragraph. Here's an example of the abstract block style set on a paragraph:

```
= Document Title

[abstract]
.Abstract
Documentation is a distillation of many long adventures.

== First Section
```

The abstract block style does not require the open block or paragraph to have a title, and it does not depend on a subsequent section to terminate it.

```
= Document Title

[abstract]
--
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 a quote block inside an abstract block.

There's also an abstract section style.

Preface

A preface is a special section that precedes the first chapter of a book or a book part.

Preface for a book

The `preface` section style can only be used when the doctype is `book`. A preface can contain subsections. When a book doesn't contain parts, the preface must be defined as a level 1 section (`==`) and

any preface subsections must start at level 2 (===).

```
= Book Title
:doctype: book

[preface]
== Our Preface

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 had eroded into several
minor foothills and a creeping alluvial plain of loose papers.

=== Preface Subsection

Chinchillas rule the world.

== Chapter 1

...
```

Preface for a book part

To create a preface for a book part, the preface must be defined as a level 1 section (==) and any subsections must start at level 2 (===). The preface must be the first section in the part.

```
= Book Title
:doctype: book

= Part 1

[preface]
== Part 1 Preface

The preface for part 1.

=== Preface Subsection

More part 1 prefacing.

== Chapter 1

...
```

Book Parts

Parts can only be used when the document type is `book`. The presence of at least one part implies that the document is a multi-part book. (There's no dedicated `doctype` for a multi-part book to distinguish it from a book with only chapters).

Anatomy of a part

A part is a level 0 section. A part must contain at least one level 1 section. The first part is the first level 0 section in the document that comes after the document title. Like the document title, a part is designated by a level 0 section title (`=`).

```
= Book Title
:doctype: book

= Part I

...
```

A part can have an optional introduction (similar to the preamble of the document), known as a part intro. The part intro is the content between the part title and the first section in the part. The part intro can be marked explicitly using the `partintro` style on either a paragraph or open block.

AsciiDoc provides document attributes to control the numbering and labeling parts.

Part intro

The content between the part title and the first section in the part is the part intro. Normally, the part intro is inferred, as shown here:

```
= Book Title
:doctype: book

= Part I

This is the implicit partintro.

== Chapter A
```

You can mark the part intro explicitly by adding the `partintro` style on the sole block before the first section.

```
= Book Title
:doctype: book

= Part I

[partintro]
This is the implicit partintro.

== Chapter A
```

Special sections for parts

A part can have its own preface, bibliography, glossary and index.

```
= Multi-Part Book with Parts that Have Special Sections
Author Name <author@example.com>
:doctype: book

[preface]
= Book Preface

This is the preface for the whole book.

=== Preface Subsection

Chinchillas rule the world.

= Part 1

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 is a preface just for part 2.

== Chapter 3

The mud had turned to cement...
```

Special sections can also be correlated directly with the book, as part siblings. Since the book preface in the previous example comes before the first part, you can write it as level 1 section if you prefer.

```
[preface]
== Book Preface

This is the preface for the whole book.

=== Preface Subsection

Chinchillas rule the world.
```

Read on to find out how to use special sections as part siblings.

Special sections as part siblings

In a multi-part book, parts occupy the top level in the hierarchy. If you were to define a special section at level 1 that follows a part in a multi-part book, it will become a child of that part. If you want the special section to be owned by the book instead, as a sibling of parts, it must be defined at the top level too.

The AsciiDoc syntax allows special sections in a multi-part book to be defined using a level 0 section title (i.e., `=`). When the document is parsed, the level of the special section will automatically be adjusted to a level 1 section in the model. Despite this level change, the special section remains as a sibling of parts in the hierarchy. The one level of offset (level 0 instead of level 1) is only a hint to the parser to make the special section a sibling of parts.

You can see this syntax used for the appendix in the following example.

```
= Multi-Part Book
:doctype: book

= Part Title

== Chapter Title

[appendix]
= Appendix Title
```

For consistency, it's best to also make special sections part-like if they come before the first part. However, technically the syntax doesn't require it.

```
= Multi-Part Book
:doctype: book

[preface]
= Book Preface

= Part Title
```

```
== Chapter Title
```

If the special section supports nested sections, the next level must be level 2 (i.e., ===), since the special section itself has level 1. Here's an example of a multi-part book that has a special section before the part and a special section with subsections after the part.

```
= Multi-Part Book with Special Sections
Author Name <author@example.com>
:doctype: book
:toc:

[colophon]
= The Colophon

Text at the beginning of a book describing facts about its production.

= The First Part

== The First Chapter

Chapters can be grouped by preceding them with a level 0 Book Part title.

[appendix]
= The Appendix

=== Basics

A multipart book can have appendixes, which should be defined at section level 0.

=== Subsections

Subsections of an appendix in a multipart book should start at level 2.
```

When you convert this document, notice that the special sections are siblings of the part in the table of contents.

Notice that the subsections of the special sections are only a single level below the parent section rather than two levels below.

Part Numbers and Signifier

Number book parts

Book part numbers are controlled by the `partnums` attribute, not `sectnums`. To autogenerate part numbers, set the `partnums` attribute in the book header.

```
= The Secret Manual
:doctype: book
:sectnums:
:partnums:

= Defensive Operations

== An Introduction to DefenseOps

= Managing Werewolves
```

When rendered, part numbers are displayed as Roman numerals.

```
Part I. Defensive Operations
1. An Introduction to DefenseOps
Part II. Managing Werewolves
```

Customize the part signifier

The signifier (i.e., prefix) *Part* is automatically added to the beginning of the part titles when the `partnums` attribute is set. The signifier is offset from the auto-generated part number by a space. You can modify the signifier by defining the `part-signifier` attribute in the header of your book.

```
:part-signifier: Component
```

To remove the prefix, unset the `part-signifier` attribute in the document header:

```
:!part-signifier:
```

Chapters

Customize the chapter signifier

If the `chapter-signifier` (i.e., prefix) is set, the value of this attribute is automatically added to the beginning of chapter titles in a book when the `sectnums` attribute is also set. The signifier is offset from the auto-generated chapter number by a space. For example, the chapter title may appear in

the output document as follows:

```
Chapter 1. Title
```

A chapter is defined as a level 1 section without a block style when the doctype type is book. You can modify the signifier by defining the `chapter-signifier` attribute in the header of your book.

```
:chapter-signifier: Peatükk
```

To remove the prefix, unset the `chapter-signifier` attribute in the document header:

```
:!chapter-signifier:
```

Appendix

The `appendix` section style can be used in books and articles, and it can have subsections. While the AsciiDoc structure allows appendices to be placed anywhere, it's customary to place them near the end of the document.

Appendix section syntax

For articles, the appendix must be defined as a level 1 section (`==`). For example:

```
= Article Title
:appendix-caption: Exhibit
:sectnums:
:toc:

== Section

=== Subsection

[appendix]
== First Appendix

=== First Subsection

=== Second Subsection

[appendix]
== Second Appendix
```

The table of contents will appear as follows:

```
1. Section
```

```
  1.1. Subsection
Exhibit A: First Appendix
  A.1. First Subsection
  A.2. Second Subsection
Exhibit B: Second Appendix
```

For books, the appendix must be defined as a level 1 section (==) if you want the appendix to be a adjacent to the chapters. 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 2 section (===).

The following example shows how to define an appendix for a multi-part book.

```
= Book Title
:doctype: book
:sectnums:
:toc:

= First Part

== Chapter

=== Subsection

== Second Part

== Chapter

[appendix]
= First Appendix

=== First Subsection

=== Second Subsection

[appendix]
= Second Appendix
```

The table of contents will appear as follows:

```
First Part
1. Chapter
  1.1. Subsection
Second Part
2. Chapter
Appendix A: First Appendix
  A.1. First Subsection
  A.2. Second Subsection
```

```
Appendix B: Second Appendix
```

Appendix label

When rendered, the titles of sections marked as `appendix` will include:

- A label, taken from the value of the `appendix-caption` attribute, which defaults to “Appendix”
- A letter that represents the order of the appendix within the sequence of appendices (A, B, …)
- A colon
- The section title

For example:

```
Appendix A: Data Access Matrix
```

The prefix can be modified by setting the `appendix-caption` attribute and overriding the default value with a custom value.

```
:appendix-caption: Exhibit
```

Unset the attribute to remove the prefix.

```
:appendix-caption!:
```

Glossary

You can include a glossary in an article, book, and book part by setting the `glossary` style on a section.

Glossary section syntax

The glossary section is defined as a level 1 section (`==`) when:

- the doctype is `article`
- the doctype is `book` and the book doesn’t contain any parts
- the glossary is for a book part

```
[glossary]
== Terminology
```

If the book has parts, and the glossary is for the whole book, the section is defined as a level 0 section (`=`).

```
[glossary]
= Glossary
```

Glossary description list style syntax

In addition to setting `glossary` on the section, the block style `glossary` must be set on the description list in the section.

```
[glossary]
== Glossary

[glossary]
mud:: wet, cold dirt
rain::
    water falling from the sky
```

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).

Bibliography section syntax

To conform to output formats, a bibliography must be its own section at any level. The section must be assigned the `bibliography` section style. By adding the `bibliography` style to the section, you implicitly add it to each unordered list in that section.

You would define the bibliography as a level 1 section (`==`) when:

- the doctype is `article`
- the doctype is `book` and the book doesn't contain any parts
- the bibliography is for a part

```
[bibliography]
== Bibliography
```

You can also define it as a deeper section, in which case the doctype doesn't matter and it's scoped to the parent section.

If the book has parts, and the bibliography is for the whole book, the section is defined as a level 0 section (`=`).

```
[bibliography]
```

```
= Bibliography
```

Bibliography entries syntax

Bibliography entries are declared as items in an unordered list.

Example 87. Bibliography with 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,gang]]] Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides.
Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. 1994.
```

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., `[[[sections:bibliography:::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>>`).

The Pragmatic Programmer [sections:bibliography:::pp] should be required reading for all developers. To learn all about design patterns, refer to the book by the “Gang of Four” [gang].

References

- [sections:bibliography:::pp] Andy Hunt & Dave Thomas. The Pragmatic Programmer: From Journeyman to Master. Addison-Wesley. 1999.
- [gang] Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. 1994.

To escape a bibliography anchor anywhere in the text, use the syntax `[[[sections:bibliography:::word]]]`. This prevents the anchor from being matched as a bibliography anchor or a normal anchor.

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., `[[[sections:bibliography:::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, `[[[sections:bibliography:::label,1]]]` will be converted to `[1]`.

If you want more advanced features such as automatic numbering and custom citation styles, try the asciidoctor-bibtex project.

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.

Index catalog

Although index terms are always processed, only Asciidoctor PDF and the DocBook toolchain support creating an index catalog automatically. The built-in HTML5 converter in Asciidoctor does not generate an index.

To create an index, define a level 1 section (`==`) marked with the style `index` at the end of your document. (In a multipart book, the index can be the last level 0 section (`=`)).

```
[index]
== Index
```

Both Asciidoctor PDF and the DocBook toolchain will automatically populate an index into this seed section.

The index will consist of term entries that link to (or otherwise cite) the location of each marked index term. You learn how to mark an index term in the next section.

Index terms

Every index term, as well as every occurrence of that index term, must be explicitly marked in the AsciiDoc document. It's not enough just to mark the first occurrence of an index term if you want every occurrence to appear in the index. Instead, each occurrence you want to be cited in the index must be marked explicitly.

There are two types of index terms in AsciiDoc:

flow index term

`indexterm2:[<primary>]`
`((<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 and is case sensitive. If you want the entry to appear in the index using a different case, use an adjacent concealed index term, such as `(((term)))Term`.

concealed index term

`indexterm:[<primary>, <secondary>, <tertiary>]`
`(((<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")))
```

Placement of hidden index terms

Hidden index entries should be directly adjacent to the paragraph content to which they apply. Example 88 shows where to place hidden index terms for a paragraph.

Example 88. Correct

```
=== Create a new Git repository

(((Repository, create)))
```

```
(((Create Git repository)))
To create a new git repository,
```

If the terms are offset from the paragraph content by an empty line, it will cause an empty paragraph to be created in the parsed document, thus leaving extra space in the generated output. Example 89 and Example 90 show where you should not place hidden index terms for a paragraph.

Example 89. Incorrect

```
=== Create a new Git repository

(((Repository, create)))
(((Create Git repository)))

To create a new git repository,
```

Example 90. Also incorrect

```
=== Create a new Git repository
(((Repository, create)))
(((Create Git repository)))

To create a new git repository,
```

Section Attributes and Styles Reference

Section attributes

Feature	Attribute	Value(s)	Notes
Appendix label	`appendix-caption`	*Appendix* (default) or user defined text	
Chapter signifier	`chapter-signifier`	*Chapter* (default) or user defined text	`book` doctype only; default only set in Asciidoctor PDF
Discrete heading	`discrete`	NA	
ID prefix	`idprefix`	`_` (default) or user defined text	Set by default
ID word separator	`idseparator`	`_` (default) or user defined character	Set by default
Part signifier	`part-signifier`	*Part* (default) or user defined text	`book` doctype only
Part numbers	`partnums`	NA	`book` doctype only
Section anchor	`sectanchors`	NA	
Section ID	`sectids`	NA	Set by default

Feature	Attribute	Value(s)	Notes
Section link	`sectlinks`	NA	
Section numbers and special section numbers	`sectnums`	empty (default) or `all`	Can be toggled on or off in the flow of the document
Section numbers level depth	`sectnumlevels`	`3` (default) when `sectnums` is set; 0 through 5	Value of `0` is the same as `sectnums!`
Hide special section title	`options` or `%`	`untitled`	DocBook only; can only be set on special sections

Section styles

Feature	Style Name	Doctypes	Converters
Abstract	`abstract`	`article`	All
Acknowledgments	`acknowledgments`	`book`	All
Appendix	`appendix`	All	All
Bibliography	`bibliography`	All	All
Colophon	`colophon`	`book`	All
Dedication	`dedication`	`book`	All
Glossary	`glossary`	All	All
Index	`index`	All	DocBook
Preface	`preface`	`book`	All

Paragraphs

The primary block type in most documents is the paragraph. That's why in AsciiDoc, you don't need to use any special markup or attributes to create paragraphs. You can just start typing sentences and that content becomes a paragraph.

This page introduces you to the paragraph in AsciiDoc and explains how to set it apart from other paragraphs.

Create a paragraph

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 an empty line, and then continue typing your content.

Example 91. Two paragraphs in an AsciiDoc document

```
Paragraphs don't require any special markup in AsciiDoc.
A paragraph is just one or more lines of consecutive text.

To begin a new paragraph, separate it by at least one empty line from the previous
paragraph or block.
```

The result of Example 91 is displayed below.

Paragraphs don't require any special markup in AsciiDoc. A paragraph is just one or more lines of consecutive text.

To begin a new paragraph, separate it by at least one empty line from the previous paragraph or block.

Hard Line Breaks

Adjacent lines of regular text in AsciiDoc are combined into a single paragraph when converted. That means you can wrap paragraph text in the source document, either at a specific column or by putting each sentence or phrase on its own line. The line breaks separating adjacent lines won't appear in the output. Instead, the line break will be (effectively) converted to a single space. (In fact, all repeating space characters are reduced to a single space, just like in HTML.)

Hard line breaks are automatically retained in literal, listing, source, and verse blocks and paragraphs.

If you want line breaks in a paragraph to be preserved, there are several techniques you can use. For any single line, you can terminate it with a space followed by a plus sign. This syntax signals to the processor to end the line in the output with a hard line break.

```
line one +
line two
```

To add this behavior to every line in the paragraph, set the `hardbreaks` option on the paragraph instead.

```
[%hardbreaks]
line one
line two
```

Alternately, you can tell the processor to preserve all line breaks in every paragraph in the document by setting the `hardbreaks-option` document attribute, though this option should be used wisely.

```
:hardbreaks-option:

line one
line two
```

To insert an empty line in the middle of the paragraph, you can use the hard line break syntax on a line by itself. This allows you to insert space between lines in the output without introducing separate paragraphs.

```
line one +
 +
line three
```

If you want the paragraph to start with a hard line break, you need to place an `{empty}` attribute reference at the start of the line. That's because a line that starts with a space has a different meaning. The `{empty}` attribute reference allows you to insert nothing at the start of the line.

```
{empty} +
line two
```

To be consistent, you can always start an empty line with `{empty}`.

```
{empty} +
line two +
{empty} +
line four
```

Note that `empty` is a built-in document attribute in AsciiDoc.

If you're writing a story with dialogue, and you want to prefix the dialogue lines with `--`, the per-

line syntax is the most appropriate choice. For example:

```
-- Come here! -- I said. +
-- What is it? -- replied Lance.
```

If you were to use the `hardbreaks` option instead, the second `--` would not only be substituted with an endash, it would also consume the preceding newline. As a result, both lines in the source would end up appearing on the same line in the output.

Inline line break syntax

To preserve a line break in a paragraph, insert a space followed by a plus sign (`+`) at the end of the line. This results in a visible line break (e.g., `<br>`) following the line.

Example 92. Line breaks preserved using a space followed by the plus sign (+)

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

The result of Example 92 is displayed below.

Rubies are red,
Topazes are blue.

hardbreaks option

To retain all of the line breaks in an entire paragraph, assign the `hardbreaks` option to the paragraph using an attribute list.

Example 93. Line breaks preserved using the hardbreaks option

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

The result of Example 93 is displayed below.

Ruby is red.
Java is beige.

hardbreaks-option attribute

To preserve line breaks in all paragraphs throughout your entire document, set the `hardbreaks-option` document attribute in the document header.

Example 94. Line breaks preserved throughout the document using the hardbreaks-option attribute

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

Rubies are red,
Topazes are blue.
```

Preamble and Lead Style

Preamble

Content between the end of the document header and the first section title in the document body is called the preamble. A preamble is optional.

Example 95. Preamble

```
= The Intrepid Chronicles

This adventure begins on a frigid morning.
We've run out of coffee beans, but leaving our office means venturing into certain
peril.
Yesterday, a colony of ravenous Wolpertingers descended from the foothills.
No one can find the defensive operations manual, and our security experts are on an
off-the-grid team-building retreat in Katchanga.

What are we going to do?

== Certain Peril

Daylight trickles across the cobblestones...
```

When using the default Asciidoctor stylesheet, if the first paragraph does not have an explicit role, it is styled as if it has the lead role. The result of Example 95 is displayed below.

The Intrepid Chronicles

This adventure begins on a frigid morning. We've run out of coffee beans, but leaving our office means venturing into certain peril. Yesterday, a colony of ravenous Wolpertingers descended from the foothills. No one can find the defensive operations manual, and our security experts are on an off-the-grid team-building retreat in Katchanga.

What are we going to do?

Certain Peril

Daylight trickles across the cobblestones…

Lead role

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

Example 96. Setting role to lead using the shorthand syntax

```
== Section title

This is a regular paragraph.

[.lead]
This is a paragraph styled as a lead paragraph.
```

The result of Example 96 is displayed below.

Section title

This is a regular paragraph.

This is a paragraph styled as a lead 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.

Example 97. 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 (`normal`) informs the CSS not to style it as a lead paragraph.

Paragraph Alignment

AsciiDoc 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`).

In Example 98, the paragraph is assigned the `text-center` role in an attribute list.

Example 98. Assign text-center to a paragraph

```
[.text-center]
This text is centered, so it must be important.
```

These built-in text alignment roles may not be honored by all converters. Though, you can expect them to be supported when the output format is either HTML or PDF.

Discrete Headings

A discrete heading is declared and styled in a manner similar to that of a section title, but:

- it's not part of the section hierarchy,
- it can be nested in other blocks,
- it cannot have any child blocks,
- it's not included in the table of contents.

In other words, it's a unique block element that looks like a section title, but is not an offshoot of a section title.

The `discrete` style effectively demotes the section title to a normal heading. Discrete headings are the closest match to headings in other markup languages such as Markdown.

To make a discrete heading, add the `discrete` attribute to any section title. 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 is designated by one to six equal signs, just like a regular section title.

Alternately, you may use the `float` attribute to identify a discrete heading. In this context, the term "float" does not refer to a layout. Rather, it means not bound to the section hierarchy. The term comes from an older version of AsciiDoc, in which discrete headings were called Floating Titles. DocBook refers to a discrete heading as a bridgehead, or free-floating heading.

Breaks

Thematic breaks

A line with three single quotation marks (i.e., `'''`), shown in Example 99, is a special macro that inserts a thematic break (aka horizontal rule). Like other block forms, the line must be offset by a preceding paragraph by at least one empty line.

Example 99. Thematic break syntax

```
'''
```

The result of Example 99 is displayed below.

Markdown-style thematic breaks

Asciidoctor recognizes Markdown thematic breaks. The motivation for this support 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, spaces between the characters is optional.

Example 100. Markdown-style thematic break syntax

```
---

- - -

***

* * *
```

Page breaks

A line with three less-than characters (i.e., `<<<`), shown in Example 101, is a special macro that serves as a hint to the converter to insert a page break. Like other block forms, the line must be offset by a preceding paragraph by at least one empty line.

Example 101. Page break syntax

```
<<<
```

A page break is only relevant for page-oriented / printable output formats such as DocBook, PDF,

and HTML in print mode.

If the page break macro falls at the top of an empty page, it will be ignored. This behavior can be overridden by setting the `always` option on the macro as shown in Example 102.

Example 102. Forced page break

```
[%always]
<<<
```

Some converters support additional options on the page break macro. For example, Asciidoctor PDF allows the page layout of the new page to be specified.

Example 103. With page layout

```
[page-layout=landscape]
<<<
```

If a converter supports columns, the page break can be converted into a column break by the addition of the `column` role.

Example 104. Column break

```
left column

[.column]
<<<

right column
```

When columns are not enabled or supported, the column break is expected to act as a page break.

Text Formatting and Punctuation

Just as we emphasize certain words and phrases when we speak, we emphasize words and phrases in text using formatting and punctuation. AsciiDoc provides an assortment of formatting marks for applying visual emphasis and typographic punctuation to your document. You can build on these basic formatting marks using built-in and user-defined roles. This page covers the formatting marks that AsciiDoc provides and the rules for applying and customizing them.

Formatting terms and concepts

Formatting marks and pairs

A **formatting mark** is a symbolic character, such as `*`, `_`, or `~`, that indicates the inline style you want the AsciiDoc converter to apply to the text. Formatting marks come in pairs.

A **formatting pair** consists of an identical opening mark and closing mark that encloses the text you want to style. The formatted text (i.e., the text enclosed by a formatting pair) can span multiple, contiguous lines.

The **opening mark** specifies where you want the style to start. The **closing mark** specifies where you want the style to end.

Formatting pairs can be nested, but they cannot be overlapped. If the pairs are overlapped, the behavior is unspecified and the AsciiDoc processor may produce malformed output.

A formatting pair is defined as either constrained or unconstrained, depending on where it's allowed to be applied. An unconstrained pair can be applied anywhere, whereas the application of a constrained pair is more restrictive.

Constrained formatting pair

When a space-like character directly precedes the text to format, and a space-like character or punctuation mark (`,`, `;`, `"`, `.`, `?`, or `!`) directly follows the text, and the text does not start or end with a space-like character, a **constrained formatting pair** can be used. A constrained pair uses a single opening mark and a single closing mark to enclose the text to be styled (e.g., `*strong*`).

For example, you use this form to format a word that stands alone,

```
That is *strong* stuff!
```

to format a sequence of words,

```
That is *really strong* stuff!
```

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

```
This stuff is *strong*!
```

As you can see, the constrained pair offers a more succinct markup at the tradeoff of having more limited (constrained) use. However, it should suffice in most cases, so the abbreviated markup is a benefit. You can think of a constrained pair as being a weaker markup hint than an unconstrained pair.

Unconstrained formatting pair

An **unconstrained formatting pair** can be used anywhere in the text. When the conditions are not met for a constrained formatting pair, the situation calls for an unconstrained formatting pair. An unconstrained pair consists of a double opening mark and a double closing mark that encloses the text to be styled (e.g., `Sara**h**`).

For example, you'd use an unconstrained pair to format one or more letters in a word.

```
The man page, short for **man**ual page, is a form of software documentation.
```

The unconstrained pair provides a more brute force approach to formatting at the tradeoff of being more verbose. You'll typically switch to an unconstrained pair when a constrained pair isn't sufficient, or when you are writing in a CJK like such as Chinese. See When should I use an unconstrained pair? for more examples of when to use an unconstrained pair.

Inline text and punctuation styles

AsciiDoc provides six inline text styles and one punctuation style that are applied solely with formatting marks.

Bold (type: strong)

Text that is bold will stand out against the regular, surrounding text due to the application of a thicker and/or darker font. Bold is useful when the text needs to catch the attention of a person visually scanning a page. The formatting mark for bold is an asterisk (*).

Italic (type: emphasis)

Text is often italicized in order to stress a word or phrase, quote a speaker, or introduce a term. Italic type slants slightly to the right, and depending on the font, may have cursive swashes and flourishes. The formatting mark for italic is an underscore (_).

Monospace (type: monospaced)

Technical content often requires text to be styled in a way that indicates a command or source code. Such text is usually emphasized using a fixed-width (i.e., monospace) font. The formatting mark for monospace is a backtick (`).

Highlight (type: mark)

Another way to draw attention to text is to highlight it. This semantic style is used for reference or notation purposes, or to mark the importance of a key subject or point. The formatting mark

for highlight is a hash (#).

Styled phrase (type: unquoted)

Adding a role to a span of text that uses the highlight formatting mark (#) converts to generic phrase that can be styled. AsciiDoc defines several built-in roles that you can use to style text, and the style/theming system of the converter can allow you to define styles for a custom role.

Subscript and superscript (type: subscript/superscript)

Subscript and superscript text is common in mathematical expressions and chemical formulas. The formatting mark for subscript is a tilde (~). The formatting mark for superscript is a caret (^).

Curved quotation marks and apostrophes (type: double/single)

By default, the AsciiDoc processor outputs straight quotation marks and apostrophes. They can be changed to curved by adding backticks (`) as a formatting hint.

Quotes substitution

When the AsciiDoc processor encounters text enclosed by designated formatting marks, those marks are replaced by the start and end tags of the corresponding HTML or XML element, depending on your backend, during the quotes substitution step. You can control when inline formatting is applied to inline text, macros, or blocks with the quotes substitution value.

Bold

Text that is marked up as bold will stand out against the regular, surrounding text due to the application of a thicker and/or darker font. Bold is useful when the text needs to catch the attention of a site visitor quickly scanning a page.

The bold presentation of text maps to the formatted text type known as **strong** in the AsciiDoc language.

Bold syntax

You can mark a word or phrase as bold by enclosing it in a single pair of asterisks (e.g., `*word*`) (constrained). You can mark bounded characters (i.e., characters within a word) as bold by enclosing them in a pair of double asterisks (e.g., `char**act**ers`) (unconstrained).

Example 105. Bold inline formatting

```
A bold *word*, and a bold *phrase of text*.

Bold c**hara**cter**s** within a word.
```

You don't need to use double asterisks when an entire word or phrase marked as bold is directly followed by a common punctuation mark, such as ;, ", and !.

The results of Example 105 are displayed below.

A bold **word**, and a bold **phrase of text**.

Bold c**hara**cter**s** within a word.

Mixing bold with other formatting

You can add multiple emphasis styles to bold text as long as the syntax is placed in the correct order.

Example 106. Order of inline formatting syntax

```
`*_monospace bold italic phrase_*` & ``**__char__**``acter``**__s__**``
```

Monospace syntax (`) must be the outermost formatting pair (i.e., outside the bold formatting pair). Italic syntax (_) is always the innermost formatting pair.

The results of Example 106 are displayed below.

`monospace bold italic phrase` & **`char`**acter**`s`**

Italic

Text is often italicized in order to stress a word or phrase, quote a speaker, or introduce a term. Italic text slants slightly to the right, and depending on the font, may have cursive swashes and flourishes.

The italic presentation of text maps to the formatted text type known as **emphasis** in the AsciiDoc language.

Italic syntax

You can emphasize (aka italicize) a word or phrase by enclosing it in a single pair of underscores (e.g., `_word_`) (constrained). You can emphasize bounded characters (i.e., characters within a word) by enclosing them in a pair of double underscores (e.g., `char__act__ers`) (unconstrained).

Example 107. Italic inline formatting

```
An italic _word_, and an italic _phrase of text_.

Italic c__hara__cter__s__ within a word.
```

You don't need to use double underscores when an entire word or phrase marked as italic is directly followed by a common punctuation mark, such as `;`, `"`, and `!`.

The result of Example 107 is rendered below.

An italic *word*, and an italic *phrase of text*.

Italic c*hara*cters within a word.

Mixing italic with other formatting

You can add multiple emphasis styles to italic text as long as the syntax is placed in the correct order.

Example 108. Order of inline formatting syntax

```
`*_monospace bold italic phrase_*` & ``**__char__**``acter``**__s__**``
```

Monospace syntax (`) must be the outermost formatting pair. Bold syntax (*) must be outside the italics formatting pair. Italic syntax is always the innermost formatting pair.

The result of Example 108 is rendered below.

`monospace bold italic phrase` & **`char`**acter**`s`**

Monospace

In AsciiDoc, a span of text enclosed in a single pair of backticks (`) 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).

The monospace presentation of text maps to the formatted text type known as **monospaced** in the AsciiDoc language.

Constrained

Here's an example:

Example 109. Constrained monospace syntax

```
"`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!_
```

The result of Example 109 is rendered below.

"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!*

Unconstrained

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

Here's an example:

```
The command will re``link`` all packages.
```

Mixed Formatting

Monospaced text can also be formatted in bold or italic or both, as long as the markup pairs are entered in the right order. The monospace markup must be the outermost formatting mark, then the bold marks, then the italic marks.

Example 110. Order of inline formatting syntax

```
`*_monospaced bold italic_*`
```

The result of Example 110 is rendered below.

_`monospaced bold italic`_

Literal Monospace

To learn how to make monospace text that's not otherwise formatted, see Literal Monospace.

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 all text substitutions by default.

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 a 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 without special formatting (i.e., as literal text).

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 pair 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+` ``). The plus characters are a shorthand for the `pass:c[]` enclosure.

Example 111 contains literal, monospaced text.

Example 111. Literal monospace syntax

```
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.

Example 112 shows literal, monospaced text that contains plus characters.

Example 112. Literal monospace syntax with + 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 Inline Passthroughs.

Text Span and Built-in Roles

Instead of applying explicit formatting to text, you can enclose a span of a text in a non-formatting element. This type of markup is referred to as a text span (formerly known as *unquoted text*). It's purpose is to allow attributes such as role and ID to be applied to unformatted text. Though those attributes can still be used to apply styles to the text.

Text span syntax

When text is enclosed in a pair of single or double hash symbols (`#`) **and** has at least one role, the role(s) will be applied to that text without adding any other implicit formatting.

If no attrlist is present, the formatting pair will be interpreted as highlighted text instead.

Example 113. Text span syntax

```
The text [.underline]#underline me# is underlined.
```

When Example 113 is converted to HTML, it translates into the following output.

Example 114. Text span HTML output

```
The text <span class="underline">underline me</span> is underlined.
```

As you can see, it's up to the stylesheet to provide styles for this element. Typically, this means you'll need to define custom inline styles that map to the corresponding class. In this case, since `underline` is a built-in role, the style is provided for you.

Built-in roles for text

The AsciiDoc language provides a handful of built-in roles you can use to provide formatting hints for the text. While these roles are often used with a text span, they can also be used with any other formatted text for which a role is accepted.

Not all converters recognize these roles, though you can expect them to at least be supported by the HTML converter.

These roles are as follows:

underline
: Applies an underline decoration to the span of text.

overline
: Applies an overline decoration to the span of text.

line-through
: Applies a line-through (aka strikethrough) decoration to the span of text.

nobreak
: Disables words within the span of text from being broken.

nowrap
: Prevents the span of text from wrapping at all.

pre-wrap
: Prevents sequences of space and space-like characters from being collapsed (i.e., all spaces are preserved).

Deprecated roles

There are several built-in roles that were once supported in AsciiDoc, but have since been deprecated. These roles include `big`, `small`, named colors (e.g., `aqua`), and named background colors (e.g., `aqua-background`). You should create your own semantic roles in place of these deprecated roles.

Highlight

Highlight syntax

When text is enclosed in a pair of single or double hash symbols (`#`), and no role is assigned to it, the text will be rendered as highlighted (aka marked) text for notation purposes.

Example 115. Highlighted style syntax

```
Mark my words, #automation is essential#.
```

When Example 115 is converted to HTML, it translates into the following output.

Example 116. Highlighted text HTML output

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

The result of Example 115 is rendered below.

Mark my words, automation is essential.

Quotation Marks and Apostrophes

This page describes how to insert curved quotation marks and apostrophes using the AsciiDoc syntax. It covers the shorthand syntax, the limitations of that syntax, and when it's necessary to input these characters directly.

Single and double quotation mark syntax

AsciiDoc does not assign special meaning to single or double quotation marks when used as constrained formatting pairs (e.g., around a word or phrase). In this case, the `'` and `"` characters are taken to be straight quotation marks (also known as dumb, vertical, or typewriter quotation marks). When an AsciiDoc processor encounters straight quotation marks in this context, it outputs them as entered.

Example 117. Single and double straight quotation marks syntax

```
In Ruby, '\n' represents a backslash followed by the letter n.
Single quotes prevent escape sequences from being interpreted.
In contrast, "\n" represents a newline.
```

The result of Example 117 is rendered below.

In Ruby, '\n' represents a backslash followed by the letter n. Single quotes prevent escape sequences from being interpreted. In contrast, "\n" represents a newline.

You can instruct the AsciiDoc processor to output curved quotation marks (also known as smart, curly, or typographic quotation marks) by adding a repurposed constrained monospace formatting pair (i.e., a pair of backticks, `) directly inside the pair of quotation marks. The combination of these two formatting pairs forms a new constrained formatting pair for producing single and double curved quotation marks.

Example 118. 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 pair of double quotes (") and a pair of backticks (`).

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

The result of Example 118 is rendered below.

> “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.”

There’s no unconstrained equivalent for producing double and single curved quotation marks. In that case, it’s necessary to input the curved quotation marks directly using the characters “, ”, ‘, and ’.

Apostrophe syntax

When entered using the ' key, the AsciiDoc processor translates a straight apostrophe directly preceded and followed by a word character, such as in contractions and possessive singular forms, as a curved apostrophe. This partial support for smart typography without any special syntax is a legacy inherited from AsciiDoc.py.

Example 119. Curved apostrophe replacement

```
Olaf's desk was a mess.
```

The result of Example 119 is rendered below.

> Olaf’s desk was a mess.

If you don’t want a straight apostrophe that’s bounded by two word characters to be rendered as a curved apostrophe, escape it by preceding it with a backslash (\).

Example 120. Escape an apostrophe

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

The result of Example 120 is rendered below.

Olaf's desk ...

An apostrophe directly bounded by two word characters is processed during the replacements substitution phase, not the inline formatting (quotes) phase. To learn about additional ways to prevent the replacements substitution, see Escape and Prevent Substitutions and Inline Passthroughs.

An apostrophe directly followed by a space or punctuation, such as the possessive plural form, is not curved by default. To render a curved apostrophe when not bounded by two word characters, mark it as you would the second half of single curved quote (i.e., `'`). This syntax for a curved apostrophe is effectively unconstrained.

Example 121. Curved apostrophe syntax

```
Olaf had been with the company since the ``'00s.
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 for Example 121 below, notice that the plural possessive apostrophe, seen trailing *werewolves*, is curved, as is the omission apostrophe before *00s*.

Olaf had been with the company since the ’00s. 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.

Possessive monospace

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

Example 122. Use a curved apostrophe with monospace in a word

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

A ``std::vector````'s size is the number of items it contains.
```

The result of Example 122 is rendered below.

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

A `std::vector`'s size is the number of items it contains.

You'll need to use a similar syntax when the last (or only) word in the monospace phrase ends in an "s" (i.e., the plural possessive form).

Example 123. Use a curved apostrophe with monospace at the end of a word

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

The result of Example 123 is below. The word *class* is rendered in monospace with a curved apostrophe at the end of it.

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

You can get the same result by inserting a typographic apostrophe immediately following a constrained formatting pair. In this case, you're able to leverage the fact that a typographic apostrophe is a punctuation character to avoid the need to resort to unconstrained formatting.

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

As you can see, it's often simpler to input the curved apostrophe directly using the character `’`. The shorthand syntax AsciiDoc provides is only meant as a convenience.

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.

Subscript and superscript syntax

Text is rendered as subscript (below the baseline) when you enclose it in a pair of tildes (`~`). Text is rendered as superscript (above the baseline) when you enclose it in a pair of carets (`^`)

Superscript and subscript have unique boundary constraints in AsciiDoc that are neither constrained nor unconstrained. Rather, they are unconstrained with the key restriction that the text must be continuous. (It may not contain spaces). This restriction is in place to avoid unexpected behavior where `~` and `^` have meaning in other contexts. It's a tradeoff to have a more predictable syntax.

Subscript

One tilde (`~`) on either side of a continuous run of text makes it subscript.

Superscript

One caret (^) on either side of a continuous run of text makes it superscript.

Example 124. Subscript and superscript syntax

```
"`Well the H~2~O formula written on their whiteboard could be part
of a shopping list, but I don't think the local bodega sells
E=mc^2^,`" Lazarus replied.
```

The result of Example 124 is rendered below.

"Well the H_2O formula written on their whiteboard could be part of a shopping list, but I don't think the local bodega sells $E=mc^2$," Lazarus replied.

If you need to include spaces in the superscript or subscript text, you must use the attribute reference `{sp}` in place of the space character.

Example 125. Superscript syntax that contains spaces

```
The deepest body of water is Deep Creek Lake.^[citation{sp}needed]^
```

To write text that makes use of more complex variations and combinations of superscript and subscript, such as in equations and formulas, you're encourages to use the stem block or inline macro.

Using Custom Inline Styles

Custom style syntax

You can assign built-in roles (e.g., `big` or `underline`) or custom roles (e.g., `term` or `required`) to any formatted text. These roles, in turn, can be used to apply styles to the text. In HTML, this is done by mapping styles to the role in the stylesheet using a CSS class selector.

Example 126. Text with built-in role

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

[.big]##O##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.

The results of Example 126 are displayed below.

Do werewolves believe in small print?

Once upon an infinite loop.

Although built-in roles such as `big` and `small` are supported by most AsciiDoc processors, it's really better to define your own semantic role names and map styles to them accordingly.

Here's how you can assign a custom role to text so you can apply your own styles to it.

Example 127. Text with custom role

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

When Example 127 is converted to HTML, the word *asciidoctor* is enclosed in a `<span>` element and the role `userinput` is used as the element's CSS class.

Example 128. HTML output

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

The following example shows how you can assign styles to elements that have this role using a CSS class selector.

```
.userinput {
  font-family: monospace;
  font-size: 1.1em;
  line-height: calc(1 / 1.1);
}
```

Troubleshoot Unconstrained Formatting Pairs

An unconstrained formatting pair is often used to format just one or a few characters in a word.

When should I use unconstrained formatting?

Consider the following questions:

1. Is there a letter, number, or underscore directly outside the opening or closing formatting marks?
2. Is there a colon, semicolon, or closing curly bracket directly before the opening formatting mark?
3. Is there a space directly inside of a formatting mark?
4. Are you writing in a CJK langauge such as Chinese?

If you answered "yes" to any of these questions, you need to use an unconstrained pair.

To help you determine whether a particular syntax pattern requires an unconstrained pair versus a constrained pair, consider the following scenarios:

Constrained or Unconstrained?

AsciiDoc	Result	Formatting Pair	Reason
`Sara__h__`	Sara*h*	Unconstrained	The letter `a` is directly adjacent to the opening mark.
`**B**old`	**B**old	Unconstrained	The `o` is directly adjacent to the closing mark.
`–**2016**`	–**2016**	Unconstrained	The `;` is directly adjacent to the opening mark.
`** bold **`	** bold **	Unconstrained	There are spaces directly inside the formatting marks.
`□□□**□**□`	□□□□□	Unconstrained	There are CJK characters directly outside the formatting marks.
`*2016*–`	**2016**–	Constrained	The adjacent `&` is not a letter, number, underscore, colon, or semicolon.
`*9*-to-*5*`	**9**-to-**5**	Constrained	The adjacent hyphen is not a letter, number, underscore, colon, or semicolon.

Unconstrained pair edge cases

There are cases when it might seem logical to use a constrained pair, but an unconstrained pair is required. 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 monospace phrase inside curved quotation marks, such as “`end points`”.

You might start with the following syntax:

```
"`end points`"
```

That only gives you “end points”. The backticks contribute to making the curved quotation marks, but the word isn’t rendered in monospace.

Adding another pair of backticks isn’t enough either.

```
"``end points``"
```

The parser ignores the inner pair of backticks and interprets them as literal characters, rendering the phrase as “`end points`”.

You have to use an unconstrained pair of monospace formatting marks to render the phrase in monospace and a constrained pair of backticks to render the quotation marks as curved. That’s three pairs of backticks in total.

Example 129. A monospace phrase inside curved quotation marks

```
"```end points````"
```

If, instead, you wanted to surround the monospace phrase with typewriter quotation marks, such as "`end points`", then you need to interrupt the curved quotation marks by applying a role to the monospace phrase or escaping the typewriter quote. For example:

Example 130. A monospace phrase inside typewriter quotation marks

```
"[.code]``end points``" or \"``end points``"
```

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

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

Rendered possessive, monospace phrase

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

Alternately, you could encode the curved apostrophe directly in the AsciiDoc source to get the same result.

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

This situation is expected to improve in the future when the AsciiDoc language switches to using a parsing expression grammar for inline formatting instead of the current regular expression-based strategy. For details, follow Asciidoctor issue #61.

Escape unconstrained formatting marks

Since unconstrained formatting marks are meant to match anywhere in the text, context free, that means you may catch them formatting text that you don’t want styled sometimes. 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. The processor will interpret the double underscore in front of *__kernel* as an unconstrained formatting mark. In the second sentence, you might expect *CB###2* and *CB###3* to be highlighted and displayed using a monospace font. 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, or
- wrap the text you don't want formatted in an inline passthrough.

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

```
:scores: __
:hash3: ###

The {scores}kernel qualifier can be used with the {scores}attribute{scores} keyword...

#`CB{hash3}2`# and #`CB{hash3}3`#
```

This works because attribute expansion is performed after text formatting (i.e., quotes substitution) in the normal substitution order.

Here's how you'd write these lines using the inline single plus macro 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. Everything between the plus symbols is escaped from interpolation (attribute references, text formatting, etc.). However, the text still receives proper output escaping for HTML special characters (e.g., `<` becomes `<`).

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.

Lists

Unordered Lists

You can make unordered lists in AsciiDoc by starting lines with a designated marker. An unordered list is a list with items prefixed with symbol, such as a disc (aka bullet).

AsciiDoc builds on the well-established convention of using either an asterisk or hyphen to identify a list item. Adjacent list items are joined into a single list. Unordered lists can be nested by varying the marker character or length (asterisk only). List items may contain attached blocks. They can also be interleaved with other types of lists.

Basic unordered list

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

```
* Edgar Allan 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. Empty lines are required before and after a list. Additionally, empty lines are permitted, but not required, between list items.

Rendered unordered list

- Edgar Allan 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 Allan Poe
* Sheri S. Tepper
* Bill Bryson
```

Rendered unordered list with a title

Kizmet's Favorite Authors

- Edgar Allan 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 Allan 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.

Forcing lists apart

If you have adjacent lists, they have the tendency to want to fuse together. To force lists apart, insert a line comment (//) surrounded by empty 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
```

This technique works for all list types. See Separating Lists for more details.

Nested unordered list

To nest an item, just add another asterisk (*) to the marker. Continue doing this 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

If you prefer, you can indent the marker an arbitrary number of spaces from the left margin. The indentation is not significant and may aid in visualizing the nesting level.

You can nest unordered lists to any depth. Keep in mind, however, that some interfaces will begin flattening lists after a certain depth. For instance, GitHub starts flattening list after 10 levels of nesting.

```
* Level 1 list item
** Level 2 list item
*** Level 3 list item
**** Level 4 list item
***** Level 5 list item
****** etc.
* Level 1 list item
```

Unordered lists can be nested to any depth

- Level 1 list item
    - Level 2 list item
        - Level 3 list item
            - Level 4 list item
                - Level 5 list item
                    - etc.
- Level 1 list item

Determining list depth

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. For example, you can create a second level using the hyphen marker instead of two asterisks.

Example 131. Using hyphen to mark the second level is not recommended

```
* Level 1 list item
- Level 2 list item
* Level 1 list item
```

However, it's much more intuitive to follow the convention that the marker length (i.e., number of asterisks) equals the level of nesting. The hyphen should only be used as the marker for the first

level.

marker length = level of nesting

After all, we're shooting for plain text markup that is readable *as is*.

Markers

When rendered, an unordered list item is designated by a leading marker (bullet) (not to be confused with the marker used to define the list). This marker can be controlled using the list style. If no marker is specified, a default marker will be selected by the renderer.

Default markers

By default, AsciiDoc assumes that the first three levels of an unordered list will be styled using the markers disc, circle, and squared when rendered. Consider the following list:

```
* disc
** circle
*** square
```

Default alternating markers for nested lists

- disc
  - circle
    - square

Observe that the marker for the first level is a disc (filled circle), the second level is a circle (outline), and the third level is a square (filled). The AsciiDoc processor does not specify these markers explicitly in the model or converted output. Rather, these defaults are added by the renderer (e.g., CSS), adhering to a convention established by HTML.

Beyond the third level of nesting, the marker choice is not specified. Typically, the renderer will continue to use the square marker, as shown in an earlier example.

Custom markers

AsciiDoc offers numerous marker styles for lists. The list marker can be specified 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) (not supported in DocBook output)

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 markers:

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

A list with square markers

- one
- two
- three

Once the list style is set, that style is used for all nested lists until it is set again. The assumption is that it's no longer possible to infer the alternation, so it stops. The inherited style is not specified in the model, but rather applied by the renderer (e.g., CSS). For example, if we set the list style to circle on the top-level list, it will be used for all levels.

```
[circle]
* circles
** all
*** the
**** way
***** down
```

The list style is inherited once set

- circles
  - all
    - the
      - way
        - down

The inherited style can be set or reset at any level.

```
[square]
* squares
** up top
[circle]
*** circles
**** down below
```

The list style can be reset

- squares
  - up top
    - circles
      - down below

Ordered Lists

Basic ordered list

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 number the ordered list explicitly, you have to manually keep the list numerals sequential. Otherwise, you will get a warning. This differs from other lightweight markup languages. But there's a reason for it.

Using explicit numbering is one way to adjust the numbering offset of a list (only supported in Asciidoctor 2.1.0 or better). For instance, you can type:

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

However, there's a simpler way to accomplish the same result without the manual effort. You can use the `start` attribute on the list to define the number at which you want the numerals to start.

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

The start value is always a positive integer value, even when using a different numeration style such as loweralpha.

When not to use the start attribute

When an ordered list item contains block content—such as an image, source block, or table—you may observe that the number of the next item in the list resets to 1. In fact, what's happened is that a new list has been started where the number resets due to a missing list continuation.

In these cases, you should not resort to using the `start` attribute to fix the numbering. Not only does that require manual adjustment as items are added to the list, it doesn't address the underlying semantics problem, which is what is causing it to be broken. Instead, use a list continuation between each block element you want to attach to the list item to ensure the list item is continuous. The list continuation glues the blocks together within a given item and keeps them at the same level of indentation.

- For details on how to use a list continuation, refer to the Compound List Items page.
- For an example of the list continuation used in an ordered list, see the launch steps in this .adoc file in GitHub.
- To see how those launch steps look in the final output, see the Launch the Quick Start section of the generated deployment guide. The list continuations prevent step 2 from resetting to 1. They also prevent step 5, which is pulled in from a separate AsciiDoc file, from resetting to 1.

To present list 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

Nested ordered list

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 the convention that the number of dots equals the level of nesting.

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. The AsciiDoc syntax tries 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 page demonstrates how to combine all three list types.

Number styles

For ordered lists, AsciiDoc 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:

Block style	CSS list-style-type
arabic	decimal
decimal [1]	decimal-leading-zero
loweralpha	lower-alpha
upperalpha	upper-alpha
lowerroman	lower-roman
upperroman	upper-roman
lowergreek [1]	lower-greek

[1] 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] [2]	[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

[2] Default numeration if block style is not specified

Custom numeration 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 numeration applied.

When the role shorthand (`.custom`) is used on an ordered list, the numeration 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 numeration style. For instance, to start an alphabetic list at letter "c", set the numeration style to loweralpha and the start attribute to 3.

Escaping the list marker

If you have paragraph text that begins with a list marker, but you don't intend it to be a list item, you need to escape that marker by using the attribute reference to disrupt the pattern.

Consider the case when the line starts with a P.O. box reference:

```
P. O. Box
```

In order to prevent this paragraph from being parsed as an ordered list, you need to replace the first space with `{empty}`.

```
P.{empty}O. Box
```

Now the paragraph will remain as a paragraph.

In the future, it will be possible to escape an ordered list marker using a backslash, but that is not currently possible.

Checklists

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:

Example 132. Checklist syntax

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

The result of Example 132 is displayed below.

- ☑ checked
- ☑ also checked
- ☐ not checked

  normal list item

Not all items in the list have to be checklist items, as Example 132 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 from AsciiDoc is typically 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 the Options Attribute):

Example 133. Checklist with interactive checkboxes

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

The result of Example 133 is displayed below.

- ☑ checked
- ☑ also checked
- ☐ not checked

  normal list item

Separating Lists

In AsciiDoc, lists items have natural affinity towards one another. If adjacent lines start with the same list marker, they are joined together into the same list, even if separated by empty lines. If the adjacent line starts with a different list marker, even if offset by empty lines, it will be placed into a nested list.

These rules make it easier to keep list items together in a single list. However, they can present a challenge if what you want is to create separate lists. Fortunately, there are ways to force a change to this behavior. The techniques described on this page work for all list types.

Using a line comment

To force lists apart, you can insert a line comment (i.e., `//`) surrounded by empty lines on either side between the two lists.

Here's an example that shows where to place the line comment to separate two adjacent unordered lists. The `-` following the line comment prefix is a hint to authors to indicate that the comment line serves as an "end of list" marker:

```
* Apples
* Oranges

//-

* Walnuts
* Almonds
```

This technique works for separating any type of list.

Using a block attribute line

Another way to start a new list is to place a block attribute line (even an empty one) above the second list, offset by an empty line.

Here's an example that shows where to place the block attribute line to separate unordered and ordered lists that are adjacent.

```
* Apples
* Oranges

[]
. Wash
. Slice
```

The preceding empty line is important. If that were not present, the ordered list would still be nested inside the ordered list. If the second list requires block attributes, you can add them to the block attribute line.

This technique works for separating any type of list.

Compound List Items

This page covers how to create lists that have compound list items.

A **compound list item** is a list item that has blocks attached to it, including paragraphs, which follow the (optionally empty) principal text. In other words, the list item contains block content. This scenario is different from a list item whose principal text merely spans multiple lines, a distinction which is further explained on this page. The page goes on to explain how to attach a block to a list item in an ancestor list.

In additional to unordered and ordered lists, callout and description lists also support compound list items. On this page, the term list item refers to any list item in an unordered, ordered, callout, and description list. For a description list, it refers specifically to the description of the list item (not the list item term).

The main focus of the syntax covered on this page is to keep the list continuous (i.e., to prevent the list from breaking).

Multiline principal text

As with regular paragraph text, the principal text in a list item can span any number of lines as long as those lines are contiguous (i.e., adjacent with no empty lines). Multiple lines are combined into a single paragraph and wrap as regular paragraph text. This behavior holds even if the lines are indented, as shown in the third bullet in this example:

```
* The document header in AsciiDoc is optional.
If present, it must start with a document title.

* Optional author and revision information lines
immediately follow the document title.

* The document header must be separated from
  the remainder of the document by one or more
  empty lines and it cannot contain empty lines.
```

- The document header in AsciiDoc is optional. If present, it must start with a document title.
- Optional author and revision information lines immediately follow the document title.
- The document header must be separated from the remainder of the document by one or more empty lines and it cannot contain empty lines.

When list items contain more than one line of text, leave an empty line between items to make the list easier to read while working in the code. An empty line between two list items will not break the list.

Empty lines in a list

Empty lines between two items in a list (ordered or unordered) will not break the list. For ordered lists, this means the numbering will be continuous rather than restarting at 1. (See Separating Lists to learn how to force two adjacent lists apart).

If an empty line after a list item is followed by the start of a block, such as a paragraph or delimited block rather than another list item, the list will terminate at this point. If this happens, you'll notice that a subsequent list item will be placed into a new list. For ordered lists, that means the numbering will restart (at 1).

To keep the list continuous in those cases—such as when you're documenting complex steps in a procedure—you must use a list continuation to attach blocks to the list item. For ordered lists, this will ensure that the numbering continues from one list item to the next rather than being reset.

Attach blocks using a list continuation

In addition to the principal text, a list item may contain block elements, including paragraphs, delimited blocks, and block macros. To add block elements to a list item, you must "attach" them (in a series) using a list continuation. This technique works for unordered and ordered lists as well as callout and description lists.

A **list continuation** is a `+` symbol on a line by itself, immediately adjacent to the block being attached. The attached block must be left-aligned, just like all blocks in AsciiDoc.

A `+` at the end of a line, rather than on a line by itself, is not a list continuation. Instead, it creates a hard line break.

Here's an example of a list item that uses a list continuation:

```
* 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 a list continuation, you can attach any number of block elements to a list item. Unless the block is inside a delimited block which itself has been attached, 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 a paragraph to the first list 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 lines immediately follow the document
```

```
title.
+
----
= Document Title
Doc Writer <doc.writer@asciidoc.org>
v1.0, 2022-01-01
----
```

Here's how the source is rendered:

A list with compound 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 lines immediately follow the document title.

  ```
  = Document Title
  Doc Writer <doc.writer@asciidoc.org>
  v1.0, 2022-01-01
  ```

Notice that we inserted an empty line after the attached paragraph block. That's because only a sibling list item can interrupt a paragraph. If the next list item had been a nested list item instead of a sibling, this empty line would have been required. Otherwise, the nested list marker and text would have just become the next line of the paragraph. For consistency, a best practice is to always include an empty line at the end of a compound list item. That way, you never have to remember when it's required.

A sibling or nested list item acts as an interrupting line for the principal text of a list item. Only a sibling list item acts as an interrupting line for an attached block, such as a paragraph. (The AsciiDoc Language working group has decided that the latter exception will be removed, so it's best not to depend on it.)

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 compound 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 compound 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 (yet) be nested.

Drop the principal text

If the principal text of a list item is empty, 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 empty by using the `{empty}` attribute reference.

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

```
. {empty}
+
----
```

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

Here's how the source is rendered:

A list with compound content

1. ```
   print("one")
   ```

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

Attach blocks to an ancestor list

Instead of attaching a block to the current list item, you may need to end that list and attach a block to its ancestor instead. There are two ways to express this composition in the AsciiDoc syntax. You can either enclose the child list in an open block, or you can use insert empty lines above the list continuation to first escape from the nesting. Let's look at enclosing the child list in an open block first, since that is the preferred method.

Enclose in open block

If you plan to attach blocks to a list item as a sibling of a nested list, the most robust way of creating that structure is to enclose the nested list in an open block. That way, it's clear where the nested list ends and the current list item continues.

Here's an example of a list item with a nested list followed by an attached paragraph. The open block makes the boundaries of the nested list clear.

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

Here's how the source is rendered:

A nested list enclosed in an open block

- grandparent list item
  - parent list item
    - child list item

  paragraph attached to grandparent list item

The main limitation of this approach is that it can only be used once in the hierarchy (i.e., it can only enclose a single nested list). That's because the open block itself cannot be nested. If you require more control, then you must use the ancestor list continuation.

Ancestor list continuation

Normally, a list continuation will attach a block to the current list item. For each empty line you add before the list continuation, the association will move up one level in the nesting. In other words, an empty line signals to the list continuation to back out of the current list by one level. As a result, the block will be attached to the current item in an ancestor list. This syntax is referred to as an **ancestor list continuation**.

The ancestor list continuation is a fragile syntax. For one, it may not be apparent to new authors that the empty lines before the list continuation are significant. That's because the AsciiDoc syntax generally ignores repeating empty lines. There are also scenarios where even these empty lines are collapsed, thus preventing the ancestor list continuation from working as expected. Use this feature of the syntax with caution. If possible, enclose the nested list in an open block, as described in the previous section.

Here's an example of a paragraph that's attached to the parent list item after the nested list ends. The empty line above the list continuation indicates that the block should be attached to current list item in the parent list.

```
* 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 empty 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 empty 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

Summary

On this page, you learned that the principal text of a list item can span multiple contiguous lines, and that those lines can be indented for readability without affecting the output. You learned that you can attach any type of block content to a list item using the list continuation. You also learned that using this feature in combination with the open block makes it easier to create list items with compound content, to attach blocks to a parent list, or to drop the principal text. Finally, you learned that you can use the ancestor list continuation to attach blocks to the current item in an ancestor list, and the risks with doing so.

Description Lists

A description list (often abbreviated as dlist in AsciiDoc) is an association list that consists of one or more terms (or sets of terms) that each have a description. This list type is useful when you have a list of terms that you want to emphasize and describe with text or other supporting content.

You may know this list variation by the antiquated term *definition list*. The preferred term is now *description list*, which matches the terminology used by the HTML specification.

Anatomy

A description list item marks the beginning of a description list. Each item in a description list consists of:

- one or more terms, each followed by a term delimiter (typically a double colon, `::`, unless the list is nested)
- one space or newline character
- the description in the form of text, attached blocks, or both

If a term has an anchor, the anchor must be defined at the start of the same line as the term.

The first term defines which term delimiter is used for the description list. The terms for the remaining entries at that level must use the same delimiter.

The valid set of term delimiters is fixed. When the term delimiter is changed, that term begins a new, nested description list (similar to how ordered and unordered lists work). The available term delimiters you can use for this purpose are as follows:

- `::`
- `:::`
- `::::`
- `;;`

There's no direct correlation between the number of characters in the delimiter and the nesting level. Each time you change delimiters (selected from this set), it introduces a new level of nesting. This is how list depth is implied in a language with a left-aligned syntax. It's customary to use the delimiters in the order shown above to provide a hint that the list is nested at a certain level.

Basic description list

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 label 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.

Mixing lists

The content of a description list can be any AsciiDoc element. For instance, we could split up a grocery list by aisle, using description list terms for the aisle names.

```
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
```

Nested description list

Finally, you can mix and match the three list types within a single hybrid list. The AsciiDoc syntax tries 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 lists. Notice how the term delimiter is changed from `::` to `:::` to create a nested description list.

```
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:

A hybrid list

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 compound content in a list item as well.

Horizontal Description List

If you want the first term and description of an item to start on the same line (i.e., horizontal arrangement), add the horizontal style to the description 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.

By default, the term and description columns will be sized automatically. If the content is not arranged in the way that you want, you need to adjust the width distribution.

You can control the width of the term and description columns using the (improperly named) `labelwidth` and `itemwidth` attributes on the list, respectively. Both attributes are optional. The value of each attribute is a number from 0 to 100 (a unitless percentage). If both attributes are specified, their values should add up to 100.

```
[horizontal,labelwidth=25,itemwidth=75]
A short term:: The term for this item likely fits inside the column's width.
A long term that wraps across multiple lines:: The term for this item wraps since the
width of the term column is restricted using the `labelwidth` attribute.
```

A short term
: The term for this item likely fits inside the column's width.

A long term that wraps across multiple lines
: The term for this item wraps since the width of the term column is restricted using the `labelwidth` attribute.

When converting to HTML, you can assign a role to the description list instead to control the column widths using CSS.

Question and Answer Lists

A question and answer (qanda) list is a special form of a description list that renders as an ordered list. The entries are numbered using Arabic numerals starting at 1.

Question and answer list syntax

Each entry in the description list represents one question and answer combination. The term or terms are used as the question and the description is used as the answer. If an entry has multiple questions, each question is rendered on a new line.

```
[qanda]
What is the answer?::
This is the answer.

Are cameras allowed?::
Are backpacks allowed?::
No.
```

Rendered qanda list

1. *What is the answer?*

This is the answer.

2. *Are cameras allowed?*

   Are backpacks allowed?

   No.

Description Lists With Marker

Not yet an official part of the AsciiDoc language and thus should be considered experimental.

Asciidoctor introduces a list type that's a hybrid between a description list and either an unordered or ordered list. This hybrid list, often referred to as an unordered or ordered description list, has the appearance of an unordered or ordered list, respectively, except that the subject of each list item is emphasized in bold text and either offset from the description by a stop character followed by a space or stacked above it.

Currently, only Asciidoctor PDF supports the syntax defined on this page, though support in the HTML converter is in the works. Asciidoctor EPUB 3 supports a slightly different variation in that is uses a different block style for unordered lists (`itemized` instead of `unordered`). Though that difference will be aligned in a future release.

Introduction

In an unordered and ordered description list, the first term in each item is preceded by a marker. Additional terms are ignored. The marker is a bullet for an unordered list or a number for an ordered list. The term effectively becomes the subject, appearing in bold text.

Here's an example of a description list with marker.

- **boolean:** use true and false, not 1 and 0 or T and F
- **number:** use Arabic numerals without punctuation (other than the decimal point in a floating point number)
- **enumerated value:** use only one of the allowed values, respecting case

This list type also provides control over the stop character that's inserted after the term so it can more naturally flow into the item description. It can also be configured so that the subject is stacked above the description. This page describes the syntax of this list type and how to customize its appearance.

Syntax

In AsciiDoc, a description list with a marker is defined just like a normal description list. The difference is that it must be annotated with either the `unordered` or `ordered` block style. The `unordered`

block style creates an unordered list and the `ordered` block style creates an ordered list.

Here's an example of an unordered description list.

```
[unordered]
boolean:: use true and false, not 1 and 0 or T and F
number:: use Arabic numerals without punctuation (other than the decimal point to make
a floating point number)
enumerated value:: use only one of the allowed values, respecting case
```

Here's how this syntax will appear, where supported:

- **boolean:** use true and false, not 1 and 0 or T and F
- **number:** use Arabic numerals without punctuation (other than the decimal point in a floating point number)
- **enumerated value:** use only one of the allowed values, respecting case

To make an ordered list instead, change the block style to `ordered`.

```
[ordered]
&:: ampersand
>:: greater than
```

Here's how this syntax will appear, where supported:

1. **&:** ampersand
2. **>:** greater than

Subject stop

By default, the subject (i.e., the term) is followed immediately by a colon (still in bold) and offset from the description by a space. You can replace the colon with another character (or sequence of characters) using the block attribute named `subject-stop`.

```
[unordered,subject-stop=)]
alpha:: partially feature complete, unstable, and subject to change
beta:: feature complete and undergoing testing
```

Here's how this syntax will appear, where supported:

- **alpha)** partially feature complete, unstable, and subject to change
- **beta)** feature complete and undergoing testing

If the term ends with a period or the value of the subject-stop attribute, the subject stop is not added.

To insert a space between the subject and visible stop character(s), add a space character at the start of the value of the subject-stop attribute. You'll also need to enclose the value in double quotes so the space character is preserved.

Stacked

A description list with marker uses a run-in layout by default. In other words, the subject appears on the same line as the description, separated by the subject stop and a space. To make the subject appear above the description, like in a normal description list, add the `stack` role to the list. In this case, the stop character is only added if specified explicitly.

```
[unordered.stack]
boolean:: use true and false, not 1 and 0 or T and F
number:: use Arabic numerals without punctuation (other than the decimal point to make
a floating point number)
enumerated value:: use only one of the allowed values, respecting case
```

Here's how this syntax will appear, where supported:

- **boolean**
  use true and false, not 1 and 0 or T and F
- **number**
  use Arabic numerals without punctuation (other than the decimal point in a floating point number)
- **enumerated value**
  use only one of the allowed values, respecting case

We may decide to replace the `stack` role with the `stacked` option (i.e., `%stacked`). Alternately, we may decide to reverse the default behavior and make a description list with marker stacked by default, with `run-in` as an option (i.e., `%run-in`). These adjustments will be made when this feature is standardized.

Alternatives

As an alternative to using a description list with marker, you can use a normal unordered or ordered list and format the subject and stop character manually.

```
* *boolean:* use true and false, not 1 and 0 or T and F
* *number:* use Arabic numerals without punctuation (other than the decimal point in a
floating point number)
* *enumerated value:* use only one of the allowed values, respecting case
```

This syntax gives you maximum portability in the short-term.

Although lacking proper semantics, the other way to achieve the same result is to nest a single-item description list inside an otherwise empty list item.

```
* {empty}
boolean:: use true and false, not 1 and 0 or T and F
* {empty}
number:: use Arabic numerals without punctuation (other than the decimal point in a
floating point number)
* {empty}
enumerated value:: use only one of the allowed values, respecting case
```

Links

AsciiDoc offers a variety of ways of creating links (aka hyperlinks) to other addressable resources. The pages in this section document how to add and customize links in AsciiDoc.

URLs and links

The **target** of a link is a Uniform Resource Locator (URL), otherwise known as a web address. The text the reader clicks to navigate to that target is referred to as the **link text**.

You may sometimes see the term URI used in place of a URL. Although the URI is more technically correct in some cases, URL is the accepted term.

The URL is the web address of a unique resource. A URL can either be absolute or relative. An absolute URL consists of a scheme, an authority (i.e., domain name), a path with an optional file extension, and metadata in the form of a query string and fragment (e.g., `https://example.org/asciidoc/links.html?source=home`). You may recognize an absolute URL as what you type in the location bar of a web browser, such as the one for this page. A relative URL is the portion of an absolute URL that starts after either the root path or a subpath (e.g., `guides/getting-started.html`).

Since an absolute URL has a distinct, recognizable syntax, an AsciiDoc processor will detect URLs (unless escaped) and automatically convert them to links wherever the macros substitution step is applied. This also works for bare email addresses. You can learn more about this behavior in Autolinks. To make a link to a relative URL, you must be specify it explicitly as the target of a link macro.

Link-related macros

Instead of showing the bare URL or email address as the link text, you may want to customize that text. Or perhaps you want to apply attributes to the link, such as a role. To do so, you'd use either the URL macro or, if you're linking to a complex URL, the more decisive link macro. (You can also use the link macro to make a link to an addressable resource using a relative URL or a URL that is not otherwise recognized as an absolute URL).

When linking to an email address, you can use the specialized mailto macro to enhance the link with prepopulated subject and body text.

Encode reserved characters

If the URL contains reserved characters, such as double quote (`"`), space, or an unconstrained AsciiDoc formatting mark, you'll need to encode these characters using URI encoding. For example, a double quote is encoded as `%22`. An underscore is encoded as `%5F`. If you do not encode these characters, the URL may be mangled or cause the processor to fail.

Let's assume we are creating a URL that contains a query string parameter named `q` that contains reserved characters:

```
https://example.org?q=label:"Requires docs"
```

To encode a URL, open the development tools in your browser and pass the URL to the `encodeURI` function:

```
encodeURI('http://example.org?q=label:"Requires docs"')
```

Here's the encoded URL that we'd use in the AsciiDoc document:

```
https://example.org?q=label:%22Requires%20docs%22
```

Depending on the capabilities of the web application, the space character can be encoded as `+` instead of `%20`.

Hide the URL scheme

If the link text is a bare URL (aka URI), whether that link was created automatically or using a link-related macro, you can configure the AsciiDoc processor to hide the scheme (e.g., *https://*). Hiding the scheme can make the URL more readable—perhaps even recognizable—to a person less familiar with technical nomenclature.

To configure the AsciiDoc processor to display the linked URL without the scheme part, set the `hide-uri-scheme` attribute in the header of the AsciiDoc document.

```
= Document Title
:hide-uri-scheme: ①

https://asciidoctor.org
```

① Note the use of `uri` instead of `url` in the attribute name.

When the `hide-uri-scheme` attribute is set, the above URL will be displayed to the reader as follows:

asciidoctor.org

Note the absence of *https://* in the URL. The prefix will still be present in the link target.

Autolinks

The AsciiDoc processor will detect common URLs (unless escaped) wherever the macros substitution step is applied and automatically convert them into links. This page documents the recognized URL schemes and how to disable this behavior on a case-by-case basis.

URL schemes for autolinks

AsciiDoc recognizes the following common URL schemes without the help of any markup:

- http
- https
- ftp
- irc
- mailto

The URL in the following example begins with a recognized protocol (i.e., https), so the AsciiDoc processor will automatically turn it into a hyperlink.

```
The homepage for the Asciidoctor Project is https://www.asciidoctor.org. ①
```

① The trailing period will not get caught up in the link.

The URL is also used as the link text. If you want to use custom link text, you must use the URL macro.

In plain text documents, a bare URL is often enclosed in angle brackets.

```
You'll often see <https://example.org> used in examples.
```

To accommodate this convention, the AsciiDoc processor will still recognize the URL as an autolink, but will discard the angle brackets in the output (as they are not deemed significant).

The angled brackets are also a convenient way to delinate the boundaries of the URL. If the bare URL is capturing more characters than it should, you can mark the boundaries explicitly using the angle brackets.

```
<https://google.com> -- where you find stuff
```

Without the angle brackets delinating the boundaries of the URL, the emdash gets caught up in the URL. This happens because the replacements substitution is applied before the macros substitution. So instead of the URL being followed by a space, it's followed directly by the emdash. The angle brackets ensures it does get included in the URL.

Any link created from a bare URL (i.e., an autolink) automatically gets assigned the "bare" role. This allows the theming system (e.g., CSS) to recognize autolinks (and other bare URLs) and style them distinctly.

Email autolinks

AsciiDoc also detects and autolinks most email addresses.

```
Email us at hello@example.com to say hello.
```

In order for this to work, the domain suffix must be between 2 and 5 characters (e.g., .com) and only common symbols like period (.), hyphen (-), and plus (+) are permitted. For email address which do not conform to these restriction, you can use the email macro.

Escaping URLs and email addresses

To prevent automatic linking of a URL or email address, you can add a single backslash (\) in front of it.

```
Once launched, the site will be available at \https://example.org.

If you cannot access the site, email \help@example.org for assistance.
```

The backslash in front of the URL and email address will not appear in the output. The URL and email address will both be shown in plain text.

Since autolinks are a feature of the macros substitution, another way to prevent automatic linking of a URL or email address is to turn off the macros substitution using incremental subs.

```
[subs=-macros]
Once launched, the site will be available at https://example.org.
```

The subs attribute is only recognized on a leaf block, such as a paragraph.

URL Macro

If you're familiar with the AsciiDoc syntax, you may notice that a URL almost looks like an inline macro. All that's missing is the pair of the trailing square brackets. In fact, if you add them, then a URL is treated as an inline macro. We call this a URL macro.

This page introduces the URL macro, when you would want to use it, and how it differs from the link macro.

From URL to macro

To transform a URL into a macro, add a pair of square brackets to the end of the URL. For example:

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

Since no text is specified, this macro behaves the same as an autolink. In this case, the link automatically gets assigned the "bare" role.

When the URL is followed by a pair of square brackets, the URL scheme dually serves as the macro

name. AsciiDoc recognizes all the URL schemes for autolinks as macro names (e.g., `https`). That's why we say "URL macros" and not just "URL macro". It's a family of macros. With the exception of the mailto macro, all the URL macros behave the same, and also behave the same as the link macro.

So why might you graduate from a URL to a URL macro? One reason is to force the URL to be parsed when it would not normally be recognized, such as if it's enclosed in double quotes:

```
Type "https://asciidoctor.org[]" into the location bar of your browser.
```

Another is when the URL of an autolink is being ended too soon. The URL macro allows you to explicitly mark the boundary of the URL.

```
Use http://example.com?menu=<value>[] to open to the menu named `<value>`.
```

A more common reason is to specify custom link text.

Custom link text

Instead of displaying the URL, you can configure the link to display custom text. When the reader clicks on the text, they are directed to the target of the link, the URL.

To customize the text of the link, insert that text between the square brackets of the URL macro.

```
Chat with other Fedora users in the irc://irc.freenode.org/#fedora[Fedora IRC
channel].
```

Since the text is subject to normal substitutions, you can apply formatting to it.

```
Ask questions in the https://chat.asciidoc.org[*community chat*].
```

Link attributes

You can use the attribute list to further customize the link, such as to make it target a new window and apply a role to it.

```
Chat with other AsciiDoc users in the https://chat.asciidoc.org[*project
chat*^,role=green].
```

To understand how the text between the square brackets of a URL macro is parsed, see attribute parsing.

Link Macro

The link macro is the most explicit method of making a link in AsciiDoc. It's only necessary when the behavior of autolinks and URL macros proves insufficient. This page covers the anatomy of the link macro, when it's required, and how to use it.

Anatomy

The link macro is an inline macro. Like other inline macros, its syntax follows the familiar pattern of the macro name and target separated by a colon followed by an attribute list enclosed in square brackets.

```
link:<target>[<attrlist>]
```

The `<target>` becomes the target of the link. the `<attrlist>` is the link text unless a named attribute is detected. See link macro attribute list to learn how the `<attrlist>` is parsed.

Like all inline macros, the link macro can be escaped using a leading backslash (`\`).

Link to a relative file

If you want to link to a non-AsciiDoc file that is relative to the current document, use the `link` macro in front of the file name.

To link to a relative AsciiDoc file, use the xref macro instead.

Here's an example that demonstrates how to use the link macro to link to a relative file path:

```
link:downloads/report.pdf[Get Report]
```

The AsciiDoc processor will create a link to *report.pdf* with the text "Get Report", even though the target is not a URL.

If the target file is an HTML file, and you want to link directly to an anchor within that document, append a hash (`#`) followed by the name of the anchor after the file name:

```
link:tools.html#editors[]
```

Note that when linking to a relative file, even if it's an HTML file, the link text is required.

When to use the link macro

Since AsciiDoc provides autolinks and URL macros, the link macro is not often needed. Here are the few cases when the link macro is necessary:

- The target is not a URL (e.g., a relative path)

- The target must be enclosed in a passthrough to escape characters with special meaning
- The URL macro is not bounded by spaces, brackets, or quotes.
- The target is a URL that does not start with a scheme recognized by AsciiDoc

The most common situation is when the target is not a URL. For example, you would use the link macro to create a link to a relative path.

```
link:report.pdf[Get Report]
```

If the relative path is another AsciiDoc file, you should use the xref macro instead.

You may also discover that spaces are not permitted in the target of the link macro, at least not in the AsciiDoc source. The space character in the target prevents the parser from recognizing the macro. So it's necessary to escape or encode it. Here are three ways to do it:

Example 134. Escape a space using a passthrough

```
link:pass:[My Documents/report.pdf][Get Report]
```

Example 135. Encode a space using a character reference ( )

```
link:My&#32;Documents/report.pdf[Get Report]
```

Example 136. Encode a space using URL encoding (%20)

```
link:My%20Documents/report.pdf[Get Report]
```

Escaping or encoding the space ensures that the space does not prevent the link macro from being recognized. The downside of using URL encoding is that it will be visible in the automatic link text since the browser does not decode it in that location. In this case, the character reference is preferable.

There are other characters that are not permitted in a link target as well, such as a colon. You can escape those using the same technique.

Example 137. Encode a colon using URL encoding (%3A)

```
link:Avengers%3A%20Endgame.html[]
```

Another common case is when you need to use a passthrough to escape characters with special meaning. In this case, the AsciiDoc processor will not recognize the target as a URL, and thus the link macro is necessary. An example is when the URL contains repeating underscore characters.

```
link:++https://example.org/now_this__link_works.html++[]
```

A similar situation is when the URL macro is not bounded by spaces, brackets, or quotes. In this case, the link macro prefix is required to increase the precedence so that the macro can be recognized.

```
|link:https://asciidoctor.org[]|
```

Finally, if the target is not recognized as a URL by AsciiDoc, the link macro is necessary. For example, you might use the link macro to make a file link.

```
link:file:///home/username[Your files]
```

Final word

The general rule of thumb is that you should only put the `link:` macro prefix in front of the target if the target is *not* a URL. Otherwise, the prefix just adds verbosity.

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. The AsciiDoc language plans to handle URLs more carefully in the future (see Asciidoctor issue #281), which may be solved by moving to a grammar-based parser (see Asciidoctor 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}.
```

The AsciiDoc processor 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 the double plus inline macro (`++`) around the URL only. However, when you use this approach, the AsciiDoc processor 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 Asciidoctor issue #625.

Link & URL Macro Attribute Parsing

If named attributes are detected between the square brackets of a link or URL macro, that text is parsed as an attribute list. This page explains the conditions when this occurs and how to write the link text so it is recognized as a single positional attribute.

Link text alongside named attributes

Normally, the whole text between the square brackets of a link macro is treated as the link text (i.e., the first positional attribute).

```
https://chat.asciidoc.org[Discuss AsciiDoc]
```

However, if the text contains an equals sign (`=`), the text is parsed as an attribute list. The exact rules for attribute list parsing and positional attributes are rather complex, and discussed on Positional and Named Attributes. To be sure the link text is recognized properly, you can apply these two simple checks:

- contains no comma (`,`) or equals sign (`=`) or
- enclosed in double quotes (`"`)

There are several other situations in which text before the first comma may be recognized as the link text. Let's consider some examples.

The following example shows a URL macro with custom link text alongside named attributes.

```
https://chat.asciidoc.org[Discuss AsciiDoc,role=resource,window=_blank]
```

Let's consider a case where the link text contains a comma and the macro also has named attributes. In this case, you must enclose the link text in double quotes so that it is capture in its entirety as the first positional attribute.

```
https://example.org["Google, DuckDuckGo, Ecosia",role=teal]
```

Similarly, if the link text contains an equals sign, you can enclose the link text in double quotes to ensure the parser recognizes it as the first positional attribute.

```
https://example.org["1=2 posits the problem of inequality"]
```

If the quoted link text itself contains the quote character used to enclose the text, escape the quote character in the text by prefixing it with a backslash.

```
https://example.org["href=\"#top\" attribute"] creates link to top of page
```

The double quote enclosure is not required in all cases when the link text contains an equals sign. Strictly speaking, the enclosure is only required when the text preceding the equals sign matches a valid attribute name. However, it's best to use the double quotes just to be safe.

Finally, to use named attributes without specifying link text, you simply specify the named attributes. (In other words, you leave the first positional attribute empty, in which case the target will be used as the link text).

```
https://chat.asciidoc.org[role=button,window=_blank,opts=nofollow]
```

The link macro recognizes all the common attributes (id, role, and opts). It also recognizes a handful of attributes that are specific to the link macro.

Target a separate window

By default, the link produced by a link macro will target the current window. In other words, clicking on it will replace the current page.

You can configure the link to open in a separate window (or tab) using the `window` attribute.

```
https://asciidoctor.org[Asciidoctor,window=read-later]
```

In the HTML output, the value of the `window` attribute is assigned to the `target` attribute on the `<a>` tag (e.g., `target=read-later`).

Target a blank window

Most of the time, you'll use the `window` attribute to target a blank window. Configuring a link that

points to a location outside the current site is common practice to avoid disrupting the reader's flow. To enable this behavior, you set the `window` attribute to the special value `_blank`.

```
https://asciidoctor.org[Asciidoctor,window=_blank]
```

In the HTML output, the value of the `window` attribute is assigned to the `target` attribute on the `<a>` tag (e.g., `target=_blank`). If the target is `_blank`, the processor will automatically add the `rel=noopener` attribute as well.

The underscore at the start of the value `_blank` can unexpectedly form a constrained formatting pair when another underscore appears somewhere else in the line or paragraph, thus causing the macro to break. You can avoid this problem either by escaping the underscore at the start of the value (i.e., `window=\_blank`) or by using the Blank window shorthand instead.

noopener and nofollow

The `noopener` option is used to control access to the window opened by a link. **This option is only available if the `window` attribute is set.** This option adds the `noopener` flag to the `rel` attribute on the `<a>` element in the HTML output (e.g., `rel="noopener"`).

When the value of the `window` attribute is `_blank`, the AsciiDoc processor implicitly sets the `noopener` option. Doing so is considered a security best practice.

```
https://asciidoctor.org[Asciidoctor,window=_blank]
```

If the window is not `_blank`, you need to enable the `noopener` flag explicitly by setting the `noopener` option on the macro:

```
https://asciidoctor.org[Asciidoctor,window=read-later,opts=noopener]
```

If you don't want the search indexer to follow the link, you can add the `nofollow` option to the macro. This option adds the `nofollow` flag to the `rel` attribute on the `<a>` element in the HTML output, alongside `noopener` if present (e.g., `rel="nofollow noopener"`).

```
https://asciidoctor.org[Asciidoctor,window=_blank,opts=nofollow]
```

or

```
https://asciidoctor.org[Asciidoctor,window=read-later,opts="noopener,nofollow"]
```

To fine tune indexing within the site, you can specify the `nofollow` option even if the link does not target a separate window.

```
link:post.html[My Post,opts=nofollow]
```

Blank window shorthand

Configuring an external link to target a blank window is a common practice. Therefore, AsciiDoc provides a shorthand for it.

In place of the named attribute `window=_blank`, you can insert a caret (`^`) at the end of the link text. This syntax has the added benefit of not having to worry about the underscore at the start of the value `_blank` unexpectedly forming a constrained formatting pair when another underscore appears in the same line or paragraph.

```
Let's view the raw HTML of the link:view-source:asciidoctor.org[Asciidoctor
homepage^].
```

In rare circumstances, if you use the caret syntax more than once in the same line or paragraph, you may need to escape the first occurrence with a backslash. However, the processor should try to avoid making this a requirement.

If the attribute list has both link text in double quotes and named attributes, the caret should be placed at the end of the link text, but inside the double quotes.

```
https://example.org["Google, DuckDuckGo, Ecosia^",role=btn]
```

If no named attributes are present, the link text should not be enclosed in quotes.

```
https://example.org[Google, DuckDuckGo, Ecosia^]
```

Mailto Macro

The mailto macro is a specialization of the URL macro that adds support for defining an email link with text and augmenting it with additional metadata, such as a subject and body.

Link text and named attributes

Using an attribute list, you can specify link text as well as named attributes such as `id` and `role`. Unlike other URL macros, you must add the `mailto:` prefix in front of the email address in order to append an attribute list.

Here's an example of an email link with explicit link text.

```
mailto:join@discuss.example.org[Subscribe]
```

If you want to add a role to this link, you can do so by appending the `role` attribute after a comma.

```
mailto:join@discuss.example.org[Subscribe,role=email]
```

If the link text contains a comma, you must enclose the text in double quotes. Otherwise, the portion of the text that follows the comma will be interpreted as additional attribute list entries.

```
mailto:join@discuss.example.org["Click, subscribe, and participate!"]
```

To learn more about how the attributes are parsed, refer to attribute parsing.

Subject and body

Like with other URL macros, the first positional attribute of the email macro is the link text. If a comma is present in the text, and the text is not enclosed in quotes, or the comma comes after the closing quote, the next positional attribute is treated as the subject line.

For example, you can configure the email link to populate the subject line when the reader clicks the link as follows:

```
mailto:join@discuss.example.org[Subscribe,Subscribe me]
```

When the reader clicks the link, a conforming email client will fill in the subject line with "Subscribe me".

If you want the body of the email to also be populated, specify the body text in the third positional argument.

```
mailto:join@discuss.example.org[Subscribe,Subscribe me,I want to participate.]
```

When the reader clicks the link, a conforming email client will fill in the body with "I want to participate."

If you want to reuse the email address as the link text, leave the first positional attribute empty.

```
mailto:join@discuss.example.org[,Subscribe me,I want to participate.]
```

If you only want to specify a subject, leave off the body.

```
mailto:join@discuss.example.org[,Subscribe me]
```

If either the subject or body contains a comma, that value must be enclosed in double quotes.

```
mailto:join@discuss.example.org[Subscribe,"I want to participate, so please subscribe me"]
```

To learn more about how the attributes are parsed, refer to attribute parsing.

Link, URL, and Mailto Macro Attributes Reference

These attributes apply to the link, URL, and mailto (email) macros.

Attribute	Value(s)	Example Syntax	Comments
`id`	Unique identifier for element in output	https://asciidoctor.org[Home,id=home]	
`role`	CSS classes available to inline elements	https://chat.asciidoc.org[Discuss AsciiDoc,role=teal]	
`title`	Description of link, often show as tooltip.	https://asciidoctor.org[Home,title=Project home page]	
`window`	any	https://chat.asciidoc.org[Discuss AsciiDoc,window=_blank]	The blank window target can also be specified using `^` at the end of the link text.
`window` (shorthand)	`^`	https://example.org[Google, DuckDuckGo, Ecosia^] https://chat.asciidoc.org[Discuss AsciiDoc^]	
`opts`	Additional options for link creation.	https://asciidoctor.org[Home,opts=nofollow]	Option names include: `nofollow`, `noopener`

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.

Automatic anchors

It's important to understand that many anchors are already defined for you. Using default settings, the AsciiDoc processor 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 a cross reference.

For example, considering the following section.

```
= Section Title
```

The AsciiDoc processor 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 Autogenerate Section IDs for more information about how an AsciiDoc processor 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.

Internal cross references

In AsciiDoc, the shorthand xref is used to create a cross reference to an element (e.g., section, block, list item, etc.) that has an ID within the same document. The shorthand xref is processed by the macros substitution.

If the cross reference specifies both an ID and text, the text is formatted and used as the link text. If the cross reference only specifies the ID, the reftext of the target element (typically the formatted title) is automatically used as the link text. If the element does not define reftext, a stylized form of the ID is used instead. Whether the ID is assigned explicitly on the referenced element or auto-generated does not affect how this mechanism works.

Currently, an AsciiDoc processor can resolve a cross reference to the following elements:

- Section (ID or block anchor)
- Block (ID or block anchor)
- Block macro (ID)
- Inline anchor anywhere in a paragraph

- Inline anchor at the start of a list item or table cell
- Bibliography anchor in a bibliography list

Note that the processor cannot resolve the ID assigned to a span of formatted text. If the cross reference cannot be resolved, and verbose mode is enabled, the AsciiDoc processor issues a warning about a possible invalid reference. In this case, the output document will reference the target blindly, so it's possible it will still function.

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.

Example 138. Cross reference using the ID of the target section

```
The section <<anchors>> describes how automatic anchors work.
```

The result of Example 138 is displayed below.

The section Automatic anchors describes how automatic anchors work.

Explicit link text

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.

Example 139. Cross reference with custom xreflabel text

```
Learn how to <<link-macro-attributes,use attributes within the link macro>>.
```

In this case, the target will be assumed to be an ID within the same document even if it contains a dot (`.`).

You can also use the inline xref macro as an alternative to the xref shorthand.

Example 140. Inline xref macro

```
Learn how to xref:link-macro-attributes[use attributes within the link macro].
```

However, it's best to reserve the use of the xref macro for creating interdocument cross references.

When using the xref macro, if the target contains a dot (`.`), it will be treated as a reference to another document, not an ID within the same document. If the intention is to link to an ID within

the same document, the target must be proceeded by a hash (#).

Natural cross reference

You can also create a reference to a block or section using its title rather than its ID. This type of reference is referred to as a **natural cross reference**. The title must contain at least one space character or contain at least one uppercase letter.

Example 141. Cross reference using a section's title

```
Refer to <<Internal Cross References>>.
```

As a rule of thumb, the natural cross reference should only be used for rapid development or short-lived content. As the content matures, you should switch to using IDs for referencing, ideally IDs which are declared explicitly. By doing so, it ensures your references have maximum stability and are shielded against title revisions.

Document to 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 <<anchors>> describes how automatic anchors work.
```

This cross reference creates a link to the section with the ID *anchors*.

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 xref: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 link text since an AsciiDoc processor is not (yet) required to resolve the section title in a separate document.

While the link text for a local (i.e., intradocument) cross reference is optional, the link text for an interdocument cross reference is (currently) required.

When the AsciiDoc processor 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.doc* (by comparing the xref target to the include targets relative to the outermost document). 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.doc*, 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.

Navigating between source files

In certain environments, such as a web interface for a source repository or an editor preview, you might see the generated HTML when you visit the URL of the AsciiDoc source file. If not accounted for, this has consequences for inter-document cross references.

Since the default suffix for inter-document cross references in the `html5` backend is `.html`, the resulting link created in these environments may 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.

The file extension chosen for inter-document cross references is controlled by the `relfilesuffix` attribute. By default, this attribute is not set and the value of the `outfilesuffix` is used instead. If you want to change the file extension that gets used, you can do so by setting the `relfilesuffix` attribute.

The following example demonstrates how to use the `relfilesuffix` attribute to control the file extension for inter-document cross references when you want to create a source-to-source reference. The assignment is hidden behind a check for `env-name`, where `env-name` is an attribute that is only set in an environment where you need to make this type of reference.

```
= Document Title
ifdef::env-name[:relfilesuffix: .adoc]

See the xref:README.adoc[README].

We could also write the link as link:README{relfilesuffix}[README].
```

The links in the generated document will now point to *README.adoc* instead of *README.html*.

This configuration is not actually necessary on GitHub, GitLab, or the browser preview extension since those environments automatically set the value of `relfilesuffix` to match the file extension of the source file. However, this setting may still be required for other environments, so it's worth knowing.

Mapping references to a different structure

While `relfilesuffix` 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: ../
:relfilesuffix: /
```

Now, the cross reference `<<filename.adoc,link text>>` 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).

Cross Reference Text and Styles

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 AsciiDoc processing, specifically during conversion.

Since this is a newer feature of the AsciiDoc language, it may not be supported by all converters. Where you can find support for it is in Asciidoctor's HTML, PDF, and EPUB 3 converters. It's not supported by the DocBook converter since it's a feature the DocBook toolchain already provides.

Default styling

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.

Cross reference styles

The generated text of a cross reference is controlled by the xrefstyle. It will also vary for different element types (section, figure, etc). Let's consider the following document to learn how the xrefstyle value affects the generated text of a cross reference.

```
== Installation

.Big Cats
image::big-cats.png[]
```

There are three built-in styles supported by the xrefstyle document attribute that you can choose from to customize the generated text of a cross reference.

:xrefstyle: full

Uses the signifier for the reference followed by the reference number and emphasized (chapter or appendix) or title enclosed in quotes (e.g., Section 2.3, "Installation") (e.g., Figure 1, "Big Cats").

:xrefstyle: short

Uses the signifier for the reference followed by the reference number (e.g., Section 2.3) (e.g., Figure 1).

:xrefstyle: basic

Uses the title only, only applying emphasis if the reference is a chapter or appendix (e.g., Installation) (e.g., Big Cats).

The `xrefstyle` attribute can also be specified directly on the xref macro to override the xrefstyle

value for a single reference (e.g., `xref:installation[xrefstyle=short]`). The element attribute supports the same three styles.

The xrefstyle 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 `<context>-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.

Reference signifiers

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.

Validate Cross References

An AsciiDoc processor is only required to provide 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.

When using Asciidoctor, you can enable validation of cross references in several ways:

- when using the CLI, passing the `-v` CLI option
- when using the API, setting the global variable `$VERBOSE` to the value `true`
- when using the API, setting the level on the global logger to INFO (i.e., `Asciidoctor::LoggerManager.logger.level = :info`)

All of these adjustments put the processor into pedantic mode. In this mode, the parser will immediately validate cross references, issuing a warning message if the reference is not valid. If you set the global variable `$VERBOSE` to `true`, it will also enable warnings in Ruby, which may not be what you want.

Consider the following example:

```
See <<foobar>>.

[#foobaz]
== Foobaz
```

If you run Asciidoctor in verbose/pedantic mode on this document (`-v`), it will send the following warning message to the logger.

```
asciidoctor: WARNING: invalid reference: foobar
```

An AsciiDoc processor is only required to validate references within the same document (after any includes are resolved).

Footnotes

AsciiDoc provides the `footnote` macro for adding footnotes to your document. A footnote is a reference to an item in a footnote list. The footnote is defined in AsciiDoc at the location of the reference, but the text is extracted to an item in the footnote list. You can refer to the same footnote in multiple locations by assigning an ID to the first occurrence and referencing that ID in subsequent occurrences.

All AsciiDoc processors, including Asciidoctor, currently implement footnotes as endnotes. The placement and numbering of footnotes can be customized using a custom converter.

Footnote macro syntax

You can insert footnotes into your document using the footnote macro. The text of the footnote is defined between the square brackets of the footnote macro (`footnote:[text]`). The footnote macro accepts an optional ID using the target of the macro (`footnote:id[text]`). Specifying an ID allows you to refer to that same footnote from multiple locations in the document. To make a reference to a previously defined footnote, you specify the ID in the target without specifying text (`footnote:id[]`).

Example 142. Footnote syntax

```
The hail-and-rainbow protocol can be initiated at five levels:

. doublefootnote:[The double hail-and-rainbow level makes my toes tingle.] ① ②
. tertiary
. supernumerary
. supermassive
. apocalyptic

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 specify the ID of the footnote in the target slot. The text between the square brackets should be empty. If both the ID and text are specified, and the ID has already been defined by an earlier footnote, the text is ignored.

If you find that having to put the footnote macro directly adjacent to a word makes it difficult to read, you can insert an attribute reference in between that resolves to an empty string (e.g., `word{empty}footnote:[text]`).

The footnotes are numbered consecutively throughout the article.

The results of Example 142 are displayed below.

The hail-and-rainbow protocol can be initiated at five levels

1. double[1]
2. tertiary
3. supernumerary
4. supermassive
5. apocalyptic

A bold statement![2]

Another outrageous statement.[2]

Just like normal paragraph text, you can use text formatting markup in the text of the footnote.

Externalizing a footnote

Since footnotes are defined using an inline macro, the footnote content must be inserted alongside the text it's annotating. This requirement can make the text harder to read. You can solve this problem by externalizing your footnotes using document attributes.

When defining a document attribute that holds a footnote, you can name the document attributes whatever you want. A common practice is to name the attribute using the `fn-` prefix. The name of the attribute can be as verbose (`fn-disclaimer`) or concise (`fn-1`) as you prefer.

Here's the previous example with the footnotes defined in document attributes and inserted using attribute references.

Example 143. 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{fn-hail-and-rainbow}
. tertiary
. supernumerary
. supermassive
. apocalyptic

A bold statement!{fn-disclaimer}

Another outrageous statement.{fn-disclaimer}
```

Notice you still get the benefit of seeing where the footnote is placed without all the noise. And since the footnotes are now defined in the document header, they could be further externalized to an include file.

This approach works since attribute references are expanded before footnotes are parsed. However, this technique does not work if you have text formatting markup in the text of the footnote (e.g., `*bold*`). That markup will not be interpreted. That's because the attributes substitution (which replaces attribute references) is applied *after* the quotes substitution (which interprets text formatting markup). In order to use text formatting markup in the text of the footnote, you need to configure the substitutions on the value of the attribute entry using the `pass:[]` macro.

The following example demonstrates how to configure the substitutions applied to the text of an externalized footnote so that text formatting markup is honored.

Example 144. Externalized footnote with text formatting

```
:fn-disclaimer: pass:c,q[footnote:disclaimer[Opinions are _mine_, and mine *alone*.]]

A bold statement!{fn-disclaimer}

Another outrageous statement.{fn-disclaimer}
```

The `c,q` target on the pass macro instructs the processor to apply the special characters substitution followed by the quotes substitution. That means the text formatting in the footnote text will already be applied when the footnote is inserted using an attribute reference.

Footnotes in headings

Footnotes are **not officially supported in headings** (section titles and discrete headings) in prespec AsciiDoc. While the footnote gets parsed, there's no guarantee that it will work properly and may require workarounds. This limitation may be lifted once the AsciiDoc Language is defined by the specification.

If you use a footnote in a heading, you'll likely find that the footnote index is wrong (either not incremented or out of order). That's because headings (section titles and discrete headings) get converted out of document order for the purpose of generating IDs, populating up cross references, and eagerly resolving attribute references.

The only way to workaround this limitation is by assigning an explicit ID **and** reftext to any heading that contains a footnote. For example:

```
See <<heading>>.

[[heading,Heading]]
== Headingfootnote:[This is a heading with a footnote]
```

Assigning an explicit ID and reftext to a heading will prevent the heading from being converted eagerly (thus deferring the footnote substitution) until the heading is rendered. As a result, the foot-

note macro in the heading will be processed in document order.

This workaround will also prevent the footnote number from reappearing in the text of an xref.

Even with this workaround, you still have to avoid using attribute references in the heading as those also causes the heading to be converted eagerly (which forces substitutions to be applied). If you use an attribute reference in the heading, the footnotes will be processed out of document order.

[1] The double hail-and-rainbow level makes my toes tingle.

[2] Opinions are my own.

Images

There are two AsciiDoc image macro types, block and inline. As with all macros, the block and inline forms differ by the number of colons that follow the macro name. The block form uses two colons (`::`), whereas the inline form only uses one (`:`).

Block image macro

A **block image** is displayed as a discrete element, i.e., on its own line, in a document. A block image is designated by `image` macro name and followed by two colons (`::`) It's preceded by an empty line, entered on a line by itself, and then followed by an empty line.

Example 145. Block image macro

```
Content in document.

image::sunset.jpg[] ① ②

Content in document
```

① To insert a block image, type the `image` macro name directly followed by two colons (`::`).

② After the colons, enter the image file target. Type a pair of square brackets (`[]`) directly after the target to complete the macro.

The result of Example 145 is displayed below.

The target is required. The target may be a relative path or a URL. How the target is interpreted depends on the processor settings and/or output format. If the converter generates output that references the image, the path must be relative to the published document. If the converter embeds the image in the output document (e.g., inline SVG, `data-uri` is set, converting to PDF directly), the target must be resolvable at convert time. In the latter case, a URL will only be resolved if the secu-

rity settings on the processor allows it (e.g., `allow-uri-read`).

The target may contain space characters. In the HTML output, these spaces will be URL encoded (i.e., `%20`).

You can specify a comma-separated list of optional attributes inside the square brackets or leave them empty. If you want to specify alt text, enter it inside the square brackets.

Example 146. Block image macro with alt text

```
image::sunset.jpg[Sunset]
```

If the alt text contains a comma or starts with a valid attribute name followed by an equals sign, you must enclose the alt text in double quotes. The double quote enclosure effectively escapes the comma from being interpreted as an attribute separator. See Attribute list parsing to learn how the attribute list in a macro is parsed.

Example 147. Block image macro with alt text that contains a comma

```
image::sunset.jpg["Mesa Verde Sunset, by JAVH"]
```

Although you could enclose the alt text in single quotes to escape the comma, doing so implicitly enables substitutions. Unless you need substitutions to be applied to the alt text, prefer using double quotes as the enclosure.

You can also give the image an ID, title, set its dimensions and make it a link.

Example 148. Block image macro with attribute list

```
.A mountain sunset ①
[#img-sunset,link=https://www.flickr.com/photos/javh/5448336655] ②
image::sunset.jpg[Sunset,200,100] ③ ④
```

① Defines the title of the block image, which gets displayed underneath the image when rendered.

② Assigns an ID to the block and makes the image a link. The `link` attribute can also be defined inside the attribute list of the block macro.

③ The first positional attribute, *Sunset*, is the image's alt text.

④ The second and third positional attributes define the width and height, respectively.

The result of Example 148 is displayed below.

Figure 1. A mountain sunset

Figure caption label

When a title is defined on a block image, the image title will be prefixed by a caption label (Figure) and numbered automatically. To turn off figure caption labels and numbers, unset the `figure-caption` attribute in the document header.

```
= Document Title
:figure-caption!:
```

Inline image macro

An **inline image** is displayed in the flow of another element, such as a paragraph or sidebar block. The inline image macro is almost identical to the block image macro, except its macro name is followed by a single colon (`:`).

Example 149. Inline image macro

```
Click image:play.png[] to get the party started. ①

Click image:pause.png[title=Pause] when you need a break. ②
```

① In the flow of an element, enter the macro name and a single colon (`image:`), followed by the image target. Complete the macro with a pair of square brackets (`[]`).

② You can specify a comma-separated list of attributes inside the square brackets or leave them empty.

The result of Example 149 is displayed below.

Click  to get the party started.

Click  when you need a break.

The target is required. The target may be a relative path or a URL. How the target is interpreted depends on the processor settings and/or output format. If the converter generates output that references the image, the path must be relative to the published document. If the converter embeds the image in the output document (e.g., inline SVG, `data-uri` is set, converting to PDF directly), the target must be resolvable at convert time. In the latter case, a URL will only be resolved if the security settings on the processor allows it (e.g., `allow-uri-read`).

The target may contain space characters. In the HTML output, these spaces will be URL encoded (i.e., `%20`).

The alt text for an inline image has the same requirements as for a block image, with the added restriction that a closing square bracket must be escaped.

For inline images, the optional title is displayed as a tooltip.

Set the Images Directory

The path to the location of the image catalog is controlled by the `imagesdir` attribute.

imagesdir attribute syntax

`imagesdir` is a document attribute. Its value is automatically added to the beginning of every image macro target. The resolved location of a image is: `<value-of-imagesdir> + <image-macro-target>`. Therefore, you never need to reference this attribute in an image macro. You only need to set it in your document header.

Example 150. Incorrect

```
image::{imagesdir}/name-of-image.png[]
```

Example 151. Correct

```
image::name-of-image.png[]
```

The value of `imagesdir` can be an absolute path, relative path or URL. By default, the `imagesdir` value is empty. That means the images are resolved relative to the document. If an image macro's target is an absolute path or URL, the value of `imagesdir` is not added to the target path.

The benefit of the processor adding the value of `imagesdir` 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 stored in different locations, this attribute makes it possible to accommodate many different scenarios.

We recommend relying on `imagesdir` 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.

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.

Insert Images from a URL

You can reference images served from any URL (e.g., your blog, an image hosting service, your

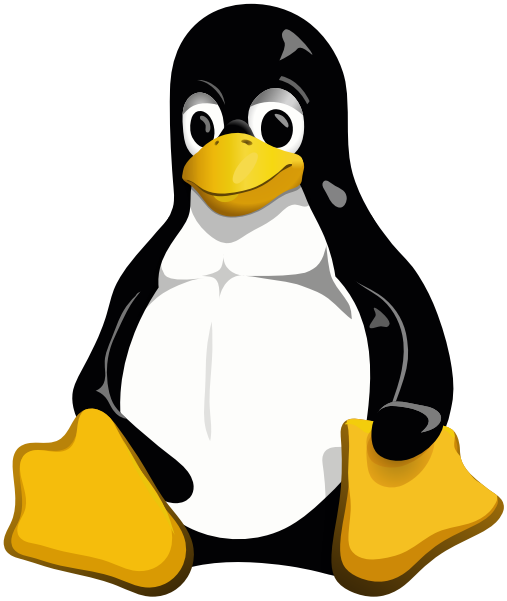
server, etc.) and never have to worry about downloading the images and putting them somewhere locally.

Image URL targets

Here are a few examples of images that have a URL target:

Example 152. Block image with a URL target

```
image::https://upload.wikimedia.org/wikipedia/commons/3/35/Tux.svg[Tux,250,350]
```



Example 153. 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  everywhere these days.

The value of `imagesdir` is ignored when the image target is a URL.

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:

Example 154. 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, `imagesdir` is used since the image target is not a URL (the value of `imagesdir` just happens to be one).

Position and Frame Images

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

Positioning attributes

AsciiDoc 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.

Example 155. 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.

Example 156. 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.

Positioning roles

Here are the examples from above, now configured to use roles that map to CSS classes in the default Asciidoctor stylesheet:

Example 157. Block image macro using positioning roles

```
[.right.text-center]
image::tiger.png[Tiger,200,200]
```

Example 158. 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

	Float		Align		
Role	left	right	text-left	text-right	text-center
Block Image	Yes	Yes	Yes	Yes	Yes
Inline Image	Yes	Yes	No	No	No

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.

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.

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 in 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.

Add Link to Image

You can turn an image into a link by using the `link` attribute.

link attribute

The link attribute on a block or image macro acts as though the image is wrapped in a link macro. While it's possible to wrap an inline image macro in a link macro, that combination is not well supported and may introduce subtle parsing problems. Therefore, you should use the `link` attribute on the image macro instead.

The value of the `link` attribute is akin to the target of the link macro. It can point to any URL or relative path.

For a block image macro, the `link` attribute can be added to the block attribute line above the macro or inside the contents of the macro.

```
[link=https://example.org]
image::logo.png[Logo]
```

or

```
image::logo.png[Logo,link=https://example.org]
```

For an inline macro, the `link` attribute must be added inside the contents of the macro.

```
image:apply.jpg[Apply,link=https://apply.example.org] today!
```

Link controls

When using the `link` attribute, you can also use the same controls supported by the link macro to control how the link is constructed. Those controls are as follows:

- `window` attribute - instructs the browser to open the link in the specified named window
- `nofollow` option - instructs search engines to not follow the link
- `noopener` option - instructs the browser to navigate to the target without granting the new browsing context access to the original document

When the value of `window` attribute is **_blank**, the `noopener` option is automatically enabled.

Here's an example that shows how to use these controls.

```
image::logo.png[Logo,link=https://example.org,window=_blank,opts=nofollow]
```

Refer to the Target a separate window section in the link macro documentation for more information about how these link controls work.

Adjust Image Sizes

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).

width and height attributes

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're mapped as the second and third (unnamed) positional attributes on both 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]
```

The value of the width and height attributes should be an integer without a unit. The px unit is implied. Although the processor may allow it, you should never rely on a % value. While the % unit was supported in older versions of HTML, it was removed starting in HTML 5. If you need to specify a % value for PDF or DocBook output, use `pdfwidth` or `scaledwidth`, respectively. To scale the image relative to the content area in HTML output, use a role.

While the values of `width` and `height` can be used to scale the image, these attributes are primarily intended to specify the intrinsic size of the image in CSS pixels. 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. The `height` attribute should only be specified if the `width` attribute is also specified, and it should respect the aspect ratio of the image.

Automatic image scaling

The default Asciidoctor stylesheet implements responsive images (using width-wise scaling). If the width of the content area is smaller than the width of the image, the image will be scaled down to fit. To support this feature, the intrinsic aspect ratio of the image is preserved at all sizes.

Thus, when specifying the image's dimensions, you should choose values that honor the intrinsic aspect ratio of the image. If the values don't respect the aspect ratio, the height is ignored by the browser.

pdfwidth attribute

AsciiDoc 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)
iw	Percentage of the intrinsic width of the image

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.

scaledwidth attribute

AsciiDoc recognizes the following attributes to size images when converting to DocBook or when converting to PDF using Asciidoctor PDF. The `scaledwidth` attribute is ignored by other converters.

- `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 no units)	Percentage of the content width (area between margins)

The `scaledwidth` attribute in AsciiDoc is mapped to the `width` attribute on the `imagedata` tag in DocBook, whereas the `width` attribute in AsciiDoc is mapped to the `contentwidth` attribute on the `imagedata` tag in DocBook. If both the `width` and `scaledwidth` attributes are specified in AsciiDoc, the

`scaledwidth` tags precedence, so the DocBook output will only have the `width` attribute.

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	role=half-width	role=half-view-width
pdf	pdfwidth=100mm (or cm, in, pc, pt, px)	Not possible (support for the scale attribute is pending)	pdfwidth=80%	pdfwidth=50vw
docbook, pdf	scaled-width=100mm (or cm, em, in, pc, pt, px)	scale=75	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%]
```

If the cascading behavior of the sizing attributes does not work for your use case, you might consider a document attribute to set the attribute that is suitable for the backend you are using. Consider the following example:

```
ifdef::backend-html5[]
:twoinches: width=144
// using a role requires adding a corresponding rule to the CSS
:full-width: role=full-width
:half-width: role=half-width
:half-size: role=half-size
:thumbnail: width=60
endif::[]
ifdef::backend-pdf[]
:twoinches: pdfwidth=2in
// NOTE use pdfwidth=100vw to make the image stretch edge to edge
:full-width: pdfwidth=100%
:half-width: pdfwidth=50%
// NOTE scale is not yet supported by the PDF converter
:half-size: pdfwidth=50%
:thumbnail: pdfwidth=20mm
endif::[]
ifdef::backend-docbook5[]
```

```
:twoinches: scaledwidth=2in
:full-width: scaledwidth=100%
:half-width: scaledwidth=50%
:half-size: scale=50
:thumbnail: scaledwidth=20mm
endif::[]
```

Then you can specify the image to be half the width of the content area using the following syntax:

```
image::image.jpg[{half-width}]
```

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.

Specify Image Format

Although to a reader, an image is just an image, different image formats must be processed differently by the converter in order to be referenced by or embedded in the output format. Some image formats, such as SVG, even activate additional behavior. On this page, you learn how an AsciiDoc processor determines the format of an image and how it can be specified explicitly using the `format` attribute if the format cannot be determined.

Automatic image format

Most of the time, the image format can be determined from the file extension (e.g., `.svg`) of the target. Here's an example that shows an image target that has a file extension:

```
image::avatar.svg[]
```

In this case, the converter can determine that this is an SVG image based on the file extension `.svg`. So the author does not need to specify the image format. The author only needs to specify the format when the image format cannot be determined.

format attribute

When the file extension is not present, or—in the case of a URL—not in its usual place, the author must specify the image format explicitly. The `format` attribute on a block or inline image macro can be used to specify the format of an image.

```
image::https://example.org/avatar[format=svg]
```

In this case, the converter would not be able to determine that this is an SVG image because the target has no file extension. But since the author has specified `format=svg`, the converter can recognize this as an SVG image.

Here are few cases when the image format cannot be determined automatically:

- the target does not have a file extension
- the target is using an unrecognized file extension
- the target is a URL that contains a query string (thus hiding the file extension)

The value of the format attribute is the sub-MIME type (the part after the forward slash) (e.g., `png` for a PNG image). One exception to this rule is the format for SVG, which is specified as `svg` instead of `svg+xml`. You can find a list of common image formats along with their MIME type values on the image file type and format guide page.

When is the format used?

There are several scenarios when a converter almost always needs to know the image format. One is to distinguish an SVG target. Another is to convert the image to a data URI. Yet another is when a converter must reencode the image.

AsciiDoc recognizes extra settings for SVG images, such as how to include it in the HTML output and whether to provide a fallback image. If the converter needs to read and process the file, it often must use a different library to process an SVG since the contents of an SVG is an XML document. So being able to recognize an SVG target is essential for almost any converter.

If the `data-uri` attribute is set on the document, an AsciiDoc processor must convert the image to a data URI. To start, the data URI for an SVG differs from a data URI for any other format. But even when creating a data URI for a non-SVG, the MIME type of that image must be included in the data URI, in which case the image format must be known.

Some converters, such as a PDF converter, have to reencode the image data. This means the converter must parse the image data, which may require loading an additional library. Doing that processing almost certainly requires knowing the format of the image. It also provides an opportunity for the converter to inform the author whether the image uses a format that cannot be processed.

As you can see, while the image format not always needed, it's needed often enough that you should specify the format if it's not apparent from the file extension of the target.

SVG Images

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 affect how the SVG behaves (or misbehaves). Therefore, these macros provide additional options to control how the SVG is included (i.e., referenced).

SVG dimensions

The viewBox attribute on the root `<svg>` element is required. The viewBox establishes the coordinate space on to which x and y values inside the SVG data are placed. Without this information, converters cannot properly interpret the SVG data and translate it to the canvas.

We strongly recommend not using the width and height attributes on the root `<svg>` element. You will find that SVGs that only specify a viewBox work best in a document. That's because the SVG data itself is infinitely scalable. By assigning an explicit width and height, you may end up limiting how the SVG can be sized or positioned in the document. It's better to specify the width (or similar, such as pdfwidth) on the image macro instead, or using CSS.

You particularly want to avoid using a percentage width, such as `width="100%"`. According to the SVG spec, this means using all available space, but without altering the aspect ratio. As a result, you can get a large gap above and below the SVG in page-oriented media, such as PDF. If you do specify a width and height, at least make sure the values are fixed and that they respect the aspect ratio of the data.

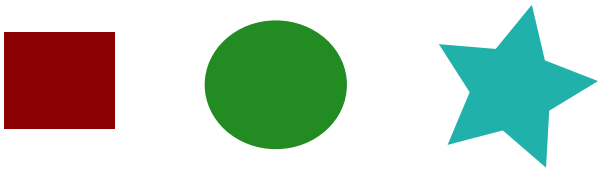
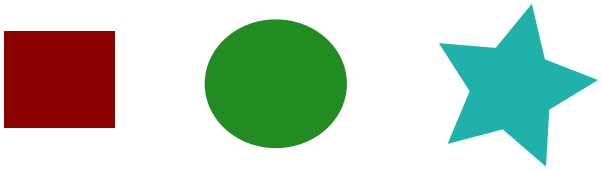
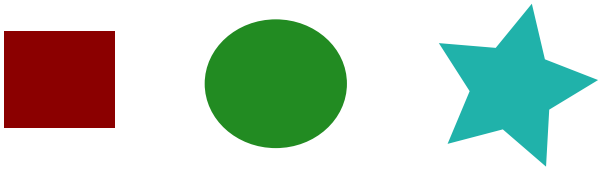
Options for SVG images

When the image target is an SVG, the `options` attribute (often abbreviated as `opts`) on the macro accepts one of the following values to control how the SVG is referenced:

- *none* (default)
- `interactive`
- `inline`

The following table demonstrates the impact these options have.

Demonstration of option values for SVG images

`image::sample.svg[Static,300]`	
	Observe that the SVG does not respond to the hover event.
`image::sample.svg[Interactive,300,opts=interactive]`	
	Observe that the color changes when hovering over the SVG.
`image::sample.svg[Embedded,300,opts=inline]`	
	Observe that the color changes when hovering over the SVG. The SVG also inherits CSS from the document stylesheets.

How these options values work and when each should be used is described below:

Option values that control how an SVG image is referenced in the HTML output

Option	HTML Element Used	Effect	When To Use
none (default)	`<img>`	Image is rasterized	Static image, no interactivity, no custom fonts
`interactive`	`<object>`	Image embedded as a live, interactive object	For using CSS animations, scripting, webfonts, or when you want to specify a fallback image
`inline`	`<svg>`	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 `interactive` option, you can specify a fallback image using the `fallback` attribute. The fallback image is used if the browser does not support the `<object>` tag. If the value of the fallback attribute is a relative path, it will be prefixed with the value of the `imagesdir` document attribute.

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.

Images Reference

Document attributes and values

Attribute	Value(s)	Example Syntax	Comments
`imagesdir`	empty, filesystem path, or base URL	`:imagesdir: images`	Added in front of a relative image target, joined using a file separator if needed. Not used if the image target is an absolute URL or path. Default value is empty.

Block and inline image attributes and values

Attribute	Value(s)	Example Syntax	Comments
`id`	User defined text	`id=sunset-img` (or `[[macros:image-ref:::sunset-img]]` or `[#sunset-img]` above block macro)	
`alt`	User defined text in first position of attribute list or named attribute	`image::sunset.jpg[Brilliant sunset]` (or `alt=Sunset`)	
`fallback`	Image path relative to `imagesdir` or an absolute path or URL	`image::tiger.svg[fallback=tiger.png]`	Only applicable if target is an SVG and opts=interactive
`title`	User defined text	in attrlist: `title="A mountain sunset"` (enclosing quotes only required if value contains a comma) above block macro: `.A mountain sunset`	Blocks: title displayed below image Inline: title displayed as tooltip
`format`	The format of the image, specified as a sub-MIME type (except in the case of an SVG, which is specified as `svg`).	`format=svg`	Only necessary when the converter needs to know the format of the image and the target does not end in a file extension (or otherwise cannot be detected).
`caption`	User defined text	`caption="Figure 8: "`	Only applies to block images.
`width`	User defined size in pixels	`image::sunset.jpg[Sunset,300]` (or `width=300`)	
`height`	User defined size in pixels	`image::sunset.jpg[Sunset,300,200]` (or `height=200`)	The height should only be set if the width attribute is set and must respect the aspect ratio of the image.

Attribute	Value(s)	Example Syntax	Comments
`link`	User defined location of external URI	`link=https://www.flickr.com/photos/javh/5448336655`	
`window`	User defined window target for the `link` attribute	`window=_blank`	
`scale`	A scaling factor to apply to the intrinsic image dimensions	`scale=80`	DocBook only
`scaledwidth`	User defined width for block images	`scaledwidth=25%`	DocBook and Asciidoctor PDF only
`pdfwidth`	User defined width for images in a PDF	`pdfwidth=80vw`	Asciidoctor PDF only
`align`	`left`, `center`, `right`	`align=left`	Block images only. `align` and `float` attributes are mutually exclusive.
`float`	`left`, `right`	`float=right`	Block images only. `float` and `align` attributes are mutually exclusive. To scope the float, use a float group.
`role`	user-defined, `left`, `right`, `th`, `thumb`, `related`, `rel`	`role="thumb right"` (or `[.thumb.right]` above block macro)	The role is preferred to specify the float position for an image. Role shorthand (`.`) can only be used in block attribute list above a block image.
`imagesdir`	empty, filesystem path, or base URL	`imagesdir=ch1/images`	Overrides the `imagesdir` set on the document. If not specified, the `imagesdir` from the document is used. (Not supported until Asciidoctor 2.1).
`opts`	Additional options for link creation and SVG targets.	`image::sunset.jpg[Sunset, link=https://example.org, opts=nofollow]` `image::chart.svg[opts=inline]`	Option names include: `nofollow`, `noopener`, `inline` (SVG only), `interactive` (SVG only)

Audio and Video

Audio macro syntax

The block audio macro enables you to embed audio streams into your documentation. You can embed self-hosted audio files that are supported by the browser.

The audio formats AsciiDoc supports is dictated by the output format, such as the formats supported by the browser when generating HTML. While this was once a precarious ordeal, HTML 5 has brought sanity to audio support in the browser by adding a dedicated `<audio>` element and by introducing several standard audio formats. Those formats are now widely supported across browsers and systems.

For a canonical list of supported web audio formats and their interaction with modern browsers, see the Mozilla Developer Supported Media Formats documentation.

Example 159. Basic audio file include

```
audio::ocean-waves.wav[]
```

You can control the audio 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.

Example 160. Set attributes for local audio playback

```
audio::ocean-waves.wav[start=60,opts=autoplay]
```

You can include a caption above the audio using the title attribute.

Example 161. Add a caption to the audio

```
.Take a zen moment
audio::ocean-waves.wav[]
```

Video macro syntax

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 Vimeo and YouTube.

The video formats AsciiDoc supports is dictated by the output format, such as the formats supported by the browser when generating HTML. While this was once a precarious ordeal, HTML 5 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 Vimeo or YouTube 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 Vimeo and YouTube videos for details about how to serve videos from these services.

Example 162. 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.

Example 163. Set attributes for local video playback

```
video::video-file.mp4[width=640,start=60,opts=autoplay]
```

You can include a caption on the video using the title attribute.

Example 164. Add a caption to a video

```
.A walkthrough of the product
video::video-file.mp4[]
```

Vimeo and YouTube videos

The video macro supports embedding videos from external video hosting services like Vimeo and YouTube. The AsciiDoc processor, specifically the converter, automatically generates the correct code to embed the video in the HTML output.

In order for an embedded YouTube video to work in Firefox when viewing the generated HTML document through the file: protocol, you must set `security.fileuri.strict_origin_policy` on the about:config settings page to `false`.

To use this feature, put the video ID in the macro target and the name of the hosting service in the first positional attribute.

Example 165. Embed a Vimeo video

```
video::67480300[vimeo]
```

Example 166. Embed a YouTube video

```
video::RvRhUHTV_8k[youtube]
```

When embedding a YouTube video, you can specify a playlist to associate with the video using the `list` attribute. The playlist must be specified by its ID.

Example 167. Embed a YouTube video with a playlist

```
video::RvRhUHTV_8k[youtube,list=PLDitloyBcHOm49bxNhvGgg0f9NRZ5lSaP]
```

Instead of using the `list` attribute, you can specify the ID of the playlist after the video ID in the target, separated by a slash.

Example 168. Embed a YouTube video with a playlist in the target

```
video::RvRhUHTV_8k/PLDitloyBcHOm49bxNhvGgg0f9NRZ5lSaP[youtube]
```

Alternatively, you can create a dynamic, unnamed playlist by listing several additional video IDs in the `playlist` attribute.

Example 169. Embed a YouTube video with a dynamic playlist

```
video::RvRhUHTV_8k[youtube,playlist="_SvwdK_HibQ,SGqg_ZzThDU"]
```

Instead of using the `playlist` attribute, you can create a dynamic, unnamed playlist by listing several video IDs in the target separated by a comma.

Example 170. Embed a YouTube video with a dynamic playlist in the target

```
video::RvRhUHTV_8k,_SvwdK_HibQ,SGqg_ZzThDU[youtube]
```

Audio and video attributes and options

Audio attributes and values

Attribute	Value(s)	Example Syntax	Notes
`title`	User defined text	`.Ocean waves`	
`start`	User-defined playback start time in seconds.	`start=30`	

Attribute	Value(s)	Example Syntax	Notes
`end`	User-defined playback end time in seconds.	`end=90`	
`options` (`opts`)	`autoplay`, `loop`, `controls`, `nocontrols`	`opts="autoplay,loop"`	The controls value is enabled by default

Video attributes and values

Attribute	Value(s)	Example Syntax	Notes
`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 Vimeo (`vimeo`) or YouTube (`youtube`).
`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.
`start`	User-defined playback start time in seconds.	`start=30`	
`end`	User-defined playback end time in seconds.	`end=90`	
`theme`	The YouTube theme to use for the frame.	`theme=light`	Valid values are `dark` (the default) and `light`.
`lang`	The language used in the YouTube frame.	`lang=fr`	A BCP 47 language tag (typically a two-letter language code, like `en`).
`list`	The ID of a playlist to associate with a YouTube video.	`list=PLabc123`	Only applies to YouTube videos.
`playlist`	Additional video IDs to create a dynamic YouTube playlist.	`playlist="video-abc,video-xyz"`	IDs must be separated by commas. Therefore, the value must be enclosed in double quotes. Only applies to YouTube videos.
`align`	`left`, `center`, `right`	`align=center`	Follows the same alignment rules as a block image.
`options` (`opts`)	`autoplay`, `loop`, `modest`, `nocontrols`, `nofullscreen`, `muted`	`opts="autoplay,loop"`	The controls are enabled by default. The `modest` option enables modest branding for a YouTube video.

Icons

Icons are a useful way to communicate information visually while at the same time eliminating text that can distract from the primary text. Icons also have the benefit of adding some flair to your document. In AsciiDoc, there are numerous ways to embellish the output of your document with icons (for backends that support this feature).

For some elements, icons are added automatically by the processor when enabled, such as the admonition icons. You can also add icons directly to the content using special markup.

This section shows you how to enable icons, covers the various icon modes, and introduces you to the icon macro for adding custom icons to your content.

Enable icons

There are three icon modes: text, image, and font.

The inclusion of icons in the output is controlled using the `icons` document attribute. By default, this attribute is not set. As a result, all icons are displayed as text. In the text icon mode, icons are effectively disabled.

To enable icons, set the `icons` attribute in the document header.

```
= Document Title
:icons:
```

Valid values for the `icons` attribute are as follows:

image (or empty)
: Icons resolve to image files in the directory specified by the `iconsdir` attribute.

font
: Icons are loaded from an icon font (like Font Awesome). Not all backends support this mode, such as DocBook.

Where icons are used

Setting the `icon` attribute turns on icons in the following locations:

- The admonition label is replaced with an icon (i.e., admonition icons)
- The icon macro
- Callout numbers

The following pages will cover these icon modes in more depth.

Image Icons Mode

Setting the `icons` attribute to `image` (or leaving it empty) instructs the AsciiDoc processor to use images for icons. This page defines where the processor looks for these image files by default, which image file extension it prepends, and how to configure both.

Enable image-based icons

To enable image-based icons, you set the `icons` attribute in the document header to the value `image`.

```
= Document Title
:icons: image
```

This setting has no affect on the parsing of the AsciiDoc document. It only influences the output generated by the converters.

Default icons directory and type

By default, the AsciiDoc processor will look for icons in the *icons* directory relative to the value of the `imagesdir` attribute. If you have not configured either attribute, that path resolves to *./images/icons*.

The processor won't look for icons of any type (i.e., format). Instead, it will look for icons that have the *.png* file extension.

Let's assume you have the following NOTE admonition block in your document:

```
NOTE: Remember the milk!
```

The AsciiDoc processor will resolve the admonition icon to *./images/icons/note.png*.

Configure the icons directory using iconsdir

To change where the AsciiDoc processor looks for icons, you can specify a different location using the `iconsdir` attribute.

For example:

```
= Document Title
:icons: image
:iconsdir: icons
```

When converting this document, the AsciiDoc processor will look for images in the *icons* directory instead of the default *./images/icons*.

Configure the icon type using icontype

If the icon path is derived, such as for an admonition icon or if the target of the icon macro does not have a file extension, the AsciiDoc processor will use the `icontype` attribute to determine which image type (i.e., format) to look for. By default, the value of this attribute is `png`, so the processor will look for an image with the file extension `.png`.

You can use the `icontype` document attribute to configure the default icon type.

For example:

```
= Document Title
:icons: image
:icontype: svg
```

For NOTE admonitions, the AsciiDoc processor will now look for the image *note.svg* in the `iconsdir` instead of *note.png*.

The value of the `icontype` attribute is ignored for the icon macro if the target has a file extension. It's only used when the icon type must be inferred.

Font Icons Mode

Setting the `icons` attribute to `font` instructs the AsciiDoc processor to select icons from an icon font.

Not all converters support this mode. If a converter does not support this mode, it will fall back to the image mode.

Enable font-based icons

To enable image-based icons, you set the `icons` attribute in the document header to the value `font`.

```
= Document Title
:icons: font
```

This setting has no affect on the parsing of the AsciiDoc document. It only influences the output generated by the converters.

When converting to HTML, the stylesheet is required in order for the font-based icons to work.

Default icon font

The icon font that is used by default is determined by the processor. Asciidoctor, for example, uses the Font Awesome icon font. You can see the available icons in Font Awesome on the Font Awesome icons page. Using the Font Awesome icons in Asciidoctor requires online access by default.

Default admonition icons

Since the names of admonitions doesn't necessarily match the names of icons in the icon font, the AsciiDoc processor must map admonition CSS classes to icon names.

When using Font Awesome as the icon set, the following mappings are recommended:

- note → info-circle
- tip → lightbulb-o
- warning → warning
- caution → fire
- important → exclamation-circle

Callout numbers and font icon mode

In the font icon mode, callout numbers are displayed as enclosed numbers. However, the icon font is not used to render these glyphs. Instead, they are styled this way using CSS. This is done to allow the range of callout numbers to be open-ended.

Icon Macro

In addition to built-in icons, you can add icons anywhere in your content where macros are substituted using the icon macro. This page covers the anatomy of the icon macro, how the target is resolved, and what features it support (subject to the icon mode).

Anatomy

The icon macro is an inline macro. Like other inline macros, its syntax follows the familiar pattern of the macro name and target separated by a colon followed by an attribute list enclosed in square brackets.

```
icon:<target>[<attrlist>]
```

The `<target>` is the icon name or path. The `<attrlist>` specifies various named attributes to configure how the icon is displayed.

For example:

```
icon:heart[2x,role=red]
```

Example

Here's an example that shows how to inserts an icon named *tags* in front of a list of tag names.

```
icon:tags[] ruby, asciidoctor
```

Here's how the HTML converter converts an icon macro when the `icons` attribute is not set or empty.

Example 171. Result: HTML output

```
<div class="paragraph">
  <p><span class="image"><img src="./images/icons/tags.png" alt="tags"></span> ruby,
asciidoctor</p>
</div>
```

Here's how the DocBook converter converts an icon macro.

```
<inlinemediaobject>
  <imageobject>
    <imagedata fileref="./images/icons/tags.png"/>
  </imageobject>
  <textobject><phrase>tags</phrase></textobject>
</inlinemediaobject> ruby, asciidoctor
```

When the image for an icon can't be located in the icons directory, the AsciiDoc processor displays the icon macro's alt (i.e. fallback) text.

How the icon is resolved

The target of the icon macro is an icon name (or path). How that target is resolved depends on the icon mode assigned to the icons attribute.

text

The icon name will be enclosed in square brackets (e.g., `[heart]`).

image

The icon name will be resolved to a file in the `iconsdir` with the file extension specified by `icontype` (e.g., *./images/icons/heart.png*).

font

The icon name will be resolved to a glyph in an icon font (as mapped by a CSS class) (e.g., `fa fa-heart`).

If you include a file extension in the image target, the icon macro will not work correctly when using the font icon mode (i.e., `icons=font`).

Icon macro attributes (shared)

The following attributes of the icon macro are shared for all icon modes.

`role`

The role applied to the element that surrounds the icon.

`title`

The title of the image displayed when the mouse hovers over it.

`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).

Role

Here's an example of an icon that uses a role to specify the color.

```
icon:tags[role=blue] ruby, asciidoctor
```

Link and window

Here's an example of an icon with a link that targets a separate window:

```
icon:download[link=https://rubygems.org/downloads/whizbang-1.0.0.gem, window=_blank]
```

Icon macro attributes (image mode only)

The attributes listed below only apply when using the image icon mode.

`alt`

The alternative text on the `<img>` element (HTML output) or text of `<inlinemediaobject>` element (DocBook output)

`width`

The width applied to the image.

For example, here's how to control the icon alt text and width when using the image icon mode:

```
icon:tags[Tags,width=16] ruby, asciidoctor
```

The icon macro doesn't support any options to change its physical position (such as alignment).

Icon macro attributes (font mode only)

The icon macro has a few attributes that modify the size and orientation of a font-based icon. These attributes are only recognized in the font icon mode.

`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`

Size

To make the icon twice the size as the default, enter `2x` inside the square brackets.

```
icon:heart[2x]
```

or

```
icon:heart[size=2x]
```

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
```

Rotate and flip

To rotate and flip an icon, specify these options using named attributes:

```
icon:shield[rotate=90, flip=vertical]
```

Keyboard Macro

The keyboard macro allows to create a reference to a key or key sequence on a keyboard. You can use this macro when you need to communicate to a reader what key or key sequence to press to perform a function.

In order to use the UI macros, you must set the `experimental` document attribute. Although this attribute is named `experimental`, the UI macros are considered a stable feature of the AsciiDoc language. The requirement to specify the attribute is merely an optimization for the processor. If the specification committee determines that an attribute is still necessary, the name of the attribute will likely change to better reflect that the macros are integral to the language.

Keyboard macro syntax

The keyboard macro uses the short (no target) macro syntax `kbd:[key(+key)*]`. Each key is displayed as entered in the document. Multiple keys are separated by a plus (e.g., `Ctrl+T`) or a comma (e.g., `Ctrl,T`). The plus is preferred.

It's customary to represent alpha keys in uppercase, though this is not enforced.

If the last key is a backslash (`\`), it must be followed by a space. Without this space, the processor will not recognize the macro. If one of the keys is a closing square bracket (`]`), it must be preceded by a backslash. Without the backslash escape, the macro will end prematurely. You can find example of these cases in the example below.

Example 172. Using the 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
```

```
|===
```

The result of Example 172 is displayed below.

Shortcut	Purpose
F11	Toggle fullscreen
Ctrl + T	Open a new tab
Ctrl + Shift + N	New incognito window
\	Used to escape characters
Ctrl +]	Jump to keyword
Ctrl + +	Increase zoom

Button and Menu UI Macros

In order to use the UI macros, you must set the `experimental` document attribute. Although this attribute is named `experimental`, the UI macros are considered a stable feature of the AsciiDoc language. The requirement to specify the attribute is merely an optimization for the processor. If the specification committee determines that an attribute is still necessary, the name of the attribute will likely change to better reflect that the macros are integral to the language.

Button macro syntax

It can be 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!

Example 173. Using the button macro syntax

```
Press the btn:[OK] button when you are finished.

Select a file in the file navigator and click btn:[Open].
```

The result of Example 173 is displayed below.

Press the **[OK]** button when you are finished.

Select a file in the file navigator and click **[Open]**.

Menu macro syntax

Trying to explain how to select a menu item can be a pain. With the `menu` macro, the symbols do the work.

Example 174. Using the 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 Example 174 appear below.

To save the file, select **File › Save**.

Select **View › Zoom › Reset** to reset the zoom level to the default setting.

If the menu has more than one item, it can be expressed using a shorthand.

The shorthand syntax for menu is not on a standards track. You can use it for transient documents, but do not rely on it long term.

In the shorthand syntax:

- each item is separated by a greater than sign (>) with spaces on either side
- the whole expression must be enclosed in double quotes (")

The text of the item itself may contain spaces.

Example 175. Using the shorthand menu syntax

```
Select "Zoom > Reset" to reset the zoom level.
```

The shorthand syntax can be escaped by preceding the opening double quote with a backslash character.

Both the menu macro and menu shorthand require the first menu item start with a word character (alphanumeric character or underscore) or ampersand (to accommodate a character reference). If you need the first menu item to start with a non-word character, you will need to substitute it with the equivalent character reference. For example, to make a menu item that starts with vertical ellipsis, you must use `⋮`.

Example 176. Using a character reference at the start of the menu

```
Select "&#8942; > More Tools > Extensions" to find and enable extensions.
```

Subsequent menu items don't have this requirement and thus can start with any character.

Admonitions

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. This page introduces you to admonition types AsciiDoc provides, how to add admonitions to your document, and how to enhance them using icons or emoji.

The examples on this page (and in these docs) use a visual theme that differs from the style provided by AsciiDoc processors such as Asciidoctor. The AsciiDoc language does not require that the admonitions be rendered using a particular style. The only requirement is that they be offset from the main text and labeled appropriately according to their admonition type.

Admonition types

The rendered style of an admonition is determined by the assigned type (i.e., name). The AsciiDoc language provides five admonition types represented by the following labels:

- `NOTE`
- `TIP`
- `IMPORTANT`
- `CAUTION`
- `WARNING`

The label is specified either as the block style or as a special paragraph prefix. The label becomes visible to the reader unless icons are enabled, in which case the icon is shown in its place.

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.

The word caution in this context translates into attention in French, which is often a good reference for how it should be applied.

To find a deeper analysis, see www.differencebetween.com/difference-between-caution-and-vs-warning/.

Admonition syntax

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 (`:`).

Example 177. 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.

The result of Example 177 is displayed below.

Wolpertingers are known to nest in server racks. Enter at your own risk.

When you want to apply an admonition to compound 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.

Example 178. 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 (`====`).

The result of Example 178 is displayed below.

Feeding the Werewolves

While werewolves are hardy community members, keep in mind the following dietary concerns:

1. They are allergic to cinnamon.
2. More than two glasses of orange juice in 24 hours makes them howl in har-

mony with alarms and sirens.

3. Celery makes them sad.

Enable admonition icons

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 font icons by setting the `icons` attribute on the document and assigning it the value `font`.

Example 179. Admonition paragraph with icons set

```
= Document Title
:icons: font

WARNING: 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 Font Icons Mode.

Using emoji for admonition icons

If image-based or font-based icons are not available, you can leverage the admonition caption to display an emoji (or any symbol from Unicode) in the place of the admonition label, thus giving you an alternative way to make admonition icons.

If the `icons` attribute is not set on the document, the admonition label is shown as text (e.g., CAUTION). 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 result you get in the HTML:

```
<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 from the AsciiDoc processor is run through a postprocessing filter that substitutes emoji shortcodes with emoji symbols. That means you can use these shortcodes instead 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.

Sidebars

On this page, you'll learn:

- How to mark up a sidebar with AsciiDoc.

A sidebar can contain any type of content, such as quotes, equations, and images. Normal substitutions are applied to sidebar content.

Sidebar style syntax

If the sidebar content is contiguous, the block style `sidebar` can be placed directly on top of the text in an attribute list (`[]`).

Example 180. Assign sidebar block style to paragraph

```
[sidebar]
Sidebars are used to visually separate auxiliary bits of content
that supplement the main text.
```

The result of Example 180 is displayed below.

Sidebars are used to visually separate auxiliary bits of content that supplement the main text.

Delimited sidebar syntax

A delimited sidebar block is delimited by a pair of four consecutive asterisks (`****`). You don't need to set the style name when you use the sidebar's delimiters.

Example 181. Sidebar block delimiter syntax

```
.Optional Title
****
Sidebars are used to visually separate auxiliary bits of content
that supplement the main text.

TIP: They can contain any type of content.

.Source code block in a sidebar
[source,js]
----
const { expect, expectCalledWith, heredoc } = require('../test/test-utils')
----
****
```

The result of Example 181 is displayed below.

Optional Title

Sidebars are used to visually separate auxiliary bits of content that supplement the main text.

They can contain any type of content.

Example 182. Source code block in a sidebar

```
const { expect, expectCalledWith, heredoc } = require('../test/test-utils')
```

Example Blocks

On this page, you'll learn:

- [x] How to mark up an example block with AsciiDoc.

An example block is useful for visually delineating content that illustrates a concept or showing the result of an operation.

An example can contain any type of content and AsciiDoc syntax. Normal substitutions are applied to example content.

Example style syntax

If the example content is contiguous, i.e., not interrupted by empty lines, the block style name `example` can be placed directly on top of the text in an attribute list (`[]`).

Example 183. Assign example block style to paragraph

```
.Optional title
[example]
This is an example of an example block.
```

The result of Example 183 is displayed below.

Optional title

This is an example of an example block.

Delimited example syntax

If the example content includes multiple blocks or content separated by empty lines, place the content between delimiter lines consisting of four equals signs (`====`).

You don't need to set the block style name when you use the example delimiters.

Example 184. Example block delimiter syntax

```
.Onomatopoeia
====
The book hit the floor with a *thud*.

He could hear doves *cooing* in the pine trees`' branches.
====
```

The result of Example 184 is displayed below.

Onomatopoeia

The book hit the floor with a **thud**.

He could hear doves **cooing** in the pine trees' branches.

Complex admonitions use the delimited example syntax.

Blockquotes

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

Basic quote syntax

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. These positional attributes are all optional.

Example 185. Anatomy of a basic quote

```
[quote,attribution,citation title and information]
Quote or excerpt text
```

You can include an optional space after the comma that separates each positional attribute. If an attribute value includes a comma, enclose the value in double or single quotes.

If the quote is a single line or paragraph (i.e., a styled paragraph), you can place the attribute list directly on top of the text.

Example 186. 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.

The result of Example 186 is displayed below.

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

Quoted block

If the quote or excerpt is more than one paragraph, place the text between delimiter lines consisting of four underscores (____).

Example 187. 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?
____
```

The result of Example 187 is displayed below.

> 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

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

Example 188. 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
```

The result of Example 188 is displayed below.

> 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

Excerpt

The quote block can be designated as an excerpt by adding the `excerpt` role. The exceptation is that this role makes the quote block appear with the quote decoration.

```
[.excerpt]
____
This text is an excerpt from the referenced literature.
____
```

The impact of this role is strictly a presentation concern and is thus handled by the styling system, such as the stylesheet for HTML.

Markdown-style blockquotes

Asciidoctor supports Markdown-style blockquotes. This syntax was adopted both to ease the transition from Markdown and because it's the most common method of quoting in email messages.

Example 189. 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
```

The result of Example 189 is displayed below.

> 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 some block content inside the blockquote, including paragraphs, lists, and nested blockquotes.

Example 190. 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 the conversation from Example 190 is rendered.

> > 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.

Be aware that not all AsciiDoc block elements are supported inside a Markdown-style blockquote. In particular, a description list is not permitted. The parser looks for the Markdown-style blockquote only after looking for a description list, meaning the description list takes precedence. Since the quote marker is a valid prefix for a description list term, the Markdown-style blockquote is not recognized in this case. If you need to put a description list inside a blockquote, you should use the AsciiDoc syntax for a blockquote instead.

The Markdown-style blockquote should only be used in simple cases and when migrating from Markdown. The AsciiDoc syntax should always be preferred, if possible.

Verses

When you need to preserve indents and line breaks, use a `verse` block. Verses are defined by setting `verse` on a paragraph or an excerpt block delimited by four underscores (`____`).

verse style syntax

When verse content doesn't contain any empty lines, you can assign the `verse` style using the first position in an attribute list.

Example 191. Verse style syntax

```
[verse,Carl Sandburg, two lines from the poem Fog]
The fog comes
on little cat feet.
```

The result of Example 191 is displayed below.

> The fog comes
> on little cat feet.
>
> — Carl Sandburg, two lines from the poem Fog

Delimited verse block syntax

When the verse content includes empty lines, enclose it in a delimited excerpt block.

Example 192. 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.
____
```

The delimited verse block from Example 192 is rendered below.

> The fog comes
> on little cat feet.
>
> It sits looking

> over harbor and city
> on silent haunches
> and then moves on.
>
> — Carl Sandburg, Fog

Verbatim and Source Blocks

Source Code Blocks

A source block is a specialization of a listing block. 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, AsciiDoc processors will integrate at least one library to syntax highlight the source code blocks in your document. For example, Asciidoctor provides integration with Rouge, CodeRay, Pygments, and highlight.js, as well as an adapter API to add support for additional libraries.

Example 193 shows a listing block with the `source` style and language `ruby` applied to its content, hence a source block.

Example 193. Source block syntax

```
[source,ruby]
----
require 'sinatra'

get '/hi' do
  "Hello World!"
end
----
```

The result of Example 193 is rendered below.

```ruby
require 'sinatra'

get '/hi' do
  "Hello World!"
end
```

Since a `source` block is most often used to designate a block with source code of a particular language, the `source` style itself is optional. The mere presence of the language on a listing block automatically promotes it to a source block.

Example 194 shows a listing block implied to be a source block because a language is specified.

Example 194. Implied source block

```
[,ruby]
----
require 'sinatra'

get '/hi' do
  "Hello World!"
end
```

```
----
```

This shorthand also works if the `source-language` attribute is set on the document, which serves as the default language for source blocks. If the `source-language` attribute is set on the document and you want to make a regular listing block, add the `listing` style to the block.

Using include directives in source blocks

You can use an include directive to insert source code into an AsciiDoc document directly from a file.

Example 195. Code inserted from another file

```
[,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.

Source Highlighting

Source highlighting is applied to text that's assigned the `source` block style (either explicitly or implicitly) and a source language. The source language is defined either on the block or inherited from the `source-language` document attribute.

source-highlighter attribute

Source highlighting isn't enabled by default. To enable source highlighting, you must set the `source-highlighter` attribute in the document header using an attribute entry.

```
= Document Title
:source-highlighter: <value>
```

For example, here's how to enable syntax highlighting using Rouge:

```
= Document Title
:source-highlighter: rouge
```

You can also declare this attribute using the CLI or API.

Available source highlighters

Table 1 lists the recognized values for the `source-highlighter` attribute and the toolchains that support the usage of the syntax highlighting libraries.

Table 1. Built-in source-highlighter values and the supporting toolchains

Library	Value	Toolchain
CodeRay	`coderay`	Asciidoctor, AsciidoctorJ, Asciidoctor PDF
highlight.js	`highlight.js`	Asciidoctor, AsciidoctorJ, Asciidoctor.js
Pygments	`pygments`	Asciidoctor, Asciidoctor PDF
Rouge	`rouge`	Asciidoctor, AsciidoctorJ, Asciidoctor PDF

To use Rouge, CodeRay, or Pygments, you must have the appropriate library installed on your system. See Rouge, CodeRay, or Pygments for installation instructions.

If you're using the client-side library Highlight.js, there's no need to install additional libraries. The generated HTML will load the required source files from a CDN, custom URL, or file path.

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".

Apply source highlighting

To apply highlighting to a block of source code, you must specify a source language. If the block is a literal block or paragraph, you must also specify the `source` style.

The AsciiDoc language does not specify the list of valid source language values. Instead, the available source language values are defined by the syntax highlighter library.

You can find the list of available languages supported by Rouge in the Rouge documentation. You can print a list of available languages supported by Pygments by running `pygmentize -L formatters`. The available languages supported by highlight.js depends on which bundle of highlight.js you are using.

Typically, the source language value is the proper name of the language in lowercase (e.g., `ruby`, `java`). Most syntax highlighters also accept using the source file extension (e.g., `js`, `rb`), though it's

important to be consistent. If the syntax highlighter doesn't recognize or support the source language, the block will not be highlighted.

Example 196. Source block with ID and source highlighting

```
[#hello,ruby] ① ② ③
--④
require 'sinatra'

get '/hi' do
  "Hello World!"
end
----
```

① The block style `source` is implied since a source language is specified.

② An optional ID can be added to the block by appending it to style using the shorthand syntax (`#`) for `id`.

③ Assign a source language to the second position.

④ An implicit source block uses the listing structural container.

The result of Example 196 is displayed below.

```
require 'sinatra'

get '/hi' do
  "Hello World!"
end
```

Example 197. Source paragraph

```
[source,xml] ①
<meta name="viewport"
  content="width=device-width, initial-scale=1.0">
②
This is normal content.
```

① Place the attribute list directly above the paragraph. In this case, the `source` style is always required.

② Once an empty line is encountered the source block ends.

The result of Example 197 is displayed below.

```
<meta name="viewport"
  content="width=device-width, initial-scale=1.0">
```

This is normal content.

shell vs console

The source language for shell and console are often mixed up. The language `shell` is intended for the contents of a shell script, often indicated by a shebang for the generic shell. If the shell script is written for a particular shell, you might use that language instead (e.g., `bash` or `zsh`). The language `console` is intended to represent text that's typed into a console (i.e., a terminal application).

Here's an example of when you would use `shell`:

```
[,shell]
----
#!/bin/sh

fail () {
    echo
    echo "$*"
    echo
    exit 1
} >&2

JAVACMD=java
which java >/dev/null 2>&1 || fail "ERROR: no 'java' command could be found in your
PATH.

exec "$JAVACMD" "$@"
----
```

Here's an example of when you would use `console`:

```
[source,console]
$ asciidoctor -v
```

Typically, the syntax highlighter will parse the prompt (e.g., `$`) at the start of each line, then handle the remaining text using the shell language.

Often times, a basic console command is represented using a literal paragraph since there isn't much to be gained from syntax highlighting in this case.

Enable line numbering

Provided the feature is supported by the source highlighter, you can enable line numbering on a source block by setting the `linenums` option on the block.

Line numbering is added by the syntax highlighter, not the AsciiDoc converter. Therefore, to get line numbering on a source block, you must have the `source-highlighter` attribute set and the library to which it refers must support line numbering. When using Asciidoctor, the only syntax highlighter that does not support line numbering is highlight.js.

The `linenums` option can either be specified as a normal block option named `linenums`, or as the third positional attribute on the block. The value of the positional attribute doesn't matter, though it's customary to use `linenums`.

Example 198. Enable line numbering using the `linenums` option

```
[%linenums,ruby]
----
puts 1
puts 2
puts 3
----
```

Example 199. Enable line numbering using the third positional attribute

```
[,ruby,linenums]
----
puts 1
puts 2
puts 3
----
```

Disable source highlighting

To disable source highlighting for a given source block, specify the language as `text` or remove the `source` style.

source-language attribute

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. Setting the `source-language` document attribute implicitly promotes listing blocks to source blocks.

Example 200. Set source-language attribute

```
= Document Title
:source-highlighter: pygments
:source-language: java

----
public void setAttributes(Attributes attributes) {
    this.options.put(ATTRIBUTES, attributes.map());
}
----
```

Notice that it's not necessary to specify the `source` style or source language on the block. To make a listing block in this situation, you must set the `listing` style on the block.

You can override the global source language on an individual block by specifying a source language

directly on the block.

Example 201. Override source-language attribute

```
= Document Title
:source-highlighter: pygments
:source-language: java

[,ruby]
require 'sinatra'
```

Highlight Select Lines

Not to be confused with source highlighting, you can highlight (i.e., emphasize) specific lines in a source block in order to call attention to them.

Usage criteria

Line highlighting can be applied to a source block if certain criteria are met.

`source-highlighter`	**Criteria to use the `highlight` attribute on a source block**
`coderay`	• The `linenums` option is enabled on the block. • The `highlight` attribute is defined on the block. **Line highlighting will only emphasize the line number itself.**
`rouge`	• The `highlight` attribute is defined on the block. • The CSS to support line highlighting is supplied by docinfo. (Needed even if `rouge-css=style`).
`pygments`	• The `highlight` attribute is defined on the block.
`highlight.js`	Not applicable. **Line highlighting isn't available when using highlight.js.**

highlight attribute

Line highlighting is activated on a source block if the highlight attribute is defined and at least one of the line numbers falls in this range.

Keep in mind that some syntax highlighter libraries require additional options (e.g., CodeRay and Rouge), and some don't support line highlighting at all (e.g., highlight.js).

The `highlight` attribute accepts a comma or semicolon delimited list of line ranges. The numbers correspond to the line numbers of the source block. If the `start` attribute is not specified, line numbers of the source block start at 1.

Here are some examples:

- 1
- 2,4,6
- 3..5
- 2,7..9

A line range is represented by two numbers separated by a double period (e.g., `2..5`). The range is inclusive.

CodeRay

Example 202. Highlight select lines when source-highlighter=coderay

```
= Document Title
:source-highlighter: coderay

[%linenums,ruby,highlight=2..5]
----
ORDERED_LIST_KEYWORDS = {
  'loweralpha' => 'a',
  'lowerroman' => 'i',
  'upperalpha' => 'A',
  'upperroman' => 'I',
}
----
```

When using CodeRay as the source highlighter, the `linenums` option is required to use line highlighting. That's because line highlighting is only applied to the line number, which is emphasized using bold text. CodeRay does not shade the line of code itself.

Rouge

Example 203. Highlight select lines when source-highlighter=rouge

```
= Document Title
:source-highlighter: rouge
:docinfo: shared

[,ruby,highlight=2..5]
----
ORDERED_LIST_KEYWORDS = {
  'loweralpha' => 'a',
  'lowerroman' => 'i',
  'upperalpha' => 'A',
  'upperroman' => 'I',
}
----
```

When using Rouge as the source highlighter, you must supply CSS to support line highlighting. You can do so by storing the required line highlighting CSS in a docinfo file, then including it in the output document by setting the `docinfo` document attribute.

Example 204. Docinfo file (docinfo.html) to support line highlighting with Rouge

```
<style>
pre.rouge .hll {
  background-color: #ffc;
  display: block;
}
pre.rouge .hll * {
  background-color: initial;
}
</style>
```

Note that this supplemental CSS is needed even when `rouge-css=style`. The Rouge integration does not embed the CSS for highlighting lines into the style attribute of the tag for each line. Instead, it sets the `hll` class on the tag (e.g., `<span class="hll">`).

Pygments

Example 205. Highlight select lines when source-highlighter=pygments

```
= Document Title
:source-highlighter: pygments

[,ruby,highlight=2..5]
----
ORDERED_LIST_KEYWORDS = {
  'loweralpha' => 'a',
  'lowerroman' => 'i',
  'upperalpha' => 'A',
  'upperroman' => 'I',
}
----
```

Highlight 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 should 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.

Listing Blocks

Blocks and paragraphs assigned the `listing` style display their rendered content exactly as you see it in the source. Listing content is converted to preformatted text (i.e., `<pre>`). The content is presented in a fixed-width font and endlines are preserved. Only special characters and callouts are replaced when the document is converted.

The listing style can be applied to content using one of the following methods:

- setting the `listing` style on a block or paragraph using an attribute list, or
- enclosing the content within a pair of listing block delimiters (`----`).

Listing style syntax

The block style `listing` can be applied to a block or paragraph, by setting the attribute `listing` using an attribute list.

Example 206. Listing style syntax

```
[listing]
This is an example of a paragraph assigned
the `listing` style in an attribute list.
```

```
Notice that the monospace marks are
preserved in the output.
```

The result of Example 206 is rendered below.

```
This is an example of a paragraph assigned
the `listing` style in an attribute list.
Notice that the monospace marks are
preserved in the output.
```

Delimited listing block

A delimited listing block is surrounded by lines composed of four hyphens (`----`). This method is useful when the content contains empty lines.

Example 207. Delimited listing block syntax

```
----
This is a _delimited listing block_.

The content inside is displayed as <pre> text.
----
```

Here's how the block in Example 207 appears when rendered.

```
This is a _delimited listing block_.

The content inside is displayed as <pre> text.
```

You should notice a few things about how the content is processed.

- The HTML element `<pre>` is escaped, that is, it's displayed verbatim, not interpreted.
- The endlines are preserved.
- The phrase *delimited listing block* isn't italicized, despite having the underscore formatting marks around it.

Listing blocks are good for displaying snippets of raw source code, especially when used in tandem with the `source` style and `source-highlighter` attribute. See Source Code Blocks to learn more about `source` and `source-highlighter`.

Listing substitutions

Content that is assigned the `listing` style, either via the explicit block style or the listing delimiters is subject to the verbatim substitution group. Only special characters and callouts are replaced automatically in listing content.

You can control the substitutions applied to a listing block using the `subs` attribute.

Example 208. Delimited listing block with custom substitutions syntax

```
[subs="+attributes"]
----
This is a _delimited listing block_
with the `subs` attribute assigned
the incremental value `+attributes`.
This attribute reference:

{replace-me}

will be replaced with the attribute's
value when rendered.
----
```

The result of Example 208 is rendered below.

```
This is a _delimited listing block_
with the `subs` attribute assigned
the incremental value `+attributes`.
This attribute reference:

I've been replaced!

will be replaced with the attribute's
value when rendered.
```

See Customize the Substitutions Applied to Blocks to learn more about the `subs` attribute and how to apply incremental substitutions to listing content.

Literal Blocks

Literal blocks display the text you write exactly as you see it in the source. Literal text is treated as preformatted text. The text is presented in a fixed-width font and endlines are preserved. Only special characters and callouts are replaced when the document is converted.

The literal style can be applied to content using any of the following methods:

- indenting the first line of a paragraph by one or more spaces,
- setting the `literal` style on a block using an attribute list, or
- enclosing the content within a pair of literal block delimiters (`....`).

Indent method

When a line begins with one or more spaces it is displayed as a literal block. This method is an easy

way to insert simple code snippets.

Example 209. Indicate literal text using an indent

```
 ~/secure/vault/defops
```

The result of Example 209 is rendered below.

```
~/secure/vault/defops
```

literal style syntax

The literal style can be applied to a block, such as a paragraph, by setting the style attribute `literal` on the block using an attribute list.

Example 210. Literal style syntax

```
[literal]
error: 1954 Forbidden search
absolutely fatal: operation lost in the dodecahedron of doom
Would you like to try again? y/n
```

The result of Example 210 is rendered below.

```
error: 1954 Forbidden search
absolutely fatal: operation lost in the dodecahedron of doom
Would you like to try again? y/n
```

Delimited literal block

Finally, you can surround the content you want rendered as literal by enclosing it in a pair of literal block delimiters (`....`). This method is useful when the content contains empty lines.

Example 211. Delimited literal block syntax

```
....
Kismet: Where is the *defensive operations manual*?

Computer: Calculating ...
Can not locate object.
You are not authorized to know it exists.

Kismet: Did the werewolves tell you to say that?

Computer: Calculating ...
....
```

The result of Example 211 is rendered below.

```
Kismet: Where is the *defensive operations manual*?

Computer: Calculating ...
Can not locate object.
You are not authorized to know it exists.

Kismet: 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.

Callouts

Callout numbers (aka callouts) provide a means to add annotations to lines in a verbatim block.

Callout syntax

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:

Example 212. Callout syntax

```
[source,ruby]
----
require 'sinatra' <1>

get '/hi' do <2> <3>
  "Hello World!"
end
----
<1> Library import
<2> URL mapping
<3> Response block
```

The result of Example 212 is rendered below.

```
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, an AsciiDoc processor provides several features to hide the callout numbers from both the source and the converted document. The sections that follow detail these features.

Automatic numbering

Just like ordered lists, it's possible to allow the processor to automatically number the callouts. To leverage this capability, you replace the numbers in each callout with a dot (e.g., `<.>`). Each time the processor comes across a callout in either the verbatim block or the callout list, it selects the next number in the sequence (scoped to that block), starting from 1.

Let's return to the previous example to see how it looks if we use automatic numbering.

Example 213. Callout syntax with automatic numbering

```
[,ruby]
----
require 'sinatra' <.>

get '/hi' do <.> <.>
  "Hello World!"
end
----
<.> Library import
<.> URL mapping
<.> Response block
```

The result is exactly the same as before.

Mixed numbering

The `<.>` callouts are automatically numbered based on their sequence among other `<.>`, not any callouts that have an explicit number. In other words, the automatic numbering is not aware of any explicit numbering. Therefore, you should generally avoid mixing them.

However, if you want to repeat a number in the verbatim block, then you can use an explicit number to create additional occurrences of that callout number.

Let's consider an example:

Example 214. Callout syntax with mixed numbering

```
[,ruby]
----
require 'asciidoctor' <.>

puts Asciidoctor::VERSION <1>

Asciidoctor.convert_file 'README.adoc' <.>
----
<.> The require statement exports the class from the gem to the global scope.
<.> We can then call methods provided by that class.
```

The result of Example 214 is rendered below.

```ruby
require 'asciidoctor' ①

puts Asciidoctor::VERSION ①

Asciidoctor.convert_file 'README.adoc' ②
```

① The require statement exports the class from the gem to the global scope.

② We can then call methods provided by that class.

The risk of this approach is that you have to keep track of which numbers are being assigned automatically.

Copy and paste friendly callouts

If you add callout numbers to example code in a verbatim (e.g., source) block, and a reader selects that source code in the generated HTML, we don't want the callout numbers to get caught up in the copied text. If the reader pastes that example code into a code editor and tries to run it, the extra characters that define the callout numbers will likely lead to compile or runtime errors. To mitigate this problem, and AsciiDoc processor uses a CSS rule to prevent the callouts from being selected. That way, the callout numbers won't get copied.

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. When font-based icons are enabled (e.g., `icons=font`), the AsciiDoc processor will recognize the line comments characters in front of a callout number—optionally offset by a space—and remove them when converting the document. When font-based icons aren't enabled, the line comment characters are not removed so that the callout numbers remain hidden by the line comment.

Here are the line comments that are supported:

Example 215. Prevent callout copy and paste

```
----
line of code // <1>
```

```
line of code # <2>
line of code ;; <3>
line of code <!--4-->
----
<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.
<4> A callout behind a line comment for XML or SGML languages like HTML.
```

The result of Example 215 is rendered below.

```
line of code ①
line of code ②
line of code ③
line of code ④
```

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.

④ A callout behind a line comment for XML or SGML languages like HTML.

Custom line comment prefix

An AsciiDoc processor recognizes the most ubiquitous line comment prefixes as a convenience. If the source language you're embedding does not support one of these line comment prefixes, you can customize the prefix using the `line-comment` attribute on the block.

Let's say we want to tuck a callout behind a line comment in Erlang code. In this case, we would set the `line-comment` character to `%`, as shown in this example:

Example 216. Custom line comment prefix

```
[source,erlang,line-comment=%]
----
-module(hello_world).
-compile(export_all).

hello() ->
    io:format("hello world~n"). % <1>
----
<1> A callout behind a custom line comment prefix.
```

The result of Example 216 is rendered below.

```
-module(hello_world).
-compile(export_all).
```

```
hello() ->
    io:format("hello world~n"). % ①
```

① A callout behind a custom line comment prefix.

Even though it's not specified in the attribute, one space is still permitted immediately following the line comment prefix.

Disable line comment processing

If the source language you're embedding does not support trailing line comments, or the line comment prefix is being misinterpreted, you can disable this feature using the `line-comment` attribute.

Let's say we want to put a callout at the end of a block delimiter for an open block in AsciiDoc. In this case, the processor will think the double hyphen is a line comment, when in fact it's the block delimiter. We can disable line comment processing by setting the `line-comment` character to an empty value, as shown in this example:

Example 217. No line comment prefix

```
[source,asciidoc,line-comment=]
----
-- <1>
A paragraph in an open block.
--
----
<1> An open block delimiter.
```

The result of Example 217 is rendered below.

```
①
A paragraph in an open block.
--
```

① An open block delimiter.

Since the language doesn't support trailing line comments, there's no way to hide the callout number in the raw source.

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 XML comment and callout number.

Here's how it appears in a listing:

Example 218. XML callout syntax

```
[source,xml]
```

```
----
<section>
  <title>Section Title</title> <!--1-->
</section>
----
<1> The section title is required.
```

The result of Example 218 is rendered below.

```
<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.

Callout icons

The font icons setting also enables callout icons drawn using CSS.

```
= Document Title
:icons: font ①

NOTE: Asciidoctor 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.

Tables

Build a Basic Table

A table is a delimited block that can have optional customizations, such as an ID and a title, as well as table-specific attributes, options, and roles. However, at its most basic, a table only needs columns and rows.

On this page, you'll learn:

- ☑ How to set up an AsciiDoc table block and its attribute list.
- ☑ How to add columns to a table using the `cols` attribute.
- ☑ How to add cells to a table and arrange them into rows.
- ☑ How to designate a row as the table's header row.

Create a table with two columns and three rows

In Example 219, we'll assign the `cols` attribute a list of column specifiers. A column specifier represents a column.

Example 219. Set up a table with two columns

```
[cols="1,1"] ① ②
|=== ③
```

① On a new line, create an attribute list. Set the `cols` attribute, followed by an equals sign (`=`).

② Assign a list of comma-separated column specifiers enclosed in double quotation marks (`"`) to `cols`. Each column specifier represents a column.

③ On the line directly after the attribute list, enter the opening table delimiter. A table delimiter is one vertical bar followed by three equals signs (`|===`). This delimiter starts the table block.

The table in Example 219 will contain two columns because there are two comma-separated entries in the list assigned to `cols`. Each entry in the list is called a column specifier. A **column specifier** represents a column and the width, alignment, and style properties assigned to that column. When each column specifier is the same number, in this case the integer `1`, all of the columns' widths will be identical. Each column in Example 219 will be the same width regardless of how much content they contain.

Next, let's add three rows to the table. Each row has the same number of cells. Since the table in Example 220 has two columns, each row will contain two cells. A cell starts with a vertical bar (`|`).

Example 220. Add three rows to the table

```
[cols="1,1"]
|===
|Cell in column 1, row 1 ①
|Cell in column 2, row 1 ②
```

```
③
|Cell in column 1, row 2
|Cell in column 2, row 2

|Cell in column 1, row 3
|Cell in column 2, row 3 ④
|=== ⑤
```

① To create a new cell, press `Shift` + `|`. After the vertical bar (`|`), enter the content you want displayed in that cell.

② On a new line, start another cell with a `|`. Each consecutive cell is placed in a separate, consecutive column in a row.

③ Rows are separated by one or more empty lines.

④ When you finish adding cells to your table, press `Enter` to go to a new line.

⑤ Enter the closing delimiter (`|===`) to end the table block.

The suggestion to start each cell on its own line and to separate rows by empty lines is merely a stylistic choice. You can enter more than one cell or all of the cells in a row on the same line since the processor creates a new cell each time it encounters a vertical bar (`|`).

The table from Example 220 is displayed below. It contains two columns and three rows of text positioned and styled using the default alignment, style, border, and width attribute values.

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

In addition to the cols attribute, you can identify the number of columns using a column multiplier or the table's first row. However, the `cols` attribute is required to customize the width, alignment, or style of a column.

Add a header row to the table

Let's add a header row to the table in Example 221. You can implicitly identify the first row of a table as a header row by entering all of the first row's cells on the line directly after the opening table delimiter.

Example 221. Add a header row to the table

```
[cols="1,1"]
|===
|Cell in column 1, header row |Cell in column 2, header row ①
②
|Cell in column 1, row 2
|Cell in column 2, row 2
```

```
|Cell in column 1, row 3
|Cell in column 2, row 3

|Cell in column 1, row 4
|Cell in column 2, row 4
|===
```

① On the line directly after the opening delimiter (`|===`), enter all of the first row's cells on a single line.

② Leave the line directly after the header row empty.

The table from Example 221 is displayed below.

Cell in column 1, header row	Cell in column 2, header row
Cell in column 1, row 2	Cell in column 2, row 2
Cell in column 1, row 3	Cell in column 2, row 3
Cell in column 1, row 4	Cell in column 2, row 4

A header row can also be identified by assigning header to the options attribute.

Add a Title

A table can have an optional title (i.e., table caption). To add a title to a table, use the block title syntax.

Example 222. Add an optional title to a table

```
.A table with a title ①
[%autowidth]
|===
|Column 1, header row |Column 2, header row

|Cell in column 1, row 2
|Cell in column 2, row 2
|===
```

① On the line directly above the table's opening delimiter (or above its optional attribute line, as shown here), enter a dot (`.`) directly followed by the text of the title.

The table from Example 222 is displayed below.

Table 2. A table with a title

Column 1, header row	Column 2, header row
Cell in column 1, row 2	Cell in column 2, row 2

You'll notice in the above result, that the processor automatically added *Table 1.* in front of the table's title. This title label can be customized or deactivated.

Customize the Title Label

When you add a title to a table, the processor automatically prefixes it with the label *Table <n>.*, where <*n*> is the 1-based index of all of the titled tables in the document. This label can be modified at the document level or per table. It can also be deactivated.

Modify the label using table-caption

You can change the label for all titled tables using the document attribute `table-caption`. (Don't let the attribute's name mislead you. It's the attribute that controls the table title labels at the document level.)

In the document header, set the `table-caption` attribute and assign it your custom label text.

```
= Document Title
:table-caption: Data Set ① ②
```

① Set the document attribute `table-caption` and assign it the text you want to precede each table title.

② Don't enter a number after the label text. The processor will automatically insert and increment the number.

In Example 223, the first and third tables have a title, but the second table doesn't have a title.

Example 223. Add two titled tables and one untitled table to a document

```
= Document Title
:table-caption: Data Set

.A table with a title
[cols="2,1"]
|===
|Lots and lots of data |A little data

|834,734 |3
|3,999,271.5601 |5
|===

|===
|Group |Climate |Example

|A
|Tropical
|Suva, Fiji

|B
|Arid
|Lima, Peru
|===
```

```
.Another table with a title
|===
|Value |Result |Notes

|Null |A mystery |See Appendix R
|===
```

Since `table-caption` is assigned the value `Data Set`, any table title should be preceded with the label *Data Set <n>*. The three tables from Example 223 are displayed below.

Data Set 3. A table with a title

Lots and lots of data	A little data
834,734	3
3,999,271.5601	5

Group	Climate	Example
A	Tropical	Suva, Fiji
B	Arid	Lima, Peru

Data Set 4. Another table with a title

Value	Result	Notes
Null	A mystery	See Appendix R

Notice that the table that doesn't have a title didn't get a label nor was it counted when the processor incremented the label number. Therefore, the third table is assigned the label *Data Set 2*.

Modify the label of an individual table using caption

You can customize the label on an individual table by setting 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). When using `caption`, assign it the exact value you want displayed (including trailing spaces). Labels assigned using `caption` don't get an automatically incremented number and only apply to the table they are set on.

If you want a space between the label and the title, you must add a trailing space to the value of the caption attribute.

Example 224. Modify the label using caption

```
[caption="Table A. "] ① ②
.A table with a custom label
[cols="3*"]
|===
|Null
|A mystery
|See Appendix R
```

```
|===
```

① Create an attribute list directly above the table's title and set the named attribute `caption`, followed by an equals sign (`=`), and then a value.

② Enclose the value in double quotation marks (`"`). Otherwise the processor will remove any trailing whitespaces, and the title text will start directly after the last character of the label.

The table from Example 224 is displayed below.

Table A. A table with a custom label

Null	A mystery	See Appendix R

If you create any subsequent tables in your document and don't set `caption` on them, the title labels will revert to the value assigned to `table-caption`.

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}"]
[%header,cols=2*]
|===
|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:

```
.{empty}
[caption="{table-caption} {counter:table-number}"]
[%header,cols=2*]
|===
|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
|===
```

Turn Off the Title Label

Disable the label using table-caption

You can disable the title label for all of the tables in a document by unsetting the `table-caption` document attribute.

```
= Title of Document
:table-caption!: ①

.A table with a title but no label
|===
|Value |Result |Notes

|Null |A mystery |See Appendix R
|===
```

① In an attribute entry, enter the name of the attribute, `table-caption`, and append a bang (`!`) to the end of the name. This unsets the attribute.

When `table-caption` is unset, table titles aren't preceded by a label and label number.

A table with a title but no label

Value	Result	Notes
Null	A mystery	See Appendix R

Disable the label using caption

To remove the label on an individual table, assign an empty value to the `caption` attribute.

```
[caption=] ①
.A table with a title but no label
[cols="2,1"]
|===
|Lots and lots of data |A little data

|834,734 |3
|3,999,271.5601 |5
|===
```

① Enter the attribute's name, `caption`, in an attribute list directly above the table title, followed by an equals sign (`=`). Don't enter a value after the `=`.

The table from the previous example is displayed below.

A table with a title but no label

Lots and lots of data	A little data
834,734	3
3,999,271.5601	5

Add Columns to a Table

The number of columns in a table is specified by the `cols` attribute or by the number of cells found in the first non-empty line after the opening table delimiter (`|===`).

Specify the number of columns with the cols attribute

The `cols` attribute is set in the attribute list on a table block. It accepts a comma-separated list of column specifiers. Each **column specifier** represents a column and the width, alignment, and style properties assigned to that column. A column specifier is commonly represented by a number, but in some cases, can be represented by a symbol or letter. In Example 225, `cols` is assigned a list of four numeric column specifiers.

Example 225. Assign column specifiers to the cols attribute

```
[cols="1,1,1,1"]
```

In Example 225, the value assigned to `cols` contains four column specifiers. The number of entries in the value's list determines the number of columns in the table. That means the table in the above example will contain four columns. When the specifier is a number, such as `1` or `50`, the integer represents the width of the column in proportion to the other columns in the table. In Example 225, each column will be the same width because the integer in each specifier is the same. Let's look at the column specifiers in Example 226 and compare it to Example 225.

Example 226. Assign column specifiers to the cols attribute

```
[cols="3,3,3,3"]
```

Both Example 225 and Example 226 will produce tables with four columns of equal width. Let's use the `cols` value in Example 226 to create a table.

Example 227. Create a table with four columns of equal width

```
[cols="3,3,3,3"] ①
|=== ②
|Column 1 |Column 2 |Column 3 |Column 4 ③
④
|Cell in column 1 ⑤
|Cell in column 2
|Cell in column 3
|Cell in column 4
|=== ⑥
```

① In an attribute list, set the `cols` attribute, followed by an equals sign (`=`), and then a list of comma-separated column specifiers enclosed in double quotation marks (`"`).

② On the line directly after the attribute list, enter the opening table delimiter. A table delimiter is one vertical bar followed by three equals signs (`|===`).

③ A table cell is specified by a vertical bar (`|`). Since four consecutive cells are entered on the first line directly after the delimiter, this row is implicitly set as the table's header row.

④ Insert an empty line after the header row.

⑤ The cells for the next row can be entered on a single line or on individual lines.

⑥ On a new line after the last cell of the last row, enter another table delimiter (`|===`) to close the table block.

Example 227 creates the table displayed below.

Result of Example 227

Column 1	Column 2	Column 3	Column 4
Cell in column 1	Cell in column 2	Cell in column 3	Cell in column 4

As specified, the table includes four columns of equal width, a header row, and a regular row. Since all of the columns in Example 227 are assigned the same width via their column specifiers (i.e., 3), the number of columns could be specified with a column multiplier. Or, you could adjust the width of an individual column by increasing the numerical value of its specifier.

Using a column multiplier

A **column multiplier** allows you to apply the same width, horizontal alignment, vertical alignment, and content style to multiple, consecutive columns in a table. A multiplier consists of an integer (`<n>`) and an asterisk (`*`). The integer represents the number of consecutive columns to be added to the table. The asterisk (`*`) is called the **multiplier operator** and is placed directly after the integer (`<n>*`). The operator tells the converter to interpret the integer as part of a column multiplier instead of a column specifier.

For example, let's rewrite the value of `[cols="5,5,5"]` as a column multiplier.

Example 228. Represent [cols="5,5,5"] using a column multiplier

```
[cols="3*"] ①
```

① Assign an integer to `cols` that represents the number of columns in the table. Enter the multiplier operator (`*`) directly after the integer.

The integer 3, combined with the * operator, indicates that the table will contain three columns of equal width.

You can use a multiplier in a comma-separated list with column specifiers, too. In Example 229, the first column is represented by a column specifier, and the next three columns are represented by a multiplier.

Example 229. Assign a column specifier and a column multiplier to cols

```
[cols="5,3*"]
|===
|Column 1 |Column 2 |Column 3 |Column 4

|Cell in column 1
|Cell in column 2
|Cell in column 3
|Cell in column 4
|===
```

As shown below, Example 229 creates a table containing a wide first column followed by three columns of equal width.

Result of Example 229

Column 1	Column 2	Column 3	Column 4
Cell in column 1	Cell in column 2	Cell in column 3	Cell in column 4

Alignment and style column operators

AsciiDoc provides operators that control the positioning and style of column content when the `cols` attribute is set. A column specifier or multiplier can contain these optional operators for one or more of the following properties:

- horizontal alignment
- vertical alignment
- content style

Many of these operators can be applied to individual cells as well.

Specify the number of columns using the first row

When all of the columns in a table use the default width, alignment, and style values, you don't need to set the `cols` attribute. Instead, you can implicitly declare the number of columns by entering all of the first row's cells on the same line. The processor will derive the number columns from the number of cells in this row. Example 230 uses its first row to indicate that it has three columns.

Example 230. Create a table with three columns using its first row

```
|===
①
|Cell in column 1, row 1 |Cell in column 2, row 1 |Cell in column 3, row 1 ②

|Cell in column 1, row 2 ③
|Cell in column 2, row 2
|Cell in column 3, row 2
```

```
|===
```

① After the opening delimiter, insert an empty line before the first row, unless you want the first row to be treated as header row.

② Enter all of the first row's cells on a single line. Each cell represents one column.

③ The cells in subsequent rows don't need to be entered on a single line.

The table in Example 230 has three columns since its first row contains three cells.

Result of Example 230

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

Adjust Column Widths

Column width

The width of a column is assigned by its column specifier. The value of a column's width is either an integer or a percentage. The default column width is 1. The integer or percentage represents the width of the column in proportion to the other columns within the total width of the table. The total width of a table is backend dependent. When using the HTML5 backend with the default Asciidoctor stylesheet, tables stretch the width of the page body unless the table width attribute is explicitly set.

Assign column widths using integers

To assign widths to the columns in a table, set the `cols` attribute and assign it a list of comma-separated column specifiers using integers.

Example 231. Assign column widths using integers

```
[cols="2,1,3"]
|===
|Column 1 |Column 2 |Column 3

|This column has a proportional width of 2
|This column has a proportional width of 1
|This column has a proportional width of 3
|===
```

As seen below, the columns stretch across the width of the page according to their proportional widths.

Result of Example 231

Column 1	Column 2	Column 3
This column has a proportional width of 2	This column has a proportional width of 1	This column has a proportional width of 3

Increase or decrease the width of a column

To increase the width of a column, use a bigger integer in the column's specifier. Let's make column 1 from Example 231 the largest column in the table by increasing its width from 2 to 6 in Example 232.

Example 232. Increase the width of a column

```
[cols="6,1,3"]
|===
|Column 1 |Column 2 |Column 3

|This column has a proportional width of 6
|This column has a proportional width of 1
|This column has a proportional width of 3
|===
```

Below, the result of Example 232 shows that column 1 is now much wider than column 3.

Result of Example 232

Column 1	Column 2	Column 3
This column has a proportional width of 6	This column has a proportional width of 1	This column has a proportional width of 3

To decrease the width of a column, use a smaller integer in its specifier. In Example 233, let's make the width of column 3 smaller, but not quite as small as column 2, by decreasing its width from 3 to 2.

Example 233. Decrease the width of a column

```
[cols="6,1,2"]
|===
|Column 1 |Column 2 |Column 3

|This column has a proportional width of 6
|This column has a proportional width of 1
```

```
|This column has a proportional width of 2
|===
```

The columns, displayed in the table below, have adjusted across the width of the page according to their proportional widths.

Result of Example 233

Column 1	Column 2	Column 3
This column has a proportional width of 6	This column has a proportional width of 1	This column has a proportional width of 2

Change column widths using percentage values

Column widths can be assigned using a percentage between `1%` and `100%`. Like with integer values, set `cols` and assign it a list of comma-separated column specifiers using percentages.

Example 234. Assign column widths using percentages

```
[cols="15%,30%,55%"]
|===
|Column 1 |Column 2 |Column 3

|This column has a width of 15%
|This column has a width of 30%
|This column has a width of 55%
|===
```

As seen in the table below, the columns stretch across the width of the page according to the percentage assigned via their column specifiers.

Result of Example 234

Column 1	Column 2	Column 3
This column has a width of 15%	This column has a width of 30%	This column has a width of 55%

When assigning percentages to `cols`, you don't have to include the percent sign (`%`). For instance, both `[cols="15%,30%,55%"]` and `[cols="15,30,55"]` are valid.

Align Content by Column

The alignment operators allow you to horizontally and vertically align a column's content. They're applied to a column specifier and assigned to the cols attribute.

Horizontal alignment operators

Content can be horizontally aligned to the left or right side of the column as well as the center of the column.

Flush left operator (<)

The less-than sign (<) left aligns the content. This is the default horizontal alignment.

Flush right operator (>)

The greater-than sign (>) right aligns the content.

Center operator (^)

The caret (^) centers the content.

A horizontal alignment operator is entered in front a vertical alignment operator (if present) and in front of a column's width (if present). If the number of columns is assigned using a multiplier (<n>*), the horizontal alignment operator is placed directly after the multiplier operator (*).

- `[cols="2,^1"]` A horizontal alignment operator is placed in front of the column width.
- `[cols=">.^1,2"]` A horizontal alignment operator is placed in front of a vertical alignment operator.
- `[cols=">,^"]` When a column width isn't specified, a horizontal alignment operator can represent both the column and the column content's alignment.
- `[cols="3*>"]` The horizontal alignment operator is placed directly after a multiplier.

Center content horizontally in a column

To horizontally center the content in a column, place the ^ operator at the beginning of the column's specifier.

Example 235. Center column content horizontally

```
[cols="^4,1"]
|===
|This content is horizontally centered.
|There isn't a horizontal alignment operator on this column's specifier, so the column
falls back to the default horizontal alignment.
Content is left-aligned by default.
|===
```

The table from Example 235 is rendered below.

Result of Example 235

This content is horizontally centered.	There isn't a horizontal alignment operator on this column's specifier, so the column falls back to the default horizontal alignment. Content is left-aligned by default.

When the columns are specified using the multiplier, place the ^ operator after the multiplier operator (*).

Example 236. Horizontal alignment and multiplier operator order

```
[cols="2*^",options=header]
|===
|Column name
|Column name

|This content is horizontally centered.
|This content is also horizontally centered.
|===
```

The table from Example 236 is rendered below.

Result of Example 236

Column name	Column name
This content is horizontally centered.	This content is also horizontally centered.

Right align content in a column

To align the content in a column to its right side, place the > operator in front of the column's specifier.

Example 237. Right align column content

```
[cols=">4,1"]
|===
|This content is aligned to the right side of the column.
|There isn't a horizontal alignment operator on this column's specifier, so the column
falls back to the default horizontal alignment.
Content is left-aligned by default.
|===
```

The table Example 237 is rendered below.

Result of Example 237

This content is aligned to the right side of the column.	There isn't a horizontal alignment operator on this column's specifier, so the column falls back to the default horizontal alignment. Content is left-aligned by default.

When the columns are specified using the multiplier, place the `>` operator after the multiplier operator (`*`).

Example 238. Right alignment and multiplier operator order

```
[cols="2*>",options=header]
|===
|Column name
|Column name

|This content is aligned to the right side of the column.
|This content is also aligned to the right side of the column.
|===
```

The table from Example 238 is rendered below.

Result of Example 238

Column name	Column name
This content is aligned to the right side of the column.	This content is also aligned to the right side of the column.

Vertical alignment operators

Content can be vertically aligned to the top or bottom of a column's cells as well as the center of a column. Vertical alignment operators always begin with a dot (`.`).

Flush top operator (.<)

The dot and less-than sign (`.<`) aligns the content to the top of the column's cells. This is the default vertical alignment.

Flush bottom operator (.>)

The dot and greater-than sign (`.>`) aligns the content to the bottom of the column's cells.

Center operator (.^)

The dot and caret (`.^`) centers the content vertically.

A vertical alignment operator is entered directly after a horizontal alignment operator (if present) and before a column's width (if present). If the number of columns is assigned using a multiplier (<n>*), the vertical alignment operator is placed directly after the horizontal alignment operator (if present). Otherwise, it's placed directly after the multiplier operator (*).

- `[cols="2,.^1"]` A vertical alignment operator is placed in front of the column width.
- `[cols=">.^1,2"]` The vertical alignment operator is placed after the horizontal alignment operator but before the column width.
- `[cols=".^,.>"]` When a column width doesn't need to be specified, a vertical alignment operator can represent both the column and the column content's alignment.
- `[cols="3*.>"]` The vertical alignment operator is placed directly after a multiplier unless there is a horizontal alignment operator. Then it's placed after the horizontal alignment operator, (e.g., `[cols="3*^.>"]`)

Align content to the bottom of a column's cells

To align the content in a column to the bottom of each cell, place the `.>` operator directly in front of the column's width.

Example 239. Bottom align column content

```
[cols=".>2,1"]
|===
|This content is vertically aligned to the bottom of the cell.
|There isn't a vertical alignment operator on this column's specifier, so the column
falls back to the default vertical alignment.
Content is top-aligned by default.
|===
```

The table from Example 239 is rendered below.

Result of Example 239

This content is vertically aligned to the bottom of the cell.	There isn't a vertical alignment operator on this column's specifier, so the column falls back to the default vertical alignment. Content is top-aligned by default.

Center content vertically in a column

To vertically center the content in a column, place the `.^` operator directly in front of the column's width.

Example 240. Center column content vertically

```
[cols=".^2,1"]
|===
```

```
|This content is centered vertically in the cell.
|There isn't a vertical alignment operator on this column's specifier, so the column
falls back to the default vertical alignment.
Content is top-aligned by default.
|===
```

The table from Example 240 is rendered below.

Result of Example 240

This content is centered vertically in the cell.	There isn't a vertical alignment operator on this column's specifier, so the column falls back to the default vertical alignment. Content is top-aligned by default.

To vertically align the content to the middle of the cells in all of the columns, enter the `.^` operator after the multiplier.

Example 241. Vertical alignment and multiplier operator order

```
[cols="2*.^",options=header]
|===
|Column name
|Column name

|This content is vertically centered.
|This content is also vertically centered.
|===
```

The table from Example 241 is rendered below.

Result of Example 241

Column name	Column name
This content is centered vertically in the cell.	This content is also centered vertically in the cell.

When a horizontal alignment operator is also applied to the multiplier, then the vertical alignment operator is placed directly after the horizontal operator (e.g., `[cols="2*>.^"]`).

Apply horizontal and vertical alignment operators to the same column

A column can have a vertical and horizontal alignment operator placed on its specifier. The horizontal operator always precedes the vertical operator. Both operators precede the column width. When a multiplier is used, the operators are placed after the multiplier.

Example 242. Horizontally and vertically align column content

```
[cols="^.>2,1,>.^1"]
|===
|Column name |Column name |Column name

|This content is centered horizontally and aligned to the bottom
of the cell.
|There aren't any alignment operators on this column's specifier,
so the column falls back to the default alignments.
The default horizontal alignment is left-aligned.
The default vertical alignment is top-aligned.
|This content is aligned to the right side of the cell and
centered vertically.
|===
```

The table from Example 242 is rendered below.

Result of Example 242

Column name	Column name	Column name
This content is centered horizontally and aligned to the bottom of the cell.	There aren't any alignment operators on this column's specifier, so the column falls back to the default alignments. The default horizontal alignment is left-aligned. The default vertical alignment is top-aligned.	This content is aligned to the right side of the cell and centered vertically.

If there is an alignment operator on a cell's specifier, it will override the column's alignment operator.

Format Content by Column

A column style operator is applied to a column specifier and assigned to the cols attribute.

Column styles and their operators

You can style all of the content in a column by adding a style operator to a column's specifier.

Style	Operator	Description
AsciiDoc	`a`	Supports block elements (lists, delimited blocks, and block macros). This style effectively creates a nested, standalone AsciiDoc document. The parent document's implicit attributes, such as `doctitle`, are shadowed and custom attributes are inherited.
Default	`d`	All of the markup that is permitted in a paragraph (i.e., inline formatting, inline macros) is supported.
Emphasis	`e`	Text is italicized.
Header	`h`	Applies the header semantics and styles to the text and cell borders.
Literal	`l`	Content is treated as if it were inside a literal block.
Monospace	`m`	Text is rendered using a monospace font.
Strong	`s`	Text is bold.

When a style operator isn't explicitly applied to a column specifier, the `d` style is assigned automatically and the column is processed as paragraph text.

Apply a style operator to a column

A style operator is always placed in the last position on a column's specifier or multiplier.

- `[cols=">e,.^3s"]` A style operator is placed directly after any other operators and the column width in the column's specifier.
- `[cols="h,e"]` When a column width isn't specified, the style operator can represent both the column and the column's content style.
- `[cols="3*.>m"]` When a multiplier is present, the style operator is placed after any horizontal and vertical alignment operators.

Let's apply a different style to each column in Example 243.

Example 243. Add a style operator to each column

```
[cols="h,m,s,e"]
|===
|Column 1 |Column 2 |Column 3 |Column 4
```

```
|This column's content and borders are rendered using the table header (`h`) styles.
|This column's content is rendered using a monospace font (m).
|This column's content is bold (`s`).
|This column's content is italicized (`e`).

|This column's content and borders are rendered using the table header (`h`) styles.
|This column's content is rendered using a monospace font (m).
|This column's content is bold (`s`).
|This column's content is italicized (`e`).
|===
```

The table from Example 243 is displayed below. Note that the style applied to each column doesn't affect the header row or override any inline formatting.

Result of Example 243

Column 1	Column 2	Column 3	Column 4
This column's content and borders are rendered using the table header (h) styles.	`This column's content is rendered using a monospace font (m).`	**This column's content is bold (s).**	*This column's content is italicized (e).*
This column's content and borders are rendered using the table header (h) styles.	`This column's content is rendered using a monospace font (m).`	**This column's content is bold (s).**	*This column's content is italicized (e).*

Additionally, if a cell specifier contains a style operator, that style will override a column's style operator.

Use AsciiDoc block elements in a column

To use AsciiDoc block elements, such as delimited source blocks and lists, in a column, place the lowercase letter `a` on the column specifier.

Example 244. Apply the AsciiDoc block style operator to the first column

```
[cols="2a,2"]
|===
|Column with the `a` style operator applied to its specifier |Column using the default
style

|
* List item 1
* List item 2
* List item 3
|
* List item 1
* List item 2
* List item 3
```

```
|
[source,python]
----
import os
print "%s" %(os.uname())
----
|
[source,python]
----
import os
print ("%s" %(os.uname()))
----
|===
```

The AsciiDoc block style effectively creates a nested, standalone AsciiDoc document in each cell in the column. The parent document's implicit attributes, such as `doctitle`, are shadowed and custom attributes are inherited.

Result of Example 244

Column with the `a` style operator applied to its specifier	Column using the default style
• List item 1 • List item 2 • List item 3	* List item 1 * List item 2 * List item 3
`import os` `print "%s" %(os.uname())`	[source,python] ---- import os print ("%s" %(os.uname())) ----

You can also apply the AsciiDoc block style operator to individual cells.

Add Cells and Rows to a Table

Table cells

Each new cell in a table is declared with a cell separator. The default **cell separator** is a vertical bar (|). All of the content entered after a cell separator is included in that cell until the processor encounters a space followed by another vertical bar (|) or a new line that begins with a |.

Example 245. Creating table cells with the default cell separator

```
[cols="3,2,3"]
|===
|This content is placed in the first cell of column 1
|This line starts with a vertical bar so this content is placed in a new cell in
column 2 |When the processor encounters a whitespace followed by a vertical bar it
ends the previous cell and starts a new cell
```

```
|===
```

When the processor encounters another |, it creates a new cell in the next consecutive column. Once the processor reaches the last column assigned to the table, the next cell it encounters is placed in a new row. Taking into account any spans, which are applied via a cell specifier, each row consists of the same number of cells.

Cell specifiers and operators

A **cell specifier** is a positional attribute placed directly in front of a cell separator that represents the position and style properties assigned to a cell's content. In the example below, a horizontal alignment operator and style operator have been assigned to the first cell's specifier.

Example 246. Using cell specifiers

```
[cols="2*"]
|===
>s|This cell's specifier indicates that this cell's content is right-aligned and bold.
|The cell specifier on this cell hasn't been set explicitly, so the  default
properties are applied.
|===
```

AsciiDoc provides operators to control the following cell properties:

- span
- duplication
- horizontal alignment
- vertical alignment
- content style

A cell specifier only applies to the cell it's placed on, not to all of the cells in the same row. Also, the operator in a cell specifier will override the operator in a column specifier if they belong to the same property.

Create a table cell

In this section, we'll set up a table and add two rows of cells to it. First, let's create two cells in Example 247 and see how they get arranged into a row.

Example 247. Add two cells to a table

```
[cols="1,1"] ①
|===
|This cell is in column 1, row 1 ②
|This cell is in column 2, row 1 ③
|===
```

① The table has two columns because two column specifiers are assigned to the `cols` attribute.

② The processor places this cell in the first column and row of the table because the vertical bar (`|`) at the beginning of this cell is the first cell separator the processor encounters after the opening table delimiter (`|===`).

③ This is the second `|` the processor encounters, so this cell is placed in the second column of the first row.

Though the two cells in Example 247 were entered on separate lines, the processor places them in the same row.

Result of Example 247

This cell is in column 1, row 1	This cell is in column 2, row 1

Since the number of columns in Example 247 is set to two by the `cols` attribute, and there are two cells entered in the table, the first row is complete. Now, let's add two more cells to the table.

Example 248. Add two more cells to a table

```
[cols="1,1"]
|===
|This cell is in column 1, row 1
|This cell is in column 2, row 1
①
|This cell is in column 1, row 2 ②
|    This cell is in column 2, row 2 ③
|===
```

① Separate rows by one or more empty lines.

② The processor places this cell on the second row because the table has two columns and this is the third cell separator (`|`) it encounters.

③ Any leading or trailing spaces around the cell content is stripped by the processor.

The table from Example 248 now has four cells arranged into two consecutive rows.

Result of Example 248

This cell is in column 1, row 1	This cell is in column 2, row 1
This cell is in column 1, row 2	This cell is in column 2, row 2

The cells in a row can be entered on the same line or consecutive lines because the row a cell in placed on is determined by the number of columns in a table and the order in which the processor encounters the cell's separator (`|`).

Enter a row's cells on a single line

You can enter a row's cells on a single line. When entering cells on a single line, **at least one space must be entered between the last character of the previous cell's content and the cell separator (`|`) of the next cell**.

Example 249. Cells entered on the same line

```
|===
|Column 1 |Column 2 |Column 3 ① ②

|Cell in column 1, row 2 |Cell in column 2, row 2 |Cell in column 3, row 2 ③

|Cell in column 1, row 3 ④
|Cell in column 2, row 3 |Cell in column 3, row 3
|===
```

① Since `cols` is not set, the first row in this table must have the cells entered on a single line in order to implicitly assign three columns to the table.

② The first row is entered on the line directly after the opening table delimiter (`|===`) and followed by an empty line. This automatically assigns the `header` option to it.

③ When multiple cells are entered on a single line, enter at least one space between the last character of the previous cell's content and the cell separator (`|`) of the next cell.

④ A row's cells can be entered on a combination of lines as long as the lines are consecutive.

The table created in Example 249 contains three columns and three rows.

Result of Example 249

Column 1	Column 2	Column 3
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2
Cell in column 1, row 3	Cell in column 2, row 3	Cell in column 3, row 3

Any leading and trailing spaces around the cell content are stripped and don't affect the table's layout when rendered.

Enter a row's cells on consecutive lines

The cells in a row can be entered on consecutive, individual lines. When using this method, remember to either set the cols attribute or enter the first row's cells on a single line.

Example 250. Cells on consecutive, 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
|===
```

The `cols` attribute in Example 250 is assigned a multiplier of `3*`, indicating that this table has three columns.

Result of Example 250

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

Entering cells on consecutive lines can improve the readability (and debugging) of your raw AsciiDoc content when you're applying specifiers to the cells, using AsciiDoc block elements in the cells, or entering a lot of content into the cells.

Create a Header Row

The first row of a table is promoted to a header row if the `header` value is assigned to the table's `options` attribute. You can assign `header` to a table's first row explicitly or implicitly.

The header row ignores any style operators assigned via column and cell specifiers. It also ignores alignment operators assigned to the table's column specifiers; however, any alignment operators assigned to a cell specifier in the header row are applied.

Explicitly assign header to the first row

The header row semantics and styles are explicitly assigned to the first row in a table by assigning `header` to the `options` attribute. The `options` attribute is set in the table's attribute list using the shorthand (`%value`) or formal syntax (`options="value"`).

The `options` attribute is represented by the percent sign (`%`) when it's set using the shorthand syntax. In Example 251, `header` is assigned to using the shorthand syntax for `options`.

Example 251. Table with `header` assigned using the shorthand syntax

```
[%header,cols="2,2,1"] ①
|===
|Column 1, header row
|Column 2, header row
|Column 3, header row

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

① Values assigned using the shorthand syntax must be entered before the `cols` attribute (or any other named attributes) in a table's attribute list, otherwise the processor will ignore them.

The table from Example 251 is displayed below.

Result of Example 251

Column 1, header row	Column 2, header row	Column 3, header row
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

In Example 252, the `options` attribute is set and assigned the `header` value using the formal syntax. The `options` attribute accepts a comma-separated list of values.

Example 252. Table with header assigned to the options attribute

```
[cols="2*",options="header"]
|===
|Column 1, header row
|Column 2, header row

|Cell in column 1, row 2
|Cell in column 2, row 2

|Cell in column 1, row 3
|Cell in column 2, row 3
|===
```

The first row of the table in Example 252 is rendered using the corresponding header styles and semantics.

Result of Example 252

Column 1, header row	Column 2, header row
Cell in column 1, row 2	Cell in column 2, row 2
Cell in column 1, row 3	Cell in column 2, row 3

Implicitly assign header to the first row

You can implicitly define a header row based on how you layout the table. The following conventions determine when the first row automatically becomes the header row:

1. The first line of content inside the table delimiters is not empty.
2. The second line of content inside the table delimiters is empty.

Example 253. First row is implicitly assigned header

```
|===
|Column 1, header row |Column 2, header row

|Cell in column 1, row 2
|Cell in column 2, row 2

|Cell in column 1, row 3
|Cell in column 2, row 3
```

```
|===
```

As seen in the result below, if all of these rules hold true, then the first row of the table is treated as a header row.

Result of Example 253

Column 1, header row	Column 2, header row
Cell in column 1, row 2	Cell in column 2, row 2
Cell in column 1, row 3	Cell in column 2, row 3

Deactivate the implicit assignment of header

To suppress the implicit behavior of promoting the first row to a header row, assign the value `noheader` to the `options` attribute using the formal (`options=noheader`) or shorthand (`%noheader`) syntax. In Example 254, `noheader` is assigned using the shorthand syntax.

Example 254. Deactivate implicit header row with noheader

```
[%noheader]
|===
|Cell in column 1, row 1 |Cell in column 2, row 1

|Cell in column 1, row 2 |Cell in column 2, row 2
|===
```

The table from Example 254 is displayed below.

Result of Example 254

Cell in column 1, row 1	Cell in column 2, row 1
Cell in column 1, row 2	Cell in column 2, row 2

Create a Footer Row

The last row of a table is promoted to a footer row if the `footer` value is assigned to the table's `options` attribute.

Assign footer to the last row

The footer row semantics and styles are applied to the last row in a table by assigning `footer` to the `options` attribute. The `options` attribute is set in the table's attribute list using the shorthand (`%value`) or formal syntax (`options="value"`).

The `options` attribute is represented by the percent sign (`%`) when it's set using the shorthand syntax. In Example 255, `footer` is assigned using the shorthand syntax for `options`.

Example 255. Table with footer assigned using the shorthand syntax

```
[%header%footer,cols="2,2,1"] ①
|===
|Column 1, header row
|Column 2, header row
|Column 3, header row

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2

|Column 1, footer row
|Column 2, footer row
|Column 3, footer row
|===
```

① Values assigned using the shorthand syntax must be entered before the `cols` attribute (or any other named attributes) in a table's attribute list, otherwise the processor will ignore them.

The table from Example 255 is displayed below.

Result of Example 255

Column 1, header row	Column 2, header row	Column 3, header row
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2
Column 1, footer row	Column 2, footer row	Column 3, footer row

In Example 256, the `options` attribute is set and assigned the `footer` value using the formal syntax. The `options` attribute accepts a comma-separated list of values.

Example 256. Table with footer assigned to the options attribute

```
[options="footer"]
|===
|Column 1, header row |Column 2, header row

|Cell in column 1, row 2
|Cell in column 2, row 2

|Cell in column 1, row 3
|Cell in column 2, row 3

|Column 1, footer row
|Column 2, footer row
|===
```

The last row of the table in Example 256 is rendered using the corresponding footer styles.

Result of Example 256

Column 1, header row	Column 2, header row
Cell in column 1, row 2	Cell in column 2, row 2
Cell in column 1, row 3	Cell in column 2, row 3
Column 1, footer row	Column 2, footer row

Align Content by Cell

The alignment operators are applied to a cell's specifier and allow you to horizontally and vertically align a cell's content.

Horizontal alignment operators

Content can be horizontally aligned to the left or right side of the cell as well as the center of the cell.

Flush left operator (<)

The less-than sign (<) aligns the content to the left side of the cell. This is the default horizontal alignment.

Flush right operator (>)

The greater-than sign (>) aligns the content to the right side of the cell.

Center operator (^)

The caret (^) centers the content horizontally in the cell.

A horizontal alignment operator is entered after a span or duplication operator (if present) and in front a vertical alignment operator (if present).

<factor><span or duplication operator><**horizontal alignment operator**><vertical alignment operator><style operator>|<cell's content>

Center content horizontally in a cell

To horizontally center the content in a cell, place the ^ operator in front of the cell's separator (|). Don't insert any spaces between the | and the operator.

Example 257. Center the content of a cell horizontally

```
|===
|Column Name |Column Name

^|This content is horizontally centered because the cell specifier includes the `+^+`
operator.
|There isn't a horizontal alignment operator on this cell specifier, so the cell falls
```

```
back to the default horizontal alignment.
Content is aligned to the left side of the cell by default.
|===
```

The table from Example 257 is rendered below.

Result of Example 257

Column Name	Column Name
This content is horizontally centered because the cell specifier includes the ^ operator.	There isn't a horizontal alignment operator on this cell specifier, so the cell falls back to the default horizontal alignment. Content is aligned to the left side of the cell by default.

If the cell specifier includes a span (`<n>`) or duplication (`<n>*`), place the `^+` directly after the span or duplication operator.

Example 258. Center content horizontally in spanned columns and duplicated cells

```
|===
|Column Name |Column Name

2+^|This cell spans two columns, and its content is horizontally centered because the
cell specifier includes the `+^+` operator.
2*^|This content is duplicated in two adjacent columns.
Its content is horizontally centered because the cell specifier
includes the `+^+` operator.
|===
```

The table from Example 258 is rendered below.

Result of Example 258

Column Name	Column Name
This cell spans two columns, and its content is horizontally centered because the cell specifier includes the ^ operator.	
This content is duplicated in two adjacent columns. Its content is horizontally centered because the cell specifier includes the ^ operator.	This content is duplicated in two adjacent columns. Its content is horizontally centered because the cell specifier includes the ^ operator.

Align the content of a cell to the right

To align the content in a cell to its right side, place the > operator in front of the cell's separator (|), but after a span (`<n>`) or duplication (`<n>*`+) operator (if present). Don't insert any spaces between the | and the operators.

Example 259. Right align the content of a cell

```
|===
|Column Name |Column Name

>|This content is aligned to the right side of the cell because the cell specifier
includes the `>` operator.
|There isn't a horizontal alignment operator on this cell specifier, so the cell falls
back to the default horizontal alignment.
Content is aligned to the left side of the cell by default.

2+>|This cell spans two columns.

Its content is aligned to the right because the cell specifier includes the `>`
operator.
The `>` operator must be placed directly after the span operator (`+`).
|===
```

The table from Example 259 is rendered below.

Result of Example 259

Column Name	Column Name
This content is aligned to the right side of the cell because the cell specifier includes the > operator.	There isn't a horizontal alignment operator on this cell specifier, so the cell falls back to the default horizontal alignment. Content is aligned to the left side of the cell by default.
	This cell spans two columns.
Its content is aligned to the right because the cell specifier includes the > operator. The > operator must be placed directly after the span operator (+).	

Vertical alignment operators

Content can be vertically aligned to the top or bottom of a cell as well as the center of a cell. Vertical alignment operators always begin with a dot (`.`).

Flush top operator (.<)

The dot and less-than sign (`.<`) aligns the content to the top of the cell. This is the default vertical alignment.

Flush bottom operator (.>)

The dot and greater-than sign (`.>`) aligns the content to the bottom of the cell.

Center operator (.^)

The dot and caret (`.^`) centers the content vertically.

A vertical alignment operator is entered after a horizontal alignment operator (if present) and in front of a style operator (if present).

<factor><span or duplication operator><horizontal alignment operator><**vertical alignment operator**><style operator>|<cell's content>

Align content to the bottom of a cell

To align the content to the bottom of a cell, place the `.>` operator in front of the cell's separator (`|`). Don't insert any spaces between the `|` and the operator.

Example 260. Align content to the bottom of a cell

```
[cols="2,1"]
|===
|Column Name |Column Name

.>|This content is aligned to the bottom of the cell because the cell specifier
includes the `.>` operator.
|There isn't a vertical alignment operator on this cell specifier, so the cell falls
back to the alignment assigned via the column specifier or the default vertical
alignment.
Content is aligned to the top of the cell by default.
|===
```

The table from Example 260 is rendered below.

Result of Example 260

Column Name	Column Name
This content is aligned to the bottom of the cell because the cell specifier includes the `.>` operator.	There isn't a vertical alignment operator on this cell specifier, so the cell falls back to the alignment assigned via the column specifier or the default vertical alignment. Content is aligned to the top of the cell by default.

If the cell specifier includes a span (`<n>`) or duplication (`<n>*`+), place the `.>` after the span or duplication operator.

Example 261. Align content to the bottom of a cell that spans rows

```
|===
|Column Name |Column Name

|There isn't a vertical alignment operator on this cell specifier, so the content is
aligned to the top of the cell by default.

.2+.>|This cell spans two rows, and its content is aligned to the bottom because the
cell specifier includes the `.>` operator.
```

```
|This content is aligned to the top of the cell by default.
|===
```

The table from Example 261 is rendered below.

Result of Example 261

Column Name	Column Name
There isn't a vertical alignment operator on this cell specifier, so the content is aligned to the top of the cell by default.	This cell spans two rows, and its content is aligned to the bottom because the cell specifier includes the `.>` operator.
This content is aligned to the top of the cell by default.	

Center content vertically in a cell

To vertically center the content in a cell, place the `.^` operator in front of the cell's separator (`|`). Don't insert any spaces between the `|` and the operator.

Example 262. Center the content of a cell vertically

```
|===
|Column Name |Column Name

.^|This content is vertically centered because the cell specifier includes the `+.^+`
operator.
|There isn't a vertical alignment operator on this cell specifier, so the cell falls
back to the default vertical alignment.
Content is aligned to the top of the cell by default.
|===
```

The table from Example 262 is rendered below.

Result of Example 262

Column Name	Column Name
This content is vertically centered because the cell specifier includes the `.^` operator.	There isn't a vertical alignment operator on this cell specifier, so the cell falls back to the default vertical alignment. Content is aligned to the top of the cell by default.

Apply horizontal and vertical alignment operators to the same cell

A cell can have a vertical and horizontal alignment operator included in its cell specifier. The horizontal operator always precedes the vertical operator.

Example 263. Align cells horizontally and vertically

```
|===
|Column 1 |Column 2 |Column 3

^.>|The specifier for this cell is `^.>`.
The content is centered horizontally and aligned to the bottom of the cell.
|There aren't any alignment operators on this cell's specifier, so the cell falls back
to the default alignments.
The default horizontal alignment is the left side of the cell.
The default vertical alignment is the top of the cell.
>.^|The specifier for this cell is `>.^`.
The content is aligned to the right side of the cell and centered vertically.

2.3+^.^|The specifier for this cell is `pass:[2.3+^.^]`.
It spans two columns and three rows.

Its content is centered horizontally and vertically.
3*.>|The specifier for this cell is `3*.>`.
The cell is duplicated in three consecutive rows in the same column.
It's content is aligned to the bottom of the cell.
|===
```

The table from Example 263 is rendered below.

Result Example 263

Column 1	Column 2	Column 3
The specifier for this cell is `^.>`. The content is centered horizontally and aligned to the bottom of the cell.	There aren't any alignment operators on this cell's specifier, so the cell falls back to the default alignments. The default horizontal alignment is the left side of the cell. The default vertical alignment is the top of the cell.	The specifier for this cell is `>.^`. The content is aligned to the right side of the cell and centered vertically.

Column 1	Column 2	Column 3
The specifier for this cell is `2.3+^.^`. It spans two columns and three rows. Its content is centered horizontally and vertically.		The specifier for this cell is `3*.>`. The cell is duplicated in three consecutive rows in the same column. It's content is aligned to the bottom of the cell.
		The specifier for this cell is `3*.>`. The cell is duplicated in three consecutive rows in the same column. It's content is aligned to the bottom of the cell.
		The specifier for this cell is `3*.>`. The cell is duplicated in three consecutive rows in the same column. It's content is aligned to the bottom of the cell.

Format Content by Cell

Cell styles and their operators

You can style all of the content in an individual cell by adding a style operator to the cell's specifier.

Style	Operator	Description
AsciiDoc	`a`	Supports block elements (lists, delimited blocks, and block macros). This style effectively creates a nested, standalone AsciiDoc document. The parent document's implicit attributes, such as `doctitle`, are shadowed and custom attributes are inherited.
Default	`d`	All of the markup that is permitted in a paragraph (i.e., inline formatting, inline macros) is supported.
Emphasis	`e`	Text is italicized.
Header	`h`	Applies the header semantics and styles to the text and cell borders.
Literal	`l`	Content is treated as if it were inside a literal block.

Style	Operator	Description
Monospace	`m`	Text is rendered using a monospace font.
Strong	`s`	Text is bold.

When a style operator isn't explicitly assigned to a cell specifier (or column specifier), the cell falls back to the default (`d`) style and is processed as regular paragraph text. (The explicit `d` style is only needed if you want to revert the cell style based to a normal (default) cell when a style is applied to the column.)

Apply a style to a table cell

The style operator is always entered last in a cell specifier. Don't insert any spaces between the `|` and the operator.

<factor><span or duplication operator><horizontal alignment operator><vertical alignment operator><**style operator**>|<cell's content>

Let's apply a style operator to each cell in Example 264.

Example 264. Apply a style operator to a cell

```
|===
|Column 1 |Column 2

2*>m|This content is duplicated across two columns (2*) and aligned to the right side
of the cell (>).

It's rendered using a monospace font (m).

.3+^.>s|This cell spans 3 rows (`3+`).
The content is centered horizontally (`+^+`), vertically aligned to the bottom of the
cell (`.>`), and styled as strong (`s`).
e|This content is italicized (`e`).

m|This content is rendered using a monospace font (m).

s|This content is bold (`s`).
|===
```

The table from Example 264 is rendered below.

Result of Example 264

Column 1	Column 2
This content is duplicated across two columns (2*) and aligned to the right side of the cell (>). It's rendered using a monospace font (m).	This content is duplicated across two columns (2*) and aligned to the right side of the cell (>). It's rendered using a monospace font (m).
This cell spans 3 rows (3+). The content is centered horizontally (^), vertically aligned to the bottom of the cell (.>), and styled as strong (s).	*This content is italicized (e).*
	This content is rendered using a monospace font (m).
	This content is bold (s).

Override the column style on a cell

When you assign a style operator to a cell, it overrides the column's style operator. In Example 265, the style operator assigned to the first column is overridden on two cells. The header row also overrides style operators. However, inline formatting markup is applied in addition to the style specified by an operator.

Example 265. Override the column style using a cell style operator

```
[cols="m,m"] ①
|===
|Column 1, header row |Column 2, header row ②

|This content is rendered using a monospace font because the column's specifier
includes the `m` operator.
|This content is rendered using a monospace font because the column's specifier
includes the `m` operator.

s|This content is rendered as bold paragraph text because the `s` operator in the
cell's specifier overrides the style operator in the column specifier. ③
|*This content is rendered using a monospace font because the column's specifier
includes the `m` operator.
It's also bold because it's marked up with the inline syntax for bold formatting.* ④

d|This content is rendered as regular paragraph text because the `d` operator in the
cell's specifier overrides the style operator in the column specifier. ⑤
|This content is rendered using a monospace font because the column's specifier
includes the `m` operator.
|===
```

① The monospace operator (m) is assigned to both columns.

② The header row ignores any style operators assigned via column or cell specifiers.

③ The strong operator (s) is assigned to this cell's specifier, overriding the column's monospace style.

④ Inline formatting is applied in addition to the style assigned via a column specifier.

⑤ The default operator (d) is assigned to this cell's specifier, resetting it to the default text style.

The table from Example 265 is displayed below.

Result of Example 265

Column 1, header row	Column 2, header row
This content is rendered using a monospace font because the column's specifier includes the m operator.	This content is rendered using a monospace font because the column's specifier includes the m operator.
This content is rendered as bold paragraph text because the s operator in the cell's specifier overrides the style operator in the column specifier.	**This content is rendered using a monospace font because the column's specifier includes the m operator. It's also bold because it's marked up with the inline syntax for bold formatting.**
This content is rendered as regular paragraph text because the d operator in the cell's specifier overrides the style operator in the column specifier.	This content is rendered using a monospace font because the column's specifier includes the m operator.

Use AsciiDoc block elements in a table cell

To use AsciiDoc block elements, such as delimited source blocks and lists, in a cell, place the a operator directly in front of the cell's separator (|). Don't insert any spaces between the | and the operator. The a can also be specified on the column in the cols attribute on the table.

Example 266. Apply the AsciiDoc block style operator to two cells

```
|===
|Normal Style |AsciiDoc Style

|This cell isn't prefixed with an `a`, so the processor doesn't 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 interprets the following lines
as an AsciiDoc list.

* List item 1
* List item 2
* List item 3

|This cell isn't prefixed with an `a`, so the processor doesn't interpret the listing
block delimiters or the `source` style.

[source,python]
----
import os
print ("%s" %(os.uname()))
----
```

```
a|This cell is prefixed with an `a`, so the listing block is processed and rendered
according to the `source` style rules.

[source,python]
----
import os
print "%s" %(os.uname())
----

|===
```

The table from Example 266 is rendered below.

Result of Example 266

Normal Style	AsciiDoc Style
This cell isn't prefixed with an `a`, so the processor doesn't interpret the following lines as an AsciiDoc list. * List item 1 * List item 2 * List item 3	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 isn't prefixed with an `a`, so the processor doesn't interpret the listing block delimiters or the `source` style. [source,python] ---- import os print ("%s" %(os.uname())) ----	This cell is prefixed with an `a`, so the listing block is processed and rendered according to the `source` style rules. `import os` `print "%s" %(os.uname())`

An AsciiDoc table cell effectively creates a nested document. As such, it inherits attributes from the parent document. If an attribute is set or explicitly unset in the parent document, it *cannot* be modified in the AsciiDoc table cell. There are a handful of exceptions to this rule, which includes doctype, toc, notitle (and its complement, showtitle), and compat-mode. Any newly defined attributes in the AsciiDoc table do not impact the attributes in the parent document. Instead, these attributes are scoped to the table cell.

If the AsciiDoc table cell starts with a preprocessor directive, that directive should be placed on the line after the cell separator. While it can be placed on the same line as the cell separator, that style is not recommended. That's because the preprocessor directive that starts after the cell separator must be treated with special handling and is thus limited to a single line (for example, a multiline preprocessor conditional is not allow in this case). By starting the contents of the AsciiDoc table cell on the line after the cell separator, the contents will be parsed as normal.

Span Columns and Rows

A table cell can span more than one column and row.

Span factor and operator

With a **span** a table cell can stretch across adjacent columns, rows, or a block of adjacent columns and rows. A span consists of a span factor and a span operator.

The **span factor** indicates the number columns, rows, or columns and rows a cell should span.

Column span factor

A single integer (`<n>`) that represents the number of consecutive columns a cell should span.

Row span factor

A single integer prefixed with a dot (`.<n>`) that represents the number of consecutive rows a cell should span.

Block span factor

Two integers (`<n>.<n>`) that represent a block of adjacent columns and rows a cell should span. The first integer, `<n>`, is the column span factor. The second integer, which is prefixed with a dot, `.<n>`, is the row span factor.

The **span operator** is a plus sign (`+`) placed directly after the span factor (`<n>.<n>+`). The span operator tells the converter to interpret the span factor as part of a span instead of a duplication.

A span is the first operator in a cell specifier.

<**span factor**><**span operator**><horizontal alignment operator><vertical alignment operator><style operator>|<cell's content>

Span multiple columns

To have a cell span consecutive columns, enter the column span factor and span operator (`<n>+`) in the cell specifier. Don't insert any spaces between the span, any alignment or style operators (if present), and the cell's separator (`|`).

Example 267. Span three columns with a cell

```
|===
|Column 1, header row |Column 2, header row |Column 3, header row |Column 4, header
row

3+|This cell spans columns 1, 2, and 3 because its specifier contains a span of `3+`
|Cell in column 4, row 2

|Cell in column 1, row 3
|Cell in column 2, row 3
|Cell in column 3, row 3
|Cell in column 4, row 3
|===
```

The table from Example 267 is displayed below.

Result of Example 267

Column 1, header row	Column 2, header row	Column 3, header row	Column 4, header row
This cell spans columns 1, 2, and 3 because its specifier contains a span of `3+`			Cell in column 4, row 2
Cell in column 1, row 3	Cell in column 2, row 3	Cell in column 3, row 3	Cell in column 4, row 3

Span multiple rows

To have a cell span consecutive rows, enter the row span factor and span operator (`.<n>+`) in the cell specifier. Remember to prefix the span factor with a dot (`.`). Don't insert any spaces between the span, any alignment or style operators (if present), and the cell's separator (`|`).

Example 268. Span two rows with a cell

```
|===
|Column 1, header row |Column 2, header row

.2+|This cell spans rows 2 and 3 because its specifier contains a span of `.2+`
|Cell in column 2, row 2

|Cell in column 2, row 3

|Cell in column 1, row 4
|Cell in column 2, row 4
|===
```

The table from Example 268 is displayed below.

Result of Example 268

Column 1, header row	Column 2, header row
This cell spans rows 2 and 3 because its specifier contains a span of `.2+`	Cell in column 2, row 2
	Cell in column 2, row 3
Cell in column 1, row 4	Cell in column 2, row 4

Span columns and rows

A single cell can span a block of adjacent columns and rows. Enter the column span factor (`<n>`), followed by the row span factor (`.<n>`), and then the span operator (`+`).

Example 269. Span two columns and three rows with a single cell

```
|===
|Column 1, header row |Column 2, header row |Column 3, header row |Column 4, header
row

|Cell in column 1, row 2
2.3+|This cell spans columns 2 and 3 and rows 2, 3, and 4 because its specifier
```

```
contains a span of `2.3+`
|Cell in column 4, row 2

|Cell in column 1, row 3
|Cell in column 4, row 3

|Cell in column 1, row 4
|Cell in column 4, row 4
|===
```

The table from Example 269 is displayed below.

Result of Example 269

Column 1, header row	Column 2, header row	Column 3, header row	Column 4, header row
Cell in column 1, row 2	This cell spans columns 2 and 3 and rows 2, 3, and 4 because its specifier contains a span of `2.3+`		Cell in column 4, row 2
Cell in column 1, row 3			Cell in column 4, row 3
Cell in column 1, row 4			Cell in column 4, row 4

Duplicate Cells

The contents of a cell can be duplicated in consecutive cells.

Duplication factor and operator

The duplication factor and operator are applied to a cell's specifier and allow you to clone a cell's content and properties across consecutive cells. A duplication is the first operator in a cell specifier.

<**duplication factor**><**duplication operator**><horizontal alignment operator><vertical alignment operator><style operator>|<cell's content>

The **duplication factor** is a single integer (`<n>`) that indicates how many times the cell's content should be duplicated.

The **duplication operator** is an asterisk (`*`) placed directly after the duplication factor (`<n>*`). The duplication operator tells the converter to interpret the duplication factor as part of a duplication instead of a span.

Duplicate a cell and its properties

To duplicate a cell, enter the duplication factor and duplication operator (`<n>*`) in the cell specifier. Don't insert any spaces between the duplication, any alignment or style operators (if present), and the cell's separator (`|`).

Example 270. Duplicate the contents of two cells

```
|===
|Column 1, header row |Column 2, header row |Column 3, header row
```

```
2*|This cell is duplicated in columns 1 and 2 because its specifier contains a
duplication of `2*`
|Cell in column 3, row 2

|Cell in column 1, row 3
|Cell in column 2, row 3
3*e|This cell specifier contains the duplication `3*` and style operator `e`.

The cell's text is italicized and duplicated in column 3, row 3 and columns 1 and 2 on
row 4.

|Cell in column 3, row 4
|===
```

The table from Example 270 is displayed below.

Result of Example 270

Column 1, header row	Column 2, header row	Column 3, header row
This cell is duplicated in columns 1 and 2 because its specifier contains a duplication of `2*`	This cell is duplicated in columns 1 and 2 because its specifier contains a duplication of `2*`	Cell in column 3, row 2
Cell in column 1, row 3	Cell in column 2, row 3	*This cell specifier contains the duplication `3*` and style operator `e`.* *The cell's text is italicized and duplicated in column 3, row 3 and columns 1 and 2 on row 4.*
This cell specifier contains the duplication `3*` and style operator `e`. *The cell's text is italicized and duplicated in column 3, row 3 and columns 1 and 2 on row 4.*	*This cell specifier contains the duplication `3*` and style operator `e`.* *The cell's text is italicized and duplicated in column 3, row 3 and columns 1 and 2 on row 4.*	Cell in column 3, row 4

Table Width

By default, a table will span the width of the content area.

Fixed width

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.

Example 271. Table with width set to 75%

```
[width=75%]
|===
|Column 1, header row |Column 2, header row |Column 3, header row

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2

|Cell in column 1, row 3
|Cell in column 2, row 3
|Cell in column 3, row 3
|===
```

Result of Example 271

Column 1, header row	Column 2, header row	Column 3, header row
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2
Cell in column 1, row 3	Cell in column 2, row 3	Cell in column 3, row 3

Autowidth

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.

Example 272. Table using autowidth

```
[%autowidth]
|===
|Column 1, header row |Column 2, header row |Column 3, header row

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2

|Cell in column 1, row 3
|Cell in column 2, row 3
|Cell in column 3, row 3
|===
```

Result of Example 272

Column 1, header row	Column 2, header row	Column 3, header row
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2
Cell in column 1, row 3	Cell in column 2, row 3	Cell in column 3, row 3

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. (Alternatively, you can set the `width` attribute to `100%`.)

Example 273. Full-width table with autowidth columns

```
[%autowidth.stretch]
|===
|Column 1, header row |Column 2, header row |Column 3, header row

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2

|Cell in column 1, row 3
|Cell in column 2, row 3
|Cell in column 3, row 3
|===
```

The table from Example 273 is rendered below. The columns are sized to the content, but the table spans the width of the page.

Result of Example 273

Column 1, header row	Column 2, header row	Column 3, header row
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2
Cell in column 1, row 3	Cell in column 2, row 3	Cell in column 3, row 3

The `autowidth` option is not recognized by the DocBook converter.

Mix fixed and autowidth columns

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).

Example 274. Table with fixed and autowidth columns

```
[cols="25h,~,~"]
|===
|small |as big as the column needs to be |the rest
|===
```

Result of Example 274

small	as big as the column needs to be	the rest

Table Borders

The borders on a table are controlled using the `frame` and `grid` block attributes. You can combine these two attributes to achieve a variety of border styles for your tables.

Frame

The border around a table is controlled using the `frame` attribute on the table. The frame attribute defaults to the value `all` (implied value), which draws a border on each side of the table. This default can be changed by setting the `table-frame` document attribute. You can override the default value by setting the frame attribute on the table to the value `all`, `ends`, `sides` or `none`.

The `ends` value draws a border on the top and bottom of the table.

Example 275. Table with frame=ends

```
[frame=ends]
|===
|Column 1, header row |Column 2, header row |Column 3, header row

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2

|Cell in column 1, row 3
|Cell in column 2, row 3
|Cell in column 3, row 3
|===
```

Result of Example 275

Column 1, header row	Column 2, header row	Column 3, header row
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2
Cell in column 1, row 3	Cell in column 2, row 3	Cell in column 3, row 3

The `sides` value draws a border on the right and left side of the table.

Example 276. Table with frame=sides

```
[frame=sides]
|===
|Column 1, header row |Column 2, header row |Column 3, header row

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2

|Cell in column 1, row 3
|Cell in column 2, row 3
|Cell in column 3, row 3
|===
```

Result of Example 276

Column 1, header row	Column 2, header row	Column 3, header row
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2
Cell in column 1, row 3	Cell in column 2, row 3	Cell in column 3, row 3

The `none` value removes the borders around the table.

Example 277. Table with frame=none

```
[frame=none]
|===
|Column 1, header row |Column 2, header row |Column 3, header row

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2

|Cell in column 1, row 3
|Cell in column 2, row 3
|Cell in column 3, row 3
|===
```

Result of Example 277

Column 1, header row	Column 2, header row	Column 3, header row
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2
Cell in column 1, row 3	Cell in column 2, row 3	Cell in column 3, row 3

Grid

The borders between the cells in a table are controlled using the `grid` attribute on the table. The grid attribute defaults to the value `all` (implied value), which draws a border between all cells. This default can be changed by setting the `table-grid` document attribute. You can override the default value by setting the grid attribute on the table to the value `all`, `rows`, `cols` or `none`.

The `rows` value draws a border between the rows of the table.

Example 278. Table with grid=rows

```
[grid=rows]
|===
|Column 1, header row |Column 2, header row |Column 3, header row

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2

|Cell in column 1, row 3
|Cell in column 2, row 3
```

```
|Cell in column 3, row 3
|===
```

Result of Example 278

Column 1, header row	Column 2, header row	Column 3, header row
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2
Cell in column 1, row 3	Cell in column 2, row 3	Cell in column 3, row 3

The `cols` value draws borders between the columns.

Example 279. Table with grid=cols

```
[grid=cols]
|===
|Column 1, header row |Column 2, header row |Column 3, header row

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2

|Cell in column 1, row 3
|Cell in column 2, row 3
|Cell in column 3, row 3
|===
```

Result of Example 279

Column 1, header row	Column 2, header row	Column 3, header row
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2
Cell in column 1, row 3	Cell in column 2, row 3	Cell in column 3, row 3

The `none` value removes all borders between the cells.

Example 280. Table with grid=none

```
[grid=none]
|===
|Column 1, header row |Column 2, header row |Column 3, header row

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2

|Cell in column 1, row 3
|Cell in column 2, row 3
|Cell in column 3, row 3
|===
```

Result of Example 280

Column 1, header row	Column 2, header row	Column 3, header row
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2
Cell in column 1, row 3	Cell in column 2, row 3	Cell in column 3, row 3

Interaction with row and column spans

Using row and column spans may interfere with the placement of borders on a table. This is a limitation of styling HTML using CSS.

When a cell extends into other rows or columns, that cell is not represented in the HTML for the rows or columns it extends into. This is a problem if the cell reaches the boundary of the table. The CSS selector only matches the cell where it starts and thus does not detect when it is touching the table boundary. It therefore cannot add or remove the border as it would for a 1x1 cell (i.e., a cell confined to a single row and column).

The interference with border placement caused by row and column spans does not always happen. Borders on a table with a rowspan or colspan that reaches the table boundary will always work correctly when the frame and grid are congruent. In this context, congruent means the frame and grid are contributing borders to the same edges.

Here are those scenarios in which the frame and grid are congruent:

- frame=all, grid=all
- frame=all, grid=none
- frame=all, grid=rows
- frame=all, grid=cols
- frame=ends, grid=rows
- frame=sides, grid=cols
- frame=none, grid=none

If you use row and column spans in a table, you are strongly encouraged to use one of these frame and grid combinations.

Table Striping

You can add zebra-striping to rows of a table. When this feature is enabled, the specified rows are shaded using a background color to create a zebra striping effect.

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 the necessary styles for table striping.

Striping attributes

Which rows are striped is controlled using the `stripes` attribute on the table. The stripes attribute defaults to the value `none` (implied value), which means rows are not striped by default. This default can be changed by setting the `table-stripes` document attribute. You can override the default value by setting the stripes attribute on the table.

The `stripes` attribute on a table and the `table-stripes` document attribute accept the following values:

- `none` - no rows are shaded (default)
- `even` - even rows are shaded
- `odd` - odd rows are shaded
- `all` - all rows are shaded
- `hover` - the row under the mouse cursor is shaded (HTML only)

stripes block attribute

In the following example, the stripes are enabled for even rows in the table body (the row that contains A2 and B2).

```
[cols=2*,stripes=even]
|===
|A1
|B1
|A2
|B2
|A3
|B3
|===
```

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=2*]
|===
|A1
|B1
|A2
|B2
|A3
|B3
|===
```

table-stripes attribute

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.

```
= Document Title
:table-stripes: even

[cols=2*]
|===
|A1
|B1
|A2
|B2
|A3
|B3
|===
```

Table Orientation

Landscape

A table can be displayed in landscape (rotated 90 degrees counterclockwise) using the `rotate` option (preferred).

```
[%rotate]
|===
|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 table can also be displayed in landscape using the `orientation` attribute.

```
[orientation=landscape]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|===
```

Currently, this is only implemented by the DocBook backend (it sets the attribute `orient="land"`).

Assign a Role to a Table

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:

```
[.name-of-role]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|===
```

Nesting 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

Col1	Col2
C11	C12

We recommend using nested tables sparingly. There's usually a better way to present the information.

CSV, TSV and DSV Data

Default table syntax

A table is delimited by a vertical bar and three equal signs (`|===`). It contains cells that are arranged into rows according to the number of columns the table is assigned. The number of columns a table contains can be specified implicitly using the number of cells in the table's first row or by setting the `cols` attribute. Each cell is specified by a vertical bar (`|`).

If you're new to AsciiDoc tables, Build a Basic Table provides step by step directions for creating your first table.

Style and layout options

Table content can be:

- styled and aligned by column or cell,
- aligned by row,
- duplicated across multiple rows, and
- marked up by any AsciiDoc syntax.

Table cells can span rows and columns.

You can adjust a table's:

- width,
- orientation, and
- border style.

You can also specify each column's width and designate header and footer rows.

Supported data formats

The default table data format is prefix-separated values (PSV); that means the processor creates a new cell each time it encounters a vertical bar (|). AsciiDoc also supports comma-separated values (CSV), tab-separated values (TSV), and delimited data values (DSV).

Escape 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 in place of | in content.
- 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 separa-

tor.

The cell separator is controlled using the `separator` attribute on the table block. You'll want to select any single character that is not found in the 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.

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:Indestructible:Dance
The Piano Guys:Code Name Vivaldi:Classical
|===
```

Result: Rendered DSV table

Artist	Track	Genre
Robyn	Indestructible	Dance
The Piano Guys	Code Name Vivaldi	Classical

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.
- Empty 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.
- 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 (`:`).
- Empty 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.

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):

Example 281. A DSV table with a custom separator

```
[format=dsv,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=dsv` and `separator=,`, the data will be processed using the DSV rules, even though the data looks like CSV.

Shorthand notation for data tables

AsciiDoc 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:Indestructible:Dance
:===
```

Result: Rendered DSV table using shorthand syntax

Artist	Track	Genre
Robyn	Indestructible	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.

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.

Table Reference

Attribute	Description	Value	Description	Notes
`caption`	defines the title label on a single table	user-defined		
`cols`	comma-separated list of column specifiers	specifiers	Specifies the number of columns and the distribution ratio and default formatting for each column. See Add Columns to a Table for details	

Attribute	Description	Value	Description	Notes
`format`	data format of the table's contents	`psv`	cells are delimited by `separator` (default `|`) (aka prefix-separated values)	
		`dsv`	cells are delimited by a colon (`:`) (aka delimiter-separated values)	
		`csv`	cells are delimited by a comma (`,`) (aka comma-separated values)	
		`tsv`	cells are delimited by a tab character (aka tab-separated values)	
`separator`	character used to separate cells	`|`	default for top-level tables	
		`!`	default for nested tables	
		user-defined	any single character or `\t` for tab (e.g., `¦` or `%`). *Ideally a character not found in the cell content.*	
`frame`	draws a border around the table	`all`	border on all sides (default)	
		`ends`	border on top and bottom ends	
		`none`	no borders	
		`sides`	border on left and right sides	
`grid`	draws boundary lines between rows and columns	`all`	draws boundary lines around each cell (default)	
		`cols`	draws boundary lines between columns	
		`rows`	draws boundary lines between rows	
		`none`	no boundary lines	

Attribute	Description	Value	Description	Notes
`stripes`	controls row shading (via background color)	`none`	no rows are shaded (default)	
		`even`	even rows are shaded	
		`odd`	odd rows are shaded	
		`hover`	row under the mouse cursor is shaded (HTML only)	
		`all`	all rows are shaded	
`align`	horizontally aligns a table with restricted width	`left`	aligns to left side of page (default)	Not recognized by Asciidoctor. To align the table, use an alignment role (e.g., `[role=center]` or `[.center]`). The alignment roles work for both HTML and PDF output. Alignment roles and the `float` attributes are mutually exclusive.
		`right`	aligns to right side of page	
		`center`	horizontally aligns to center of page	
`float`	floats the table to the specified side of the page	`left`	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 `width` attribute to take effect. The `float` and `align` attributes are mutually exclusive.
		`right`	floats the table to the right side of the page	
`halign`	horizontally aligns all of the cell contents in a table	`left`	aligns the contents of the cells to the left (default)	Not recognized by Asciidoctor. Define instead using column or cell specifiers (e.g., `3*>`), which take precedence over this value.
		`right`	aligns the contents of the cells to the right	
		`center`	aligns the contents to the cell centers	
`valign`	vertically aligns all of the cell contents in a table	`top`	aligns the cell contents to the top of the cell (default)	Not recognized by Asciidoctor. Define instead using column or cell specifiers (e.g., `3*.>`), which take precedence over this value.
		`bottom`	aligns the cell contents to the bottom of the cell	
		`middle`	aligns the cell contents to the middle of the cell	

Attribute	Description	Value	Description	Notes
orientation	rotates the table	landscape	rotated 90 degrees counterclockwise	Equivalent to setting the rotate option, which is preferred. DocBook only.
options	comma separated list of option names	header	promotes first row to the table header	header and footer rows are omitted by default
		footer	promotes last row to the table footer	
		breakable	allows the table to split across a page (default)	Mutually exclusive. DocBook only (specifically for generating PDF output).
		unbreakable	prevents the table from being split across a page	
		autowidth	disables explicit column widths (ignores distribution ratios in cols attribute)	
		rotate	Prints the table in landscape	Equivalent to setting the orientation to landscape. DocBook only.
role	comma-separated list of role names	left	floats the table to the left margin	The role is the preferred way to specify the alignment of a table with restricted width. May be specified using role shorthand (e.g., [.center]).
		right	floats the table to the right	
		center	aligns the table to center	
		stretch	stretches an autowidth table to the width of the page	
width	the table width relative to the available page width	user defined value	a percentage value between 0% and 100%	

Equations and Formulas (STEM)

If you need to include Science, Technology, Engineering and Math (STEM) expressions in your document, the AsciiDoc language supports embedding math-mode macros from LaTeX and/or AsciiMath notation as block or inline elements. These elements act as passthroughs to preserve the expressions as entered. The expressions are then passed on to the converter to be processed and rendered for display using a STEM provider (e.g., MathJax).

Activating STEM support

To activate equation and formula support, set the `stem` attribute in the document's header (or by passing the attribute to the command line or API).

Example 282. Setting the stem attribute

```
= My Diabolical Mathematical Opus
Jamie Moriarty
:stem: ①
```

① The default notation value, `asciimath`, is assigned implicitly.

By default, AsciiDoc's stem integration assumes all equations are AsciiMath if not specified explicitly. The HTML converter supports STEM content written in AsciiMath and TeX and LaTeX math notation. The DocBook converter only processes AsciiMath notation, leaving LaTeX to be processed by a separate tool in the DocBook toolchain.

If you want to use the LaTeX notation by default, assign `latexmath` to the stem attribute.

Example 283. Assigning an alternative notation to the stem attribute

```
= My Diabolical Mathematical Opus
Jamie Moriarty
:stem: latexmath
```

You can use both notations in the same document. The value of the `stem` attribute merely sets the default notation. To set the notation explicitly for a given block or inline span, just use `asciimath` or `latexmath` in place of `stem` as explained in Mixing STEM notations.

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.

Inline STEM content

The best way to mark up an inline formula is to use the `stem` macro. The text between the square brackets of the inline stem macro acts as an implicit passthrough.

Example 284. 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 expression within the square brackets (`[ ]`) of the macro.

The result of Example 284 is displayed below.

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.

Example 285. Inline stem macro with a right square bracket

```
A matrix can be written as stem:[[[a,b\],[c,d\]\]((n),(k))].
```

The result of Example 285 is displayed below.

A matrix can be written as [[a,b],[c,d]]((n),(k)).

Since the inline stem macro is an implicit passthrough, the closing square bracket it the only character you have to escape. You don't have to worry about escaping other AsciiDoc syntax, such as attribute references. All such syntax is passed through to the STEM processor as is.

Block STEM content

Block formulas are marked up by assigning the `stem` style to a delimited passthrough block.

Example 286. 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 Example 286 is rendered beautifully in the browser thanks to MathJax!

```
sqrt(4) = 2
```

You don't need to add special delimiters around the expression as the MathJax documentation suggests. The AsciiDoc processor handles that for you automatically!

Newlines in AsciiMath blocks

Newlines in an AsciiMath block are only preserved in certain circumstances. The following examples illustrate how newlines are handled.

▼ *Single newline not preserved*

```
x
y
```

▼ *Single newline preserved if escaped*

```
x\
y
```

▼ *Sequential newlines preserved if escaped*

```
x\
\
y
```

▼ *Paragraph break preserved*

```
x

y
```

▼ *Sequential newlines between paragraph break preserved*

```
x


y
```

The first preserved newline splits the expression into two. Subsequent newlines get translated into a `<br>` element.

Newlines in LaTeX blocks

Newlines in a LaTeX block are only preserved in certain circumstances. The following examples illustrate how newlines are handled.

▼ *Single newline not preserved*

```
x
y
```

▼ *Single newline preserved if escaped*

```
x\\
y
```

▼ *Sequential newlines preserved if escaped and prefixed by null character*

```
x\\
~\\
y
```

▼ *Paragraph break not preserved*

```
x

y
```

▼ *Paragraph break preserved if separated by newline spacer*

```
x
\\[1em]
y
```

The first preserved newline splits the expression into two. Subsequent newlines get translated into a `<br>` element.

Mixing STEM notations

You can use multiple notations for STEM content within the same document by using the notation's name instead of the keyword `stem`.

For example, if you want to write an inline equation using the LaTeX notation, name the macro `latexmath`.

Example 287. Inline latexmath macro syntax

```
latexmath:[C = \alpha + \beta Y^{\gamma} + \epsilon]
```

The result of Example 287 is displayed below.

```
C = \alpha + \beta Y^{\gamma} + \epsilon
```

The name that maps to the notation you want to use can also be applied to block STEM content.

Example 288. Using both asciimath and latexmath notations in a single document

```
= My Diabolical Mathematical Opus
Jamie Moriarty
:stem: latexmath

.An e-xciting limit with LaTeX!
[stem]
++++
\lim_{n \to \infty}\frac{n}{\sqrt[n]{n!}} = {\large e}
++++

.A basic square root with AsciiMath
[asciimath]
++++
sqrt(4) = 2
++++
```

Here's how the body of this example will be shown:

An e-xciting limit with LaTeX!

```
\lim_{n \to \infty}\frac{n}{\sqrt[n]{n!}} = {\large e}
```

A basic square root with AsciiMath

```
sqrt(4) = 2
```

Equation numbering

When writing expresions in LaTeX, you can configure the STEM processor to add numbers to block equations. To do so, set the `eqnums` attribute on the document to empty (or `AMS`). Let's look at an example.

```
= Document Title
:stem: latexmath
:eqnums:

[stem]
```

```
++++
\begin{equation}
x = y^2
\end{equation}
++++
```

The automatic numbering is only applied when the expression is contained within an `{equation}` container (in fact, to all such containers in a stem block).

If you want all expressions to be numbered, even those not designated as an equation, set the value of the `eqnums` attribute to `all`. Let's look at an example.

```
= Document Title
:stem: latexmath
:eqnums: all

[stem]
++++
x = y^2
++++
```

In this case, the `{equation}` container is not required.

Reference equations

When writing expresions in LaTeX, you can reference an equation in a stem block using an inline stem macro. First, you need to add a label (aka ID) to the (ideally numbered) equation in a stem block using `\label` .

```
= Document Title
:stem: latexmath
:eqnums:

[stem]
++++
\begin{equation}
\label{hypotenuse}
c = \sqrt{a^2 + b^2}
\end{equation}
++++
```

Next, we reference that equation using `\eqref`.

```
We can calculate the hypotenuse using stem:[\eqref{hypotenuse}].
```

You can also use `\eqref` within any LaTeX stem block.

Open Blocks

The most versatile block of all is the open block. It allows you to apply block attributes to a chunk of text without giving it any other semantics. In other words, it provides a generic structural container for enclosing content. The only notable limitation is that an open block cannot be nested inside of another open block.

Open block syntax

Example 289. Open block syntax

```
--
An open block can be an anonymous container,
or it can masquerade as any other block.
--
```

The result of Example 289 is displayed below.

An open block can be an anonymous container, or it can masquerade as any other block.

An open block can act as any other paragraph or delimited block, with the exception of *pass* and *table*. For instance, in Example 290 an open block is acting as a sidebar.

Example 290. Open block masquerading as a sidebar

```
[sidebar]
.Related information
--
This is aside text.

It is used to present information related to the main content.
--
```

The result of Example 290 is displayed below.

Related information

This is aside text.

It is used to present information related to the main content.

Example 291 is an open block acting as a source block.

Example 291. Open block masquerading as a source block

```
[source]
--
```

```
puts "I'm a source block!"
--
```

The result of Example 291 is displayed below.

```
puts "I'm a source block!"
```

Collapsible Blocks

You can designate block content to be revealed or hidden on interaction using the collapsible option and its companion, the open option. When the AsciiDoc source is converted to HTML, this block gets mapped to a disclosure summary element (i.e., a summary/details pair). If the output format does not support this interaction, it may be rendered as an unstyled block (akin to an open block).

Collapsible block syntax

You make block content collapsible by specifying the `collapsible` option on the example structural container. This option changes the block from an example block to a collapsible block.

Example 292. Collapsible block syntax

```
[%collapsible]
====
This content is only revealed when the user clicks the block title.
====
```

In the output, the content of this block is hidden until the reader clicks the default title, “Details”. The result of Example 292 is displayed below.

▼ *Details*

This content is only revealed when the user clicks the block title.

Like other blocks, the collapsible block recognizes the `id` and `role` attributes.

Collapsible paragraph syntax

If the content of the block is only a single paragraph, you can use the example paragraph style instead of the example structural container to make a collapsible paragraph.

Example 293. Collapsible paragraph syntax

```
[example%collapsible]
This content is only revealed when the user clicks the block title.
```

In the output, the content of this block is hidden until the reader clicks the default title, “Details”. The result of Example 293 is displayed below.

▼ *Details*

This content is only revealed when the user clicks the block title.

Customize the toggle text

If you want to customize the text that toggles the display of the collapsible content, specify a title on the block or paragraph.

Example 294. Collapsible block with custom title

```
.Click to reveal the answer
[%collapsible]
====
This is the answer.
====
```

The result of Example 294 is displayed below.

▼ *Click to reveal the answer*

This is the answer.

Notice that even though this block has a title, it's not numbered and does not have a caption prefix. That's because it's not an example block and thus does not get a numbered caption prefix like an example block would.

Default to open

If you want the collapsible block to be open by default, specify the `open` option as well.

Example 295. Collapsible block that defaults to open

```
.Too much detail? Click here.
[%collapsible%open]
====
This content is revealed by default.

If it's taking up too much space, the reader can hide it.
====
```

The result of Example 295 is displayed below.

▼ *Too much detail? Click here.*

This content is revealed by default.

If it's taking up too much space, the reader can hide it.

Use as an enclosure

Much like the open block, the collapsible block is an enclosure. If you want to make other types of blocks collapsible, such as an listing block, you can nest the block inside the collapsible block.

Example 296. Collapsible block that encloses a literal block

```
.Show stacktrace
[%collapsible]
====
```

```
....
Error: Content repository not found (url: https://git.example.org/repo.git)
    at transformGitCloneError
    at git.clone.then.then.catch
Caused by: HttpError: HTTP Error: 401 HTTP Basic: Access Denied
    at GitCredentialManagerStore.rejected
    at fill.then
....
====
```

The result of Example 296 is displayed below.

▼ *Show stacktrace*

```
Error: Content repository not found (url: https://git.example.org/repo.git)
    at transformGitCloneError
    at git.clone.then.then.catch
Caused by: HttpError: HTTP Error: 401 HTTP Basic: Access Denied
    at GitCredentialManagerStore.rejected
    at fill.then
```

Since the toggle text acts as the block title, you may decide to not put a title on the nested block, as in this example.

Comments

Like programming languages, AsciiDoc provides a way to add commentary to your document that's not carried over into the published document. This artifact is collectively known as a **comment**. Putting text in a comment is often referred to as "commenting it out".

A comment is often used to insert a writer-facing notation or to hide draft content that's not ready for publishing. In general, you can use comments anytime you want to hide lines from the processor. Comments can also be useful as a processor hint to keep adjacent blocks of content separate.

The AsciiDoc processor will ignore comments during parsing and, thus, will not include them in the parsed document. It will, however, account for the lines when mapping line numbers back to the source document.

AsciiDoc supports two styles of comments, line and block. A line comment is for making comments line-by-line (i.e., comment line). A block comment is for enclosing an arbitrary range of lines as a comment (i.e., comment block).

Comment lines

A comment line is any line outside of a verbatim block that begins with a double forward slash (`//`) that's not immediately followed by a third forward slash. Following this prefix, the line may contain an arbitrary number of characters. It's customary to insert a single space between the prefix and the comment text (e.g., `// line comment`).

Example 297. Line comment syntax

```
// A single-line comment.
```

When the processor encounters a line comment, it ignores the line and continues processing as though the line is not there. Line comments are processed as lines are read, so they can be used where paragraph text is not permitted, such as between attribute entries in the document header.

Line comments can be used strategically to separate blocks that otherwise have affinity, such as two adjacent lists.

Example 298. Line comment separating two lists

```
* first list

//

* second list
```

In this case, the single line comment effectively acts as an empty paragraph that's dropped from the parsed document. But before then, it will have served its purpose as a block boundary.

Comment blocks

A comment block is a specialized delimited block. It consists of an arbitrary number of lines bounded on either side by //// delimiter lines.

A comment block can be used anywhere a delimited block is normal accepted. The main difference is that once the block is read, it's dropped from the parsed document (effectively ignored). Additionally, no AsciiDoc syntax within the delimited lines is interpreted, not even preprocessor directives.

Example 299. Block comment syntax

```
////
A comment block.

Notice it's a delimited block.
////
```

A comment block can also be written as an open block with the comment style:

Example 300. Alternate block comment syntax

```
[comment]
--
A comment block.

Notice it's a delimited block.
--
```

A comment block that can consists of a single paragraph can be written as a paragraph with the comment style:

Example 301. Comment paragraph syntax

```
[comment]
A paragraph comment.
Like all paragraphs, the lines must be contiguous.

Not a comment.
```

If comment blocks are used in a list item, they must be attached to the list item just like any other block.

Example 302. Block comment attached to list item

```
* first item
+
////
A comment block in a list.
```

```
Notice it's attached to the preceding list item.
////

* second item
```

Within a table, a comment block can only be used in an AsciiDoc table cell.

Example 303. Block comment within a table

```
|===
a|
cell text

////
A comment block in a table.

Notice the cell has the "a" (AsciiDoc) style.
////
|===
```

Comment blocks can be very effective for commenting out sections of the document that are not ready for publishing or that provide background material or an outline for the text being drafted.

Automatic Table of Contents

A table of contents (TOC) is an index of section titles in an AsciiDoc document. When the TOC is enabled, the AsciiDoc processor automatically generates the TOC from the document's structure and inserts it into the output document. The number of levels (i.e., depth) of the TOC is configurable.

Activate the TOC

To enable the autogenerated TOC, set the `toc` document attribute. The `toc` attribute is activated with an attribute entry in the document header.

Example 304. Enable TOC with the toc attribute

```
= The Intrepid Chronicles
Kismet Lee; B. Steppenwolf; Pax Draeke
:toc: ①

== Certain Peril

Daylight trickles across the cobblestones...

=== A Recipe for Potion

We have to harvest the leaves by the light of the teal moons...

==== Searching for Ginseng

Crawling through the twisted understory...

== Dawn on the Plateau

Hanging from...
```

① Set the `toc` attribute in the header using an attribute entry. When the value of `toc` is empty, the processor will use the attribute's default value.

By default, the TOC is rendered directly below the document header, is titled *Table of Contents*, and contains section 1 and section 2 level titles only. The result of Example 304 is displayed below.

The Intrepid Chronicles

Kismet Lee · B. Steppenwolf · Pax Draeke

Table of Contents

Certain Peril

Daylight trickles across the cobblestones…

A Recipe for Potion

We have to harvest the leaves by the light of the teal moons…

Searching for Ginseng

Crawling through the twisted understory…

Dawn on the Plateau

Hanging from…

You can customize the title of the TOC, the depth of the section levels, and the position of the TOC in the document. However, not all of the attributes are supported by all converters. See TOC Attributes Reference for which attributes are available to each converter.

Activate the TOC from the CLI

The `toc` attribute can also be specified via the command line (`-a toc`).

TOC enabled via the CLI

```
$ asciidoctor -a toc my-document.adoc
```

Customize the TOC Title

You can change the title of the table of contents with the `toc-title` attribute.

Set toc-title

To generate a TOC with a custom title, set the `toc-title` attribute in the header and assign it your preferred title.

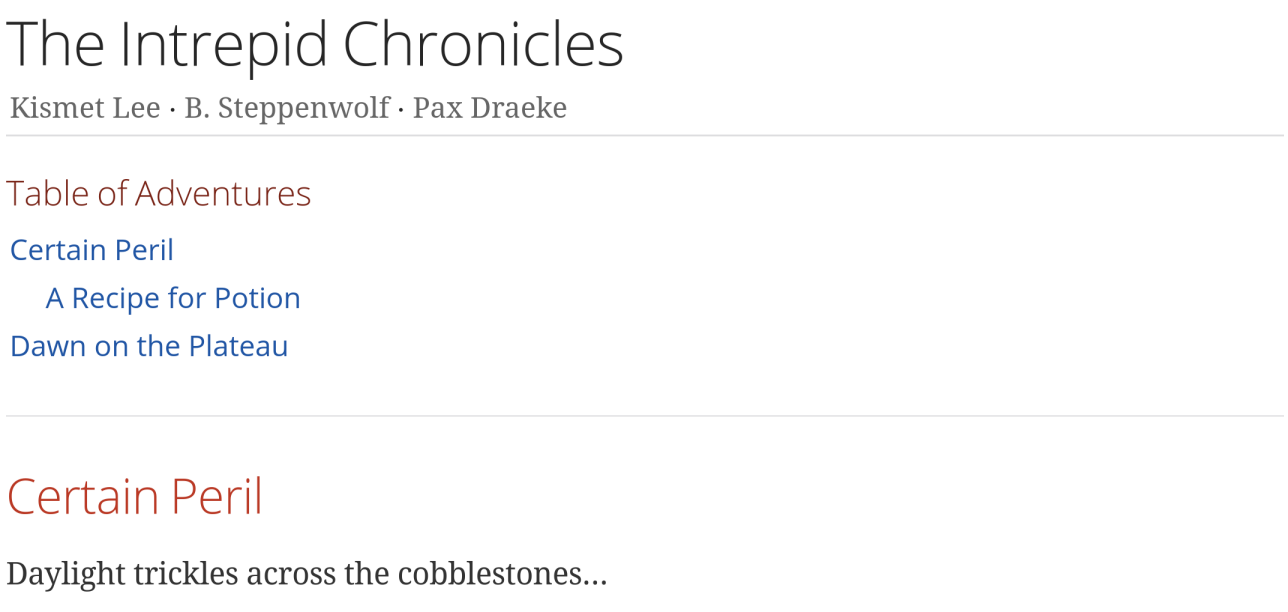
Example 305. Define a custom TOC title

```
= The Intrepid Chronicles
Kismet Lee; B. Steppenwolf; Pax Draeke
:toc: ①
:toc-title: Table of Adventures ②

== Certain Peril

Daylight trickles across the cobblestones...
```

① The `toc` attribute must be set in order to use `toc-title`.

② The `toc-title` is set and assigned the value `Table of Adventures` in the document's header.

The result of Example 305 is displayed below.



Adjust the TOC Depth

You can adjust the depth of section levels displayed in the table of contents (TOC) using the `toclevels` attribute.

Set toclevels

The `toclevels` document attribute controls the depth of the TOC. Accepted values are the integers 0 through 5. These values represent the section levels. (A section level is one less than the number of `=` signs the precede the title in the source.)

If the `toclevels` attribute is not specified, it defaults to `2`. That means the TOC displays level 1 (`==`) and level 2 (`===`) section titles and, in the case of a multipart book, level 0 (`=`) section titles (parts).

Let's use the `toclevels` attribute to increase the depth of the TOC from 2 to 4.

Example 306. Define toclevels value

```
= The Intrepid Chronicles
Kismet Lee; B. Steppenwolf; Pax Draeke
:toc: ①
:toclevels: 4 ②

== Certain Peril

Daylight trickles across the cobblestones...

=== A Recipe for Potion

We have to harvest the leaves by the light of the teal moons...

==== Searching for Ginseng

Crawling through the twisted understory...

== Dawn on the Plateau

Hanging from...
```

① The `toc` attribute must be set in order to use `toclevels`.

② `toclevels` is set and assigned the value `4` in the document header. The TOC will list the titles of any sections up to level 4 (i.e., `====`), when the document is rendered.

The result of Example 306 is displayed below.

The Intrepid Chronicles

Kismet Lee · B. Steppenwolf · Pax Draeke

Table of Contents

Certain Peril

Daylight trickles across the cobblestones…

A Recipe for Potion

We have to harvest the leaves by the light of the teal moons…

Searching for Ginseng

Crawling through the twisted understory…

Dawn on the Plateau

Hanging from…

In a multipart book, if you only want to see part titles (as well as any special sections at level 0) in the TOC, set `toclevels` to 0. If the document does not have parts, and you set `toclevels` to 0, the value is coerced to 1.

Position the TOC

By default, the table of contents is inserted directly below the document title, author, and revision lines when the `toc` attribute is set and its value is left empty or set to `auto`. This location can be changed by assigning one of the built-in positional values to the `toc` attribute. The values are:

- `left`
- `right`
- `preamble`
- `macro`
- `auto` (default)

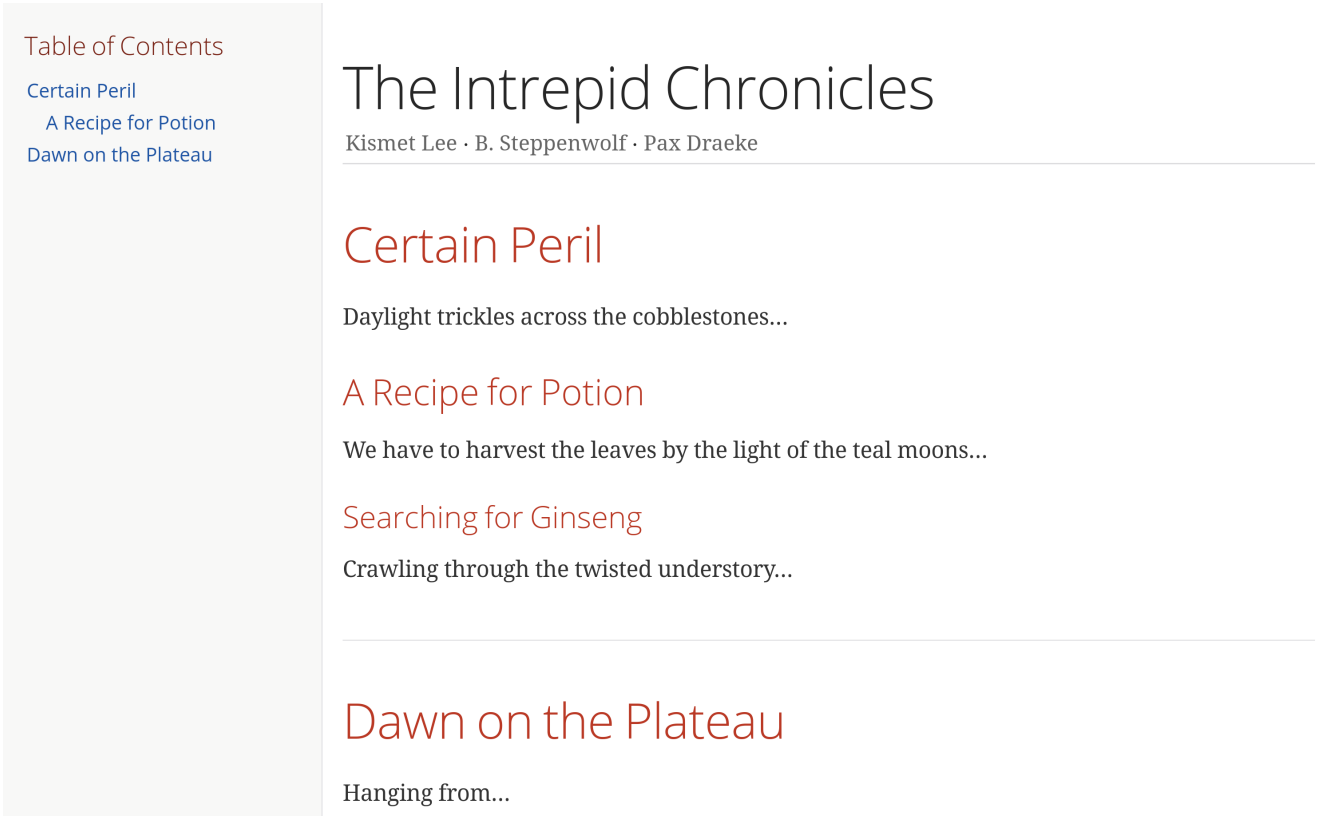
Display the TOC as a side column

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. The sidebar column containing the TOC is both fixed and scrollable.

Example 307. Assign the left value to toc

```
= The Intrepid Chronicles
Kismet Lee; B. Steppenwolf; Pax Draeke
:toc: left

== Certain Peril

Daylight trickles across the cobblestones...

=== A Recipe for Potion

We have to harvest the leaves by the light of the teal moons...

==== Searching for Ginseng

Crawling through the twisted understory...

== Dawn on the Plateau

Hanging from...
```

The result of Example 307 is displayed below.



This positioning is achieved using CSS and depends on support from the stylesheet.

The side positions (left and right) have a width requirement. These positions are only honored if there's sufficient room on the screen to fit the sidebar column (typically at least 768px). If sufficient room available is not available (i.e., the screen width falls below the breakpoint), the TOC **automatically shifts 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.

Display the TOC beneath the preamble

When `toc` is assigned the built-in value `preamble`, the TOC is placed immediately below the preamble.

Example 308. Assign the preamble value to toc

```
= The Intrepid Chronicles
Kismet Lee; B. Steppenwolf; Pax Draeke
:toc: preamble

This adventure begins on a frigid morning.
We've run out of coffee beans, but leaving our office means venturing into certain
peril.

== Certain Peril

Daylight trickles across the cobblestones...

== Dawn on the Plateau

Hanging from...
```

The result of Example 308 is displayed below.

The Intrepid Chronicles

Kismet Lee · B. Steppenwolf · Pax Draeke

This adventure begins on a frigid morning. We've run out of coffee beans, but leaving our office means venturing into certain peril.

Certain Peril

Daylight trickles across the cobblestones...

When using the `preamble` value, the TOC will not appear if your document does not have a preamble. To fix this problem, set the `toc` attribute to an empty value (i.e., leave the value empty) or assign it the value `auto`.

Use the TOC macro to position the TOC

To place the TOC in specific location in the document, assign the `macro` value to the `toc` attribute. Then, enter the table of contents block macro (i.e., TOC macro) on the line in your document where you want the TOC to appear. The TOC macro should only be used once in a document.

If the `toc` document attribute isn't assigned the value `macro`, any TOC macro in the document will be ignored.

Example 309. Assign the macro value to toc

```
= The Intrepid Chronicles
Kismet Lee; B. Steppenwolf; Pax Draeke
:toc: macro ①

== Certain Peril

toc::[] ②

Daylight trickles across the cobblestones...

== Dawn on the Plateau

Hanging from...
```

① The `toc` attribute must be set to `macro` 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.

The result of Example 309 is displayed below.

The Intrepid Chronicles

Kismet Lee · B. Steppenwolf · Pax Draeke

Certain Peril

Table of Contents

Daylight trickles across the cobblestones…

Dawn on the Plateau

Hanging from…

Embeddable HTML, editor and previewer limitations

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`
- `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.

TOC Attributes Reference

Attribute	Values	Example Syntax	Notes	Backends
`toc`	`auto`, `left`, `right`, `macro`, `preamble`	`:toc: left`	Not set by default. Defaults to `auto` if value is unspecified.	html
	`auto`, `macro`, `preamble`	`:toc: macro`	Not set by default. Defaults to `auto` if value is unspecified.	html (embeddable)
	`auto`	`:toc:`	Not set by default. When the title page is enabled in PDF output, the table of contents is placed directly after the title page.	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`	0–5	`:toclevels: 4`	Default value is `2`.	html, pdf
`toc-title`	user-defined	`:toc-title: Contents`	Default value is *Table of Contents*.	html, pdf
`toc-class`	valid CSS class name	`:toc-class: floating-toc`	Default value is *toc*.	html

Docinfo Files

Docinfo is a feature of AsciiDoc that allows you to insert custom content into the head, header, or footer of the output document. This custom content is read from files known as docinfo files by the converter. Docinfo files are intended as convenient way to supplement the output produced by a converter. Examples include injecting auxiliary metadata, stylesheets, and scripting logic not already provided by the converter.

The docinfo feature does not apply to all backends. While it works when converting to output formats such as HTML and DocBook, it does not work when converting to PDF using Asciidoctor PDF.

The docinfo feature must be explicitly enabled using the `docinfo` attribute (see Enabling docinfo). Which docinfo files are consumed depends on the value of the `docinfo` attribute as well as the backend.

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 recommended as it's already emitted by the converter.

You do not need to include the enclosing `<head>` element as it's assumed to be the envelope.

Here's an example:

Example 310. 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.

Example 311. 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, 2021. 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.

You can find a real-world example of a docinfo file for DocBook in the source of the Clojure Cookbook.

Header docinfo files

Header docinfo files are differentiated from head docinfo files by the addition of `-header` to the file name. In the HTML output, the header content is inserted immediately before the header div (i.e., `<div id="header">`). In the DocBook output, the header content is inserted immediately after the

opening tag (e.g., <article> or <book>).

One possible use of the header docinfo file is to completely replace the default header in the standard stylesheet. Just set the attribute noheader, then apply a custom header docinfo file.

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.,). 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.

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 <head>/<info> for <docname>.adoc files.	<docname>-docinfo<outfilesuffix>
	Header	Adds content to start of document for <docname>.adoc files.	<docname>-docinfo-header<outfilesuffix>
	Footer	Adds content to end of document for <docname>.adoc files.	<docname>-docinfo-footer<outfilesuffix>
Shared	Head	Adds content to <head>/<info> for any document in same directory.	docinfo<outfilesuffix>
	Header	Adds content to start of document for any document in same directory.	docinfo-header<outfilesuffix>
	Footer	Adds content to end of document for any document in same directory.	docinfo-footer<outfilesuffix>

Enabling docinfo

To specify which file(s) you want to apply, set the `docinfo` attribute to any combination of these values:

- `private-head`
- `private-header`
- `private-footer`
- `private` (alias for `private-head,private-header,private-footer`)
- `shared-head`
- `shared-header`
- `shared-footer`
- `shared` (alias for `shared-head,shared-header,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, header and footer files, if they exist, as well as the private footer file, if it exists.

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.

Locating docinfo files

By default, docinfo files are searched for in the same directory as the document file (which can be

overridden by setting the `:base_dir` API option / `--base-dir` CLI option). If you want to load them from another location, set the `docinfodir` attribute to the directory where the files are located. If the value of the `docinfodir` attribute is a relative path, that value is appended to the document directory. If the value is an absolute path, that value is used as is.

```
:docinfodir: common/meta
```

Note that if you use this attribute, only the specified folder will be searched; docinfo files in the document directory will no longer be found.

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. If this attribute is not set, it has an implied default value of `attributes` (i.e., attribute references are resolved).

For example, if you created the following docinfo file:

Example 312. Docinfo file (docinfo.xml) containing a revnumber attribute reference

```
<edition>{revnumber}</edition>
```

And this source document:

Example 313. Source document including a revision number

```
= Document Title
Author Name
v1.0, 2020-12-30
:doctype: book
:backend: docbook
:docinfo: shared
```

Then the converted DocBook output would be:

Example 314. Converted DocBook containing the docinfo

```
<?xml version="1.0" encoding="UTF-8"?>
<book xmlns="http://docbook.org/ns/docbook" xmlns:xl="http://www.w3.org/1999/xlink"
version="5.0" xml:lang="en">
<info>
<title>Document Title</title>
<date>2020-12-30</date>
<author>
<personname>
<firstname>Author</firstname>
<surname>Name</surname>
</personname>
```

```
</author>
<authorinitials>AN</authorinitials>
<revhistory>
<revision>
<revnumber>1.0</revnumber>
<date>2020-12-30</date>
<authorinitials>AN</authorinitials>
</revision>
</revhistory>
<edition>1.0</edition> ①
</info>
</book>
```

① The revnumber attribute reference in `docinfo.xml` was replaced by the source document's revision number in the converted output.

Another example is if you want to define the license link tag in the HTML head.

Example 315. Docinfo file (docinfo.html) containing a license meta tag

```
<link rel="license" href="{license-url}" title="{license-title}">
```

Now define these attributes in your AsciiDoc source:

Example 316. Source document that defines license attributes

```
= Document Title
:license-url: https://mit-license.org
:license-title: MIT
:docinfo: shared
```

Then the `<head>` tag in the converted HTML would include:

Example 317. Rendered license link tag in HTML output

```
<link rel="license" href="https://mit-license.org" title="MIT">
```

Includes

You can include content from another file into the current AsciiDoc document using the include directive. The included content can be AsciiDoc or it can be any other text format. Where that content is included in the document determines how it will be processed.

What is an include directive?

An **include directive** imports content from a separate file or URL into the content of the current document. When the current document is processed, the include directive syntax is replaced by the contents of the include file. Think of the include directive like a file expander. The include directive is a preprocessor directive, which means it has no awareness of the surrounding context.

When is an include directive useful?

The include directive is useful when you want to:

- Partition a large document into smaller files for better organization and to make restructuring simpler.[1]
- Insert source code from the external files where the code is maintained.
- Populate tables with output, such as CSV data, from other programs.
- Create document variants by combining the include directive with conditional preprocessor directives.
- Reuse content snippets and boilerplate content, such as term definitions, disclaimers, etc., multiple times within the same document.
- Define a common set of attributes across multiple documents (typically included into the document header).

Include directive syntax

An include directive must be placed on a line by itself with the following syntax:

```
include::target[leveloffset=offset,lines=ranges,tag(s)=name(s),indent=depth,encoding=encoding,opts=optional]
```

The target is required. The target may be an absolute path, a path relative to the current document, or a URL. Since the include directive is a line-oriented expression, the target may contain space characters. However, the target must not start with a space character (since that would turn it into a description list term). An absolute or relative path outside the directory of the outermost document will only be honored if the safe mode is unsafe. A URL target will only be resolved if the security settings on the processor allows it (e.g., `allow-uri-read`). See Include Content by URI.

The leveloffset, lines, tag(s), indent, encoding, and opts attributes are optional, thus reducing the simplest case to the following:

```
include::partial.adoc[]
```

Specifying the encoding is essential if the include file is not encoded in UTF-8. The value of this attribute must be an encoding recognized by Ruby (e.g., utf-8, iso-8859-1, windows-1252, etc), case insenstive. If the include file is already encoded in UTF-8 (or contains a BOM), this attribute is unnecessary.

When using consecutive include directives, you should always separate them by an empty line unless your intention is to adjoin the content in the include files so it becomes contiguous.

For example, if you're using the include directive to include individual chapters, the include directives should be offset from each other by an empty line. This strategy avoids relying on empty lines imported from the include file to keep the chapters separated. That separation should be encoded in the parent document instead.

```
include::chapter01.adoc[]

include::chapter02.adoc[]

include::chapter03.adoc[]
```

On the other hand, if you're using the include directive to lay down contiguous lines, such as common document attribute entries, then you would put the include directives on adjacent lines to avoid inserting empty lines.

```
= Document Title
Author Name
include::attributes-settings.adoc[]
include::attributes-urls.adoc[]
:url-example: https://example.org

Document body.
```

In either case, don't rely on the empty lines at the boundaries of the include file. And mind where empty lines are used in that include file.

Include processing

Although the include directive looks like a block macro, **it's not a macro and therefore isn't processed like one**. It's a preprocessor directive; it's important to understand the distinction.

A **preprocessor directive** is processed when the lines of a document 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 after the lines from the target of the include directive are added to the current document does the parser read and interpret those lines.

The include directive is disabled when Asciidoctor is run in secure mode. In secure mode, the include directive is converted to a link in the output document. See Safe Modes to learn more.

Escaping an include directive

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.

Include file resolution

The path used in an include directive can be relative or absolute.

If the path is relative, the processor resolves the path using the following rules:

- If the include directive is used in the primary (top-level) document, relative paths are resolved relative to the base directory. (The base directory defaults to the directory of the primary 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 make 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'll get the following warning message during conversion:

```
asciidoctor: WARNING: my-document.adoc: line 3: include file not found:
/.../content.adoc
```

The following message will also be inserted into the output:

```
Unresolved directive in my-document.adoc - include::content.adoc[]
```

To fix the problem, edit the file path and run the converter again. If you don't want the AsciiDoc processor to emit a warning, but rather drop the include that cannot be found, add the `opts=optional` attribute to the include directive.

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.

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 goes through some form of normalization.

The content of each include file is encoded to UTF-8. If the encoding attribute is specified on the include directive, the content is reencoded from that encoding to UTF-8. If the encoding attribute is not specified, the processor will look for the presence of a BOM and reencode the content from that encoding to UTF-8 accordingly. If neither of those conditions are met, the encoding is forced to UTF-8.

If the file is recognized as an AsciiDoc file (i.e., it has one of the following extensions: `.asciidoc`, `.adoc`, `.ad`, `.asc`, or `.txt`) additional normalization and processing is performed. First, all trailing whitespace and endlines are removed from each line and replaced with a Unix line feed. This normalization is important to how an AsciiDoc processor works. Next, the AsciiDoc processor runs the preprocessor on the lines, looking for and interpreting the following directives:

- includes
- preprocessor conditionals (e.g., `ifdef`)

Running the preprocessor on the included content allows includes to be nested, thus provides lot of flexibility in constructing radically different documents with a single primary 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).

Offset Section Levels

When your document gets large, you can split it up into subdocuments for easier editing.

```
= My book

include::chapter01.adoc[]

include::chapter02.adoc[]

include::chapter03.adoc[]
```

Note the empty lines before and after the include directives. This practice is recommended whenever including AsciiDoc content to avoid unexpected results (e.g., a section title getting interpreted as a line at the end of a previous paragraph).

Manipulate heading levels with leveloffset

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 primary 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 level offset 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.

Indent Included Content

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 indent attribute

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:

```
[source,ruby,indent=2]
----
    def names
      @name.split ' '
    end
----
```

Produces:

```
  def names
    @name.split ' '
  end
```

Use an Include File Multiple Times

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 primary 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).

Include List Item Content

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 `empty` 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 `empty` attribute and the include directive to define a list item, then include the primary text from another file:

```
* {empty}
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 compound content from another file into a list item:

```
* {empty}
+
--
include::complex-list-item.adoc[]
```

```
--
```

See dropping the principal text of a list item for another example of this technique.

Include Content by Tagged Regions

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 (per attribute rules). You can eliminate the need for quotes by using the semi-colon as the data separator instead.

Tagging 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 tag/end directives to include using the `tags` attribute.

In the include file, the tag directives (e.g., `tag::name[]` and `end::name[]`) must follow a word boundary and precede a space character or the end of line. The tag name must not be empty and must consist exclusively of non-space characters.

Typically, the tag directives will be placed after a line comment as defined by the language of the source file. This is especially important when using tags to include portions of an AsciiDoc document which may itself be converted. If the tag is not behind a line comment, the tag will be treated as regular content, which could disrupt the block structure, such as a table.

For languages that only support circumfix comments, such as XML, you can enclose the tag directives between the circumfix comment markers, offset by a space on either side. For example, in XML files, you can use `<!-- tag::name[] -->` and `<!-- end::name[] -->`.

Including by tag includes all regions marked with that tag. This makes it possible to include a group of lines from different regions of the document using a single tag.

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. The tag directives are preceded by a hash (`#`) because that's the start of a line comment in Ruby.

Example 318. 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 name to the `tag` directive. In this example, the tag is named *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 of a tagged snippet named *parse*.

⑥ This is the end of the tagged snippet named *parse*.

In the next example, the tagged region named *parse* is selected by the `include` directive.

Example 319. 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.

Example 320. Selecting the timings and the parse code snippets from a document

```
[source,ruby]
----
include::core.rb[tags=timings;parse]
----
```

It's also possible to have fine-grained tagged regions inside larger tagged regions.

In the next example, tagged regions are defined behind line comments. By putting each tag behind a line comment, regardless of how the content is included, you don't have to worry about those lines appearing in the rendered document.

```
// tag::snippets[]
// tag::snippet-a[]
```

```
snippet a
// end::snippet-a[]

// tag::snippet-b[]
snippet b
// end::snippet-b[]
// end::snippets[]
```

Let's assume 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
```

You could also include the whole file without worry that the tags will be rendered:

```
include::file-with-snippets.adoc[]
```

Now let's consider the case when the tags are not placed behind line comments. In this case, you need to ensure that tag filtering is being used or else those tags will be visible in the rendered document.

```
text a

tag::snippet-b[]
snippet b
end::snippet-b[]

text c
```

If you only want to include a specific tagged region of the file, use the following include directive:

```
include::file-with-snippets.adoc[tag=snippet-b]
```

The following lines will be selected and displayed:

```
snippet b
```

If you want to include the whole file, but also filter out any include tags, use the following include

directive:

```
include::file-with-snippets.adoc[tag=**]
```

The following lines will be selected and displayed:

```
text a
snippet b
text c
```

If you did not specify tag filtering, tag directives that aren't behind a line comment (e.g., `tag::snippet-b[]`) will also be printed too. Tag filtering is explained in more detail in the next section.

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. For example, here's how to include all lines that are not enclosed in a tag:

```
include::file-with-snippets.adoc[tag=!*]
```

When tag filtering is used, lines that contain a tag directive *are always discarded* (like a line comment). Even if you're not including content by tags, you can specify the double wildcard (`**`) to filter out all lines in the include file that contain a tag directive.

The modifiers you can use for filtering are as follows:

`*`

: The single wildcard. Select all tagged regions. May only be specified once, negated or not.

`**`

: The double wildcard. Select all the lines in the document **except for lines that contain a tag directive**. Use this symbol if you want to include a file that has tag directives, but you want to discard the lines that contain a tag directive. May only be specified once, negated or not.

`!`

: Negate the wildcard or tag.

The double wildcard is always applied first, regardless of where it appears in the list. If the double wildcard is not negated (i.e., `**`), it should only be combined with exclusions (e.g., `**;!foo`). A negated double wildcard (i.e., `!**`), which selects no lines, is usually implied as the starting point. A negated single wildcard has different meaning depending on whether it comes before tag names (e.g., `!*;foo`) or after at least one tag name (e.g., `foo;!*`).

Let's assume we have a region tagged `foo` with a nested region tagged `bar`. Here are some of the permutations you can use (along with their implied long-hand forms):

`**`

Selects all the lines in the document (except for lines that contain a tag directive). *(implies `**;*`)*

`*`

Selects all tagged regions in the document. Does not select lines outside of tagged regions. *(implies `!**;*`)*

`!*`

Selects only the regions in the document outside of tags (i.e., non-tagged regions). *(implies `**;!*`)*

`foo`

Selects only regions tagged *foo*. *(implies `!**;foo`)*

`foo;!bar`

Selects only regions tagged *foo*, but excludes any nested regions tagged *bar*. *(implies `!**;foo;!bar`)*

`foo;!*`

Selects only regions tagged *foo*, but excludes any nested tagged regions. *(implies `!**;foo;!*`)*

`*;!foo`

Selects all tagged regions, but excludes any regions tagged *foo* (nested or otherwise). *(implies `!**;*;!foo`)*

`!foo`

Selects all the lines in the document except for regions tagged *foo*. *(implies `**;!foo`)*

`!foo;!bar`

Selects all the lines in the document except for regions tagged *foo* or *bar*. *(implies `**;!foo;!bar`)*

`!*;foo`

Selects the regions in the document outside of tags (i.e., non-tagged regions) and inside regions tagged *foo*, excluding any nested tagged regions. To include nested tagged regions, they each must be named explicitly. *(implies `**;!*;foo`)*

If the filter begins with a negated tag or single wildcard, it implies that the pattern begins with `**`. An exclusion not preceded by an inclusion implicitly starts by selecting all the lines that do not contain a tag directive. Otherwise, it implies that the pattern begins with `!**`. A leading inclusion implicitly starts by selecting no lines.

Include Content by Line Ranges

The include directive supports selecting portions of the document to include. Using the `lines` attribute, you can include ranges of line numbers.

When including multiple line ranges, each entry in the list must be separated by either a comma or a semicolon. If commas are used, the entire value must be enclosed in quotes. Using the semicolon as the data separator eliminates this requirement.

Specifying 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 semicolons.

```
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]
```

Alternately, you can leave the end range unspecified and it will default to -1.

```
include::filename.txt[lines=12..]
```

Include Content by URI

Reference include content by URI

The include directive recognizes when the target is a URI and can include the content referenced by that URI. This example demonstrates how to include an AsciiDoc file from a GitHub repository directly into your document.

```
include::https://raw.githubusercontent.com/asciidoctor/asciidoctor/main/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 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 `UNSAFE` 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.

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 the to HTTP caching recommendations.

[1] Always separate consecutive include directives by an empty line unless your intent is to adjoin the content in the include files so it becomes contiguous.

Conditionals

You can include or exclude lines of text in your document using the following conditional preprocessor directives:

- ifdef
- ifndef
- ifeval

When the processor encounters one of these conditionals, it evaluates the specified condition. The condition is based on the presence or value of one or more document attributes. If the condition evaluates to true, the lines the conditional encloses are included. Otherwise, the lines are skipped.

Conditional processing

Although a conditional preprocessor directive looks like a block macro, **it's not a macro and therefore isn't processed like one**. It's a preprocessor directive; it's important to understand the distinction.

A **preprocessor directive** is processed when the lines of a document 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.

Escape a conditional directive

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.

ifdef and ifndef Directives

ifdef directive

Content between the `ifdef` and `endif` directives gets included if the specified attribute is set:

Example 321. 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.

Example 322. 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[]
```

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.

Example 323. 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::[]
```

ifndef directive

`ifndef` is the logical opposite of `ifdef`. Content between `ifndef` and `endif` gets included only if the specified attribute is *not* set:

Example 324. 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`.

Checking multiple attributes

Both the `ifdef` and `ifndef` directives accept multiple attribute names. The combinator can be “and” or “or”. The two combinators cannot be combined in the same expression.

ifdef with multiple attributes

If any attribute is set (or)

Multiple attribute names must be separated by commas (`,`). If one or more of the attributes are set, the content is included. Otherwise, the content is not included.

Example 325. If any attribute example

```
ifdef::backend-html5,backend-docbook5[Only shown if converting to HTML (backend-
html5 is set) or DocBook (backend-docbook5 is set).]
```

If all attributes are set (and)

Multiple attribute names must be separated by pluses (`+`). If all the attributes are set, the content is included. Otherwise, the content is not included.

Example 326. If all attributes example

```
ifdef::backend-html5+env-github[Only shown when converting to HTML (backend-html5
is set) on GitHub (env-github is set).]
```

ifndef with multiple attributes

The `ifndef` directive negates the results of the expression. When using the `ifndef` directive, the expression should be read with the prefix “unless”.

Unless any attribute is set (or)

Multiple attribute names must be separated by commas (`,`). If one or more of the attributes are set, the content is not included. Otherwise, the content is included.

Example 327. Unless any attribute example

```
ifndef::profile-production,env-site[Not shown if profile-production or env-site is
set.]
```

Unless all attributes are set (and)

Multiple attribute names must be separated by pluses (`+`). If all of the attributes are set, the content is not included. Otherwise, the content is included.

Example 328. Unless all attributes example

```
ifndef::profile-staging+env-site[Not shown if profile-staging and env-site are
set.]
```

ifeval Directive

Lines enclosed by an `ifeval` directive (i.e., between the `ifeval` and `endif` directives) are included if the expression inside the square brackets of the `ifeval` directive evaluates to true.

Example 329. 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`.

Unlike `ifdef` and `ifndef`, you cannot terminate a specific `ifeval` directive using its complement. For example, the following `ifeval` block is not valid:

Example 330. Invalid ifeval terminator

```
ifeval::[<condition>]
conditional content
endif::[<condition>]
```

You can only terminate the previous `ifeval` directive using an anonymous `endif::[]` directive, as shown here:

Example 331. Valid ineval terminator

```
ifeval::[<condition>]
conditional content
endif::[]
```

If you're mixing `ifeval` directives with `ifdef` or `ifndef` directives, you should always close multiline `ifdef` and `ifndef` directives by name (`endif::name-of-attribute[]`) so the `ifeval` directive does not end prematurely.

Anatomy

The expression of an `ifeval` directive consists of a left-hand value and a right-hand value with an operator in between. It's customary to include a single space on either side of the operator.

Example 332. 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}" == "main.html"]
...
endif::[]
```

Values

Each expression value can reference the name of zero or more AsciiDoc attributes using the attribute reference syntax (for example, {backend}).

Attribute references are resolved (i.e., substituted) first. Once attributes references have been resolved, each value is coerced to a recognized type.

When you expect the attribute reference to resolve to a string, that is, a sequence of characters, enclose that side of the expression in quotes. For example:

Example 333. ifeval that compares two string expressions

```
ifeval::["{backend}" == "html5"]
```

If you expect the attribute to resolve to a number, you do not need to enclose the expression in quotes. In this case, the values will be compared as numbers. The same rule applies to boolean values.

You should not attempt to mix value types in a comparison. For example, the following expression is not valid:

Example 334. Invalid ifeval expression

```
ifeval::["{sectnumlevels}" > 3]
```

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 is used and always coerced to a string.

If a value is **not** enclosed in quotes, it's subject to the following type coercion rules:

- an empty value becomes nil (aka null) (and thus safe for use in a comparison).
- 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.

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.

Both sides should be of the same value type. If they are not, the comparison will fail. If the comparison fails, the condition will evaluate to false (i.e., the content inside the directive will be skipped).

The operators follow the same rules as operators in Ruby.

Substitutions

Substitutions are applied to leaf content of a block. Substitutions determine how the text is interpreted. If no substitutions are applied, the text is passed to the converter as entered. Otherwise, the substitutions transform that text.

Substitutions replace references, formatting marks, characters and character sequences, and macros. Substitutions are organized into types and those types are bundled into groups. This page provides an overview of these classifications. Subsequent pages go into detail about each substitution type.

Substitution types

Each substitution type replace characters, markup, attribute references, and macros in text with the appropriate output for a given converter. When a document is processed, up to six substitution types may be carried out depending on the block or inline element's assigned substitution group. The processor runs the substitutions in the following order:

1. Special Characters
2. Quotes (i.e., inline formatting)
3. Attribute References
4. Character Replacements
5. Macros
6. Post Replacements

For convenience, these types are grouped and ordered into substitution groups.

Substitution groups

Each block and inline element has a default substitution group that is applied unless you customize the substitutions for a particular element. Table 5 shows the substitution types that are executed in each group.

Table 5. Substitution types used by each substitution group

Group	Special characters	Quotes	Attributes	Replacements	Macros	Post replacements
Header	Yes	No	Yes	No	No	No
None	No	No	No	No	No	No
Normal	Yes	Yes	Yes	Yes	Yes	Yes
Pass	No	No	No	No	No	No
Verbatim	Yes	No	No	No	No	No

Normal substitution group

The normal substitution group (`normal`) is applied to the majority of the AsciiDoc block and inline elements except for those specific elements listed under the groups described in the next sections.

Header substitution group

The header substitution group (`header`) is applied to metadata lines (author and revision information) in the document header. It's also applied to the values of attribute entries, regardless of whether those entries are defined in the document header or body. Only special characters, attribute references, and the inline pass macro are replaced in elements that fall under the header group.

You can use the inline pass macro in attribute entries to customize the substitution types applied to the attribute's value.

Verbatim substitution group

Literal, listing, and source blocks are processed using the verbatim substitution group (`verbatim`). Only special characters are replaced in these blocks.

Pass substitution group

No substitutions are applied to three of the elements in the pass substitution group (`pass`). These elements include the passthrough block, inline pass macro, and triple plus macro.

The inline single plus and double plus macros also belong to the pass group. Only the special characters substitution is applied to these elements.

None substitution group

The none substitution group (`none`) is applied to comment blocks. No substitutions are applied to comments.

Escaping substitutions

The AsciiDoc syntax offers several approaches for preventing substitutions from being applied. When you want to prevent punctuation and symbols from being interpreted as formatting markup, escaping the punctuation with a backslash may be sufficient. For more comprehensive substitution prevention and control, you can use inline passthrough macros or passthrough blocks.

Special Characters

The special characters substitution step searches for three characters (`<`, `>`, `&`) and replaces them with their named character references.

- The less than symbol, `<`, is replaced with the named character reference `<`.
- The greater than symbol, `>`, is replaced with the named character reference `>`.

- An ampersand, `&`, is replaced with the named character reference `&`.

Default special characters substitution

Table 6 lists the specific blocks and inline elements the special characters substitution step applies to automatically.

Table 6. Blocks and inline elements subject to the special characters substitution

Blocks and elements	Substitution step applied by default
Attribute entry values	Yes
Comments	No
Examples	Yes
Headers	Yes
Literal, listings, and source	Yes
Macros	Yes (except triple plus and inline pass macros)
Open	Yes
Paragraphs	Yes
Passthrough blocks	No
Quotes and verses	Yes
Sidebars	Yes
Tables	Yes
Titles	Yes

specialchars substitution value

The special characters substitution step can be modified on blocks and inline elements. For blocks, the step's name, `specialchars`, can be assigned to the subs attribute. For inline elements, the built-in values `c` or `specialchars` can be applied to inline text to add the special characters substitution step.

Special character substitution precedes attribute substitution, so you 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"`.

Quotes

The replacement of the formatting markup on inline elements is called the quotes substitution step.

Example 335. Syntax input

```
Happy werewolves are *really* slobbery.
```

For instance, when a document containing the markup in Example 335 is converted to HTML, any asterisks enclosing text are replaced with the start and end tags of the `<strong>` element. The resulting HTML can be seen in Example 336 below.

Example 336. HTML output

```
Happy werewolves are <strong>really</strong> slobbery.
```

Table 7 shows the HTML source code that is generated by the quotes substitution step.

Table 7. HTML source code generated from AsciiDoc formatting syntax

Name	AsciiDoc	HTML
emphasis	_word_	word
strong	*word*	word
monospace	`word`	<code>word</code>
superscript	^word^	^{word}
subscript	~word~	_{word}
double curved quotes	"`word`"	“word”
single curved quotes	'`word`'	‘word’

Default quotes substitution

Table 8 lists the specific blocks and inline elements the quotes substitution step applies to automatically.

Table 8. Blocks and inline elements subject to the quotes substitution

Blocks and elements	Substitution step applied by default
Attribute entry values	No
Comments	No
Examples	Yes
Literal, listings, and source	No
Macros	Yes (except passthrough macros)
Open	Yes
Paragraphs	Yes
Passthrough blocks	No

Blocks and elements	Substitution step applied by default
Quotes and verses	Yes
Sidebars	Yes
Tables	Varies
Titles	Yes

quotes substitution value

The quotes substitution step can be modified on blocks and inline elements. For blocks, the step's name, `quotes`, can be assigned to the subs attribute. For inline elements, the built-in values `q` or `quotes` can be applied to inline text to add the quotes substitution step.

Attribute References

Attribute references are replaced with the values of the attribute they reference when processed by the attributes substitution step.

Default attributes substitution

Table 9 lists the specific blocks and inline elements the attributes substitution step applies to automatically.

Table 9. Blocks and inline elements subject to the attributes substitution

Blocks and elements	Substitution step applied by default
Attribute entry values	Yes
Comments	No
Examples	Yes
Headers	Yes
Literal, listings, and source	No
Macros	Yes (except passthrough macros)
Open	Yes
Paragraphs	Yes
Passthrough blocks	No
Quotes and verses	Yes
Sidebars	Yes
Tables	Varies
Titles	Yes

attributes substitution value

The attributes substitution step can be modified on blocks and the inline passthrough. For blocks, the step's name, `attributes`, can be assigned to the subs attribute. For an inline passthrough, the built-in values `a` or `attributes` can be applied to inline text to add or remove the attributes substitution step. Single occurrences of an attribute reference can be escaped by prefixing the expression with a backslash.

Character Replacements

The character replacement substitution step 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. The replacements step depends on the substitutions completed by the special characters step.

Table 10. Textual symbol replacements

Name	Syntax	Unicode Replacement	Rendered	Notes
Copyright	(C)	©	©	
Registered	(R)	®	®	
Trademark	(TM)	™	™	
Em dash	--	—	—	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 ( ). Otherwise, the em dash is followed by a zero-width space () to provide a break opportunity.
Ellipsis	...	…	…	The ellipsis is followed by a zero-width space () to provide a break opportunity.
Single right arrow	->	→	→	
Double right arrow	=>	⇒	⇒	
Single left arrow	<-	←	←	
Double left arrow	<=	⇐	⇐	
Typographic apostrophe	Sam's	Sam’s	Sam’s	The typewriter apostrophe is replaced with the typographic (aka curly or smart) apostrophe.

This substitution step also recognizes HTML and XML character references as well as decimal and hexadecimal Unicode code points and substitutes them for their corresponding decimal form Uni-

code code point.

For example, to produce the § symbol you could write `§`, `§`, or `§`. When the document is processed, `replacements` will preserve the section symbol reference so the reference will appear as § when rendered. In turn, `§` in the AsciiDoc source will display as § in the rendered document.

An AsciiDoc processor allows you to use any of the named character references (aka named entities) defined in HTML (e.g., € resolves to €). 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. The recommendation is avoid using named character references, with the exception of the well-known ones defined in XML (i.e., lt, gt, amp, quot, apos). Instead, use numeric character references (e.g., €).

Anatomy of a character reference

A character reference is a standard sequence of characters that is substituted for a single character by an AsciiDoc processor. 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:

- `†` displays as †
- `€` displays as €
- `◊` displays as ◊

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:

- `†` or `†` displays as †
- `€` or `€` displays as €
- `◊` or `◊` displays as ◊

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, the AsciiDoc syntax doesn't recognize Unicode escape sequences at

this time.

AsciiDoc also provides built-in attributes for representing some common symbols. These attributes and their corresponding output are listed in Character Replacement Attributes Reference.

Default replacements substitution

Table 11 lists the specific blocks and inline elements the replacements substitution step applies to automatically.

Table 11. Blocks and inline elements subject to the replacements substitution

Blocks and elements	Substitution step applied by default
Attribute entry values	No
Comments	No
Examples	Yes
Headers	No
Literal, listings, and source	No
Macros	Yes (except passthrough macros)
Open	Yes
Paragraphs	Yes
Passthrough blocks	No
Quotes and verses	Yes
Sidebars	Yes
Tables	Varies
Titles	Yes

replacements substitution value

The replacements substitution step can be modified on blocks and inline elements. For blocks, the step's name, `replacements`, can be assigned to the subs attribute. For inline elements, the built-in values `r` or `replacements` can be applied to inline text to add the replacements substitution step.

The replacements step depends on the substitutions completed by the special characters step. This is important to keep in mind when applying the `replacements` value to blocks and inline elements.

Macros

The content of inline and block macros, such as cross references, links, and block images, are processed by the macros substitution step. The macros step replaces a macro's content with the appropriate built-in and user-defined configuration.

Default macros substitution

Table 12 lists the specific blocks and inline elements the macros substitution step applies to automatically.

Table 12. Blocks and inline elements subject to the macros substitution

Blocks and elements	Substitution step applied by default
Attribute entry values	Only the pass macro
Comments	No
Examples	Yes
Headers	No
Literal, listings, and source	No
Macros	Yes
Open	Yes
Paragraphs	Yes
Passthrough blocks	No
Quotes and verses	Yes
Sidebars	Yes
Tables	Varies
Titles	Yes

macros substitution value

The macros substitution step can be modified on blocks and inline elements. For blocks, the step's name, `macros`, can be assigned to the subs attribute. For inline elements, the built-in values `m` or `macros` can be applied to inline text to add the macros substitution step.

Post Replacements

The line break character, `+`, is replaced when the `post_replacements` substitution step runs.

Default post replacements substitution

Table 13 lists the specific blocks and inline elements the post replacements substitution step applies to automatically.

Table 13. Blocks and inline elements subject to the post replacements substitution

Blocks and elements	Substitution step applied by default
Attribute entry values	No
Comments	No
Examples	Yes
Headers	No
Literal, listings, and source	No
Macros	Yes (except passthrough macros)
Open	Yes
Paragraphs	Yes
Passthrough blocks	No
Quotes and verses	Yes
Sidebars	Yes
Tables	Varies
Titles	Yes

post_replacements substitution value

The post replacements substitution step can be modified on blocks and inline elements. For blocks, the step's name, `post_replacements`, can be assigned to the subs attribute. For inline elements, the built-in values `p` or `post_replacements` can be applied to inline text to add the post replacements substitution step.

Customize the Substitutions Applied to Blocks

Each block context is associated with a set default substitutions that best suit the content model. However, there are situations where you may need a different set of substitutions to be applied. For example, you may want the AsciiDoc processor to substitute attribute references in a listing block. Therefore, the AsciiDoc language provides a mechanism for altering the substitutions on a block.

The subs attribute

The substitutions that get applied to a block (and to certain inline elements) can be changed or modified using the `subs` element attribute. This attribute accepts a comma-separated list of substitution steps or groups. The list *replaces* the substitutions normally applied to a block unless incremental substitutions are specified.

The names of those substitution steps and groups are as follows:

`none`

Substitution group that disables all substitutions.

`normal`

Substitution group that performs all substitution types except callouts.

`verbatim`

Substitution group that replaces special characters and processes callouts.

`specialchars`

Substitution step that replaces `<`, `>`, and `&` with their corresponding entities. For source blocks, this substitution step enables syntax highlighting as well.

`callouts`

Substitution step that processes callouts in literal, listing, and source blocks.

`quotes`

Substitution step that applies inline text formatting.

`attributes`

Substitution step that replaces attribute references.

`replacements`

Substitution step that replaces hexadecimal Unicode code points and entity, HTML, and XML character references with the characters' decimal Unicode code point. The output of `replacements` may depend on whether the `specialcharacters` substitution was previously applied.

`macros`

Substitution step that processes inline and block macros.

`post_replacements`

Substitution step that processes the line break character (`+`).

If a `+` or `-` modifier is added to a step, the existing substitutions are modified accordingly (see incremental subs). Otherwise, the existing substitutions are replaced. The value also specifies the order in which the substitutions are applied.

The `subs` element attribute does not inherit to nested blocks. It can only be applied to a leaf block, which is any block that cannot have child blocks (e.g., a paragraph or a listing block).

Set the subs attribute on a block

You should almost always prefer to use incremental substitutions. Only switch to exact substitutions when you require very specific control. That's because setting the `subs` attribute on a block *only* uses the substitutions specified. In contrast, incremental substitutions amends the default substitutions for that block.

Let's look at an example where you want to process inline formatting markup in a source block. By default, source blocks (as well as other verbatim blocks) are only subject to the verbatim substitution group (specialchars and callouts). 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 *<name>*") ②
----
```

① The `subs` attribute is set in the attribute list and assigned the `verbatim` and `quotes` values. It's important to reinstate the `verbatim` substitution step to ensure special characters are encoded (which, for source blocks, also enables syntax highlighting).

② The formatting markup in this line will be replaced when the `quotes` substitution step runs.

Here's the result.

```
System.out.println("Hello <name>") ① ②
```

① The `verbatim` value enables any special characters and callouts to be processed.

② The `quotes` value enables the bold text formatting to be processed.

If enabling the quotes substitution step on the whole block causes problems, you can instead enable the macros substitution step, then use the pass macro to enable the quotes substitution step locally.

```
[source,java,subs="verbatim,macros"]
----
System.out.println("No bold *here*");
pass:c,q[System.out.println("Hello *<name>*");] ①
----
```

① The pass macro with the `c,q` target applies the specialchars and quotes substitution steps to the enclosed text.

You may be wondering why `verbatim` is specified in the previous examples since it's applied to literal blocks by default. The reason is that when you specify substitutions without a modifier, it replaces all existing substitutions. Therefore, it's necessary to start with `verbatim` in order to restore the default substitutions. You can avoid having to do this by using incremental substitutions instead, which is covered in the next section.

Add and remove substitution types from a default substitution group

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 attribute references are substituted. The `verbatim` substitution group will not be applied. To remedy

this situation, AsciiDoc provides a syntax to append or remove substitutions instead of replacing them outright.

You can add or remove a substitution type from the default substitution group 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 the beginning of a listing block's default substitution group by placing the plus (`+`) modifier at the end of the `attributes` value.

Example 337. Add attributes substitution to default substitution group

```
[source,xml,subs="attributes+"]
----
<version>{version}</version>
----
```

Similarly, you can remove the `callouts` substitution from a block's default substitution group by placing the minus (`-`) modifier in front of the `callouts` value.

Example 338. Remove callouts substitution from default substitution group

```
[source,xml,subs="-callouts"]
.An illegal XML tag
----
<1>
  content inside "1" tag
</1>
----
```

You can also specify whether the substitution type is added to the end of the substitution group. If a `+` comes before the name of the substitution, then it's added to the end of the existing list, whereas 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 `attributes` substitution step is added to the beginning of the default substitution group, the `replacements` step is added to the end of the group, and the `callouts` step is removed from the group.

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: +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 tree Processor extension.

Customize the Substitutions Applied to Text

The inline pass macro (`pass:[]`) accepts the shorthand values in addition to the longhand values for specifying substitution types.

- `c` or `specialchars`
- `q` or `quotes`
- `a` or `attributes`
- `r` or `replacements`
- `m` or `macros`
- `p` or `post_replacements`

Apply substitutions to inline text

Custom substitutions can also be applied to inline text with the pass macro. For instance, let's assume you need to mark a span of text as deleted using the HTML element `<del>` in your AsciiDoc document. You'd do this with the inline pass macro.

Example 339. Inline pass macro syntax

```
The text pass:[<del>strike this</del>] is marked as deleted.
```

The result of Example 339 is rendered below.

The text ~~strike this~~ is marked as deleted.

However, you also need to bold the text and want to use the AsciiDoc markup for that formatting. In this case, you'd assign the `quotes` substitution to the inline pass macro.

Example 340. Assign quotes to inline pass macro

```
The text pass:q[<del>strike *this*</del>] is marked as deleted, inside of which the
word "`me`" is bold.
```

The result of Example 340 is rendered below.

The text ~~strike **this**~~ is marked as deleted, inside of which the word "me" is bold.

You can also assign custom substitutions to inline text that's in a block. In the listing block below, we want to process the inline formatting on the second line.

Example 341. Listing block with inline formatting

```
[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 the `pass` macro within the block to be processed.

② The `pass` macro is assigned the `quotes` value. Text within the square brackets will be formatted.

The result of Example 341 is rendered below.

```
I better not contain *bold* or _italic_ text.
But I should contain bold text.
```

Escape and Prevent Substitutions

The AsciiDoc syntax offers several approaches for preventing substitutions from being applied.

Escape with backslashes

To prevent a punctuation character from being interpreted as an attribute reference or formatting syntax (e.g., `_`, `^`) in normal content, prepend the character with a backslash (`\`).

Example 342. Prevent unintended substitutions with a backslash in normal content

```
In /items/\{id}, the id attribute isn't replaced.
The curly braces around it are preserved.

\*Stars* isn't displayed as bold text.
The asterisks around it are preserved.

\&sect; appears as an entity reference.
It's not converted into the section symbol (&#167;).

\=> The backslash prevents the equals sign followed by a greater
than sign from combining to form a double arrow character (=>).

\[[Word]] is not interpreted as an anchor.
The double brackets around it are preserved.

[\[[Word]]] is not interpreted as a bibliography anchor.
The triple brackets around it are preserved.

\((DD AND CC) OR (DD AND EE)) is not interpreted as a flow index term.
The double brackets around it are preserved.

The URL \https://example.org isn't converted into an active link.
```

The backslash can also prevent character replacements, macros, and attribute replacements. The results of Example 342 are below.

In /items/{id}, the id attribute isn’t replaced. The curly braces around it are preserved.

Stars isn’t displayed as bold text. The asterisks around it are preserved.

§ appears as an entity reference. It’s not converted into the section symbol (§).

=> The backslash prevents the equals sign followed by a greater than sign from combining to form a double arrow character (⇒).

[[subs:prevent:::Word]] is not interpreted as an anchor. The double brackets around it are preserved.

[[[subs:prevent:::Word]]] is not interpreted as a bibliography anchor. The triple brackets around it are preserved.

((DD AND CC) OR (DD AND EE)) is not interpreted as a flow index term. The double brackets around it are preserved.

The URL https://example.org isn’t converted into an active link.

Notice that the backslash is removed so it doesn’t display in your output.

To prevent two adjacent characters (e.g., __, ##), from being interpreted as AsciiDoc syntax you need to precede it with two backslashes (\\).

Example 343. Prevent unintended substitutions with two backslashes in normal content

```
The text \\__func__ will appear with two underscores
in front of it and after it.
It won't be italicized.
```

The results of Example 343 are below.

The text __func__ will appear with two underscores in front of it and after it. It won't be italicized.

Passthroughs

A passthrough is the primary mechanism by which to escape content in AsciiDoc. They're 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.

You can control and prevent substitutions in inline text with the inline passthrough macros and for entire blocks of content with the block passthrough.

The inline + passthrough takes precedence over all other inline formatting. Therefore, if you need to output a literal plus when it would otherwise match a passthrough, you have two options.

First, you can escape it using the {plus} attribute reference:

```
`{plus}` and `{plus}`
```

Alternately, you can escape the pair using a backslash.

```
`\+` and `+`
```

The backslash is only required before the pair, not before each occurance of the plus.

Passthroughs

A passthrough is a mechanism in AsciiDoc for passing chunks of content directly through to the output. Most passthroughs give you control over which substitutions are applied to the content. AsciiDoc provides both block and inline forms of the passthrough.

The block form of the passthrough is represented either by the `++++` block delimiters or the `pass` style on a paragraph. The main use of the block form is to pass a chunk of non-AsciiDoc content directly through to the output. For example, you can use the passthrough block to pass raw HTML to the HTML output. However, by doing so, you're coupling your AsciiDoc content to an output format, thus making it less portable. It's best either to leave the use of the passthrough block up to an extension, or enclose it in a preprocessor conditional.

The inline form of the passthrough comes in more forms and thus has more uses. An inline passthrough is represented by the macro or by pairs of one to three pluses. Only the macro gives you control over the substitutions that are applied. While an inline passthrough can be used to pass raw content like HTML to the output, far more often it's used as a way to escape content from inline formatting. For example, you can use an inline passthrough to output characters that would otherwise be replaced, such as three sequential periods.

Passthrough Blocks

The `pass` style and delimited passthrough block exclude the block's content from all substitutions unless the `subs` attribute is set.

Pass style syntax

The `pass` style can also be set on a paragraph or an open block.

```
[pass]
<del>strike this</del> is marked as deleted.
```

Delimited passthrough block syntax

A passthrough block is delimited by four plus signs (`++++`).

```
++++
<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).

Control substitutions on a passthrough block

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.
++++
```

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.

Inline Passthroughs

AsciiDoc supports several inline passthrough macros and shorthands. Inline passthroughs are designed to prevent subsitutions for regions of text, or to give you more fine-grained control over which substitutions are applied.

Due to the fact that inline syntax in AsciiDoc is processed using substitutions rather than a descending grammar, it's possible to create invalid interleaving of inline elements, or other adverse interactions, that leads to invalid or illogical output. The inline passthrough provides a bailout option to mitigate these entanglements. This problem is expected to be resolved properly by the AsciiDoc Language Specification, which will mandate that inline syntax is parsed as a tree rather than through substitutions (to the degree possible).

Inline passthrough macros

single and double plus

A special syntax for preventing inline text from being formatted. Only special characters are replaced in the output format. The substitutions can't be modified for this type of passthrough.

triple plus

A special inline syntax for designating passthrough content. No substitutions are applied nor can they be added using the step and group substitution values.

inline pass macro

An inline macro named `pass` that can be used to passthrough content. You can apply specific substitutions to the macro's target using substitution types and groups.

```
pass:[content like #{variable} passed directly to the output] followed by normal
content.

content with only select substitutions applied: pass:c,a[__<{email}>__]
```

When you need to prevent or control the substitutions on one or more blocks of content, use a delimited passthrough block or the pass block style.

Single and double plus

The single and double plus passthroughs prevent text enclosed in either a pair of single pluses (`+`) or a pair of double pluses (`++`) from being formatted.

```
A +word+, a +sequence of words+, or ++char++acters that are escaped from formatting.
```

The single and double pluses represent the constrained and unconstrained passthrough, respectively. They have boundaries that match the constrained and unconstrained formatting marks. The main difference, however, is that they are applied first to suppress formatting.

This type of passthrough is intended to suppress any special meaning of the source text itself. This passthrough type still ensures, however, that the content is properly escaped in the output. That means the special characters substitution is still applied.

As with all constrained pairs, the single plus passthrough is designed to be used around a word or phrase.

```
A word or phrase between single pluses, such as +/document/{id}+, is not substituted.
However, the special characters +<+ and +>+ are still escaped in the output.

You can also escape formatting marks, like +``+.
```

Being an unconstrained pair, the double plus passthrough can be used anywhere in the text.

```
Text formatting is not applied to a link target if it is surrounded by double pluses.
For example, link:++https://example.org/now_this__link_works.html++[].

You can also escape formatting marks, like all-natural++*++.
```

```
An attribute reference within a word, such as dev++{conf}++, is not replaced.
```

The single and plus passthroughs are a surefire alternative to backslash escaping.

Note that the single and plus passthroughs only prevent substitutions. They do not format the text in monospace. If you want to do both, you must enclose the pair in a monospace formatting pair, known as literal monospace.

Triple plus

The triple plus passthrough excludes content enclosed in a pair of triple pluses (+++) from all substitutions.

```
+++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 +++<del>strike this</del>+++ is marked as deleted.
```

The text ~~strike this~~ is marked as deleted.

Inline pass macro

Like other inline passthroughs, the inline pass macro can be used to control the substitutions applied to a run of text. To exclude inline content from all of the substitutions, enclose it in the inline pass macro.

Here's one way to format text as underline when generating HTML from AsciiDoc:

```
The text pass:[<del>strike this</del>] is marked as deleted.
```

And here's the result.

The text ~~strike this~~ is marked as deleted.

Using passthroughs to send content directly to the output can couple your content to a specific output format, such as HTML. To avoid this risk, you should consider using conditional preprocessor directives to select content for different output formats based on the current backend.

What sets the inline pass macro apart from the alternatives is that it allows the substitutions to be customized. The inline pass macro also plays a critical role in the document header. In fact, it's the

only macro that is processed in the document header by default as part of the header substitution group (though it can be used to enable other substitutions, as demonstrated in this section).

Let's look at how to use the inline pass macro to hand select substitutions.

Custom substitutions

You can customize the substitutions applied to the content of an inline pass macro by specifying one or more substitution values in the target of the macro. Multiple values must be separated by commas and may not contain any spaces. The substitution value is either the formal name of a substitution type or group, or its shorthand.

The following table lists the allowable substitution values:

Substitution values accepted by the inline pass macro

Shorthand	Substitution Type
`c`	`specialchars`
`q`	`quotes`
`a`	`attributes`
`r`	`replacements`
`m`	`macros`
`p`	`post replacements`
Shorthand	**Substitution Group**
`n`	`normal`
`v`	`verbatim`

For example, the quotes substitution (i.e., `q` or `quotes`) is enabled on the inline passthrough macro as follows:

```
The text pass:q[<del>strike *this*</del>] is marked as deleted.
```

Here's the result.

The text ~~strike **this**~~ is marked as deleted.

To enable multiple substitution groups, separate each value in the macro target by a comma:

```
The text pass:q,a[<del>strike _{docname}_</del>] is marked as deleted.
```

Here's the result.

The text ~~strike *pass-macro*~~ is marked as deleted.

Nesting blocks and passthroughs

When you're using passthroughs inside literal and listing blocks, it can be easy to forget that the single plus and triple plus passthroughs are macros substitutions. To enable the passthroughs, 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();
----
```

To learn more about applying substitutions to blocks, see Customize the Substitutions Applied to Blocks.

```
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
            .antMatchers("/resources/**").permitAll()
            .anyRequest().authenticated()
            .and()
        .formLogin()
            .loginPage("/login")
            .permitAll();
```

Reference

Syntax Quick Reference

The examples on this page demonstrate the output produced by the built-in HTML converter. An AsciiDoc converter is expected to produce complementary output when generating other output formats, such as PDF, EPUB, and DocBook.

Paragraphs

Example 344. Paragraph

```
Paragraphs don't require special markup in AsciiDoc.
A paragraph is defined by one or more consecutive lines of text.
Line breaks within a paragraph are not displayed.

Leave at least one empty line to begin a new paragraph.
```

▼ *View result of Example 344*

Paragraphs don't require special markup in AsciiDoc. A paragraph is defined by one or more consecutive lines of text. Line breaks within a paragraph are not displayed.

Leave at least one empty line to begin a new paragraph.

Example 345. Literal paragraph

```
A normal paragraph.

 A literal paragraph.
 One or more consecutive lines indented by at least one space.

 The text is shown in a fixed-width (typically monospace) font.
 The lines are preformatted (i.e., as formatted in the source).
 Spaces and newlines,
 like the ones in this sentence,
 are preserved.
```

▼ *View result of Example 345*

A normal paragraph.

```
A literal paragraph.
One or more consecutive lines indented by at least one space.
```

```
The text is shown in a fixed-width (typically monospace) font.
The lines are preformatted (i.e., as formatted in the source).
```

```
Spaces and newlines,
like the ones in this sentence,
are preserved.
```

Example 346. Hard line breaks

```
Roses are red, +
violets are blue.

[%hardbreaks]
A ruby is red.
Java is black.
```

▼ *View result of Example 346*

Roses are red,
violets are blue.

A ruby is red.
Java is black.

Example 347. Lead paragraph

```
[.lead]
This text will be styled as a lead paragraph (i.e., larger font).

This paragraph will not be.
```

▼ *View result of Example 347*

This text will be styled as a lead paragraph (i.e., larger font).

This paragraph will not be.

The default Asciidoctor stylesheet automatically styles the first paragraph of the preamble as a lead paragraph if no role is specified on that paragraph.

Text formatting

Example 348. Constrained bold, italic, and monospace

```
It has *strong* significance to me.

I _cannot_ stress this enough.

Type `OK` to accept.

That *_really_* has to go.
```

```
Can't pick one? Let's use them `*_all_*`.
```

▼ *View result of Example 348*

It has **strong** significance to me.

I *cannot* stress this enough.

Type `OK` to accept.

That ***really*** has to go.

Can’t pick one? Let’s use them **`all`**.

Example 349. Unconstrained bold, italic, and monospace

```
**C**reate, **R**ead, **U**pdate, and **D**elete (CRUD)

That's fan__freakin__tastic!

Don't pass generic ``Object``s to methods that accept ``String``s!

It was Beatle**__mania__**!
```

▼ *View result of Example 349*

Create, **R**ead, **U**pdate, and **D**elete (CRUD)

That’s fan*freakin*tastic!

Don’t pass generic `Object`s to methods that accept `String`s!

It was Beatle***mania***!

Example 350. Highlight, underline, strikethrough, and custom role

```
Mark my words, #automation is essential#.

##Mark##up refers to text that contains formatting ##mark##s.

Where did all the [.underline]#cores# go?

We need [.line-through]#ten# twenty VMs.

A [.myrole]#custom role# must be fulfilled by the theme.
```

▼ *View result of Example 350*

Mark my words, automation is essential.

Markup refers to text that contains formatting marks.

Where did all the cores go?

We need ~~ten~~ twenty VMs.

A custom role must be fulfilled by the theme.

Example 351. Superscript and subscript

```
^super^script

~sub~script
```

▼ *View result of Example 351*

^{super}script

_{sub}script

Example 352. Smart quotes and apostrophes

```
"`double curved quotes`"

'`single curved quotes`'

Olaf's desk was a mess.

A ``std::vector````'s size is the number of items it contains.

All of the werewolves`' desks were a mess.

Olaf had been with the company since the `'00s.
```

▼ *View result of Example 352*

“double curved quotes”

‘single curved quotes’

Olaf’s desk was a mess.

A `std::vector`’s size is the number of items it contains.

All of the werewolves’ desks were a mess.

Olaf had been with the company since the ’00s.

Links

Example 353. Autolinks, URL macro, and mailto macro

```
https://asciidoctor.org - automatic!

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

```
devel@discuss.example.org

mailto:devel@discuss.example.org[Discuss]

mailto:join@discuss.example.org[Subscribe,Subscribe me,I want to join!]
```

▼ *View result of Example 353*

asciidoctor.org - automatic!

Asciidoctor

devel@discuss.example.org

Discuss

Subscribe

Example 354. URL macros with attributes

```
https://chat.asciidoc.org[Discuss AsciiDoc,role=external,window=_blank]

https://chat.asciidoc.org[Discuss AsciiDoc^]
```

▼ *View result of Example 354*

Discuss AsciiDoc

Discuss AsciiDoc

The `link:` macro prefix is *not* required when the target starts with a URL scheme like `https:`. The URL scheme acts as an implicit macro prefix.

If the link text contains a comma and the text is followed by one or more named attributes, you must enclose the text in double quotes. Otherwise, the text will be cut off at the comma (and the remaining text will get pulled into the attribute parsing).

Example 355. URLs with spaces and special characters

```
link:++https://example.org/?q=[a b]++[URL with special characters]

https://example.org/?q=%5Ba%20b%5D[URL with special characters]
```

Example 356. Link to relative file

```
link:index.html[Docs]
```

Example 357. Link using a Windows UNC path

```
link:\\server\share\whitepaper.pdf[Whitepaper]
```

Example 358. Inline anchors

```
[[bookmark-a]]Inline anchors make arbitrary content referenceable.

[#bookmark-b]#Inline anchors can be applied to a phrase like this one.#

anchor:bookmark-c[]Use a cross reference to link to this location.

[[bookmark-d,last paragraph]]The xreflabel attribute will be used as link text in the
cross-reference link.
```

Example 359. Cross references

```
See <<paragraphs>> to learn how to write paragraphs.

Learn how to organize the document into <<section-titles,sections>>.
```

▼ *View result of Example 359*

See Paragraphs to learn how to write paragraphs.

Learn how to organize the document into sections.

Example 360. Inter-document cross references

```
Refer to xref:document-b.adoc#section-b[Section B of Document B] for more information.

If you never return from xref:document-b.adoc[Document B], we'll send help.
```

Document header

The document header is optional. The header may not contain any empty lines and must be separated from the content by at least one empty line.

Example 361. Title

```
= Document Title

This document provides...
```

Example 362. Title and author line

```
= Document Title
Author Name <author@email.org>
```

```
This document provides...
```

Example 363. Title, author line, and revision line

```
= Document Title
Author Name <author@email.org>; Another Author <a.author@email.org>
v2.0, 2019-03-22

This document provides...
```

You cannot have a revision line without an author line.

Example 364. Document header with attribute entries

```
= Document Title
Author Name <author@email.org>
v2.0, 2019-03-22
:toc:
:homepage: https://example.org

This document provides...
```

Section titles

When the document type is `article` (the default), the document can only have one level 0 section title (`=`), which is the document title (i.e., doctitle).

Example 365. Article section levels

```
= 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
```

▼ *View result of Example 365*

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

The book document type can have additional level 0 section titles, which are interpreted as parts. The presence of at least one part implicitly makes the document a multi-part book.

Example 366. Book section levels

```
= Document Title (Level 0)

== Level 1 Section Title

= Level 0 Section Title (Part)

== Level 1 Section Title

=== Level 2 Section Title

==== Level 3 Section Title

===== Level 4 Section Title

====== Level 5 Section Title

= Another Level 0 Section Title (Part)
```

Example 367. Discrete heading (not a section)

```
[discrete]
=== I'm an independent heading!

This paragraph is its sibling, not its child.
```

▼ *View result of Example 367*

I’m an independent heading!

This paragraph is its sibling, not its child.

Automatic TOC

Example 368. Activate Table of Contents for a document

```
= Document Title
Doc Writer <doc.writer@email.org>
:toc:
```

The Table of Contents’ title, displayed section depth, and position can be customized.

Includes

Example 369. Include document parts

```
= Reference Documentation
Lead Developer

This is documentation for project X.

include::basics.adoc[]

include::installation.adoc[]

include::example.adoc[]
```

Example 370. Include content by tagged regions or lines

```
include::filename.txt[tag=definition]

include::filename.txt[lines=5..10]
```

Example 371. Include content from a URL

```
include::https://raw.githubusercontent.com/asciidoctor/asciidoctor/main/README.adoc[]
```

Including content from a URL is potentially dangerous, so it’s disabled if the safe mode is SECURE or greater. Assuming the safe mode is less than SECURE, you must also set the `allow-uri-read` attribute to permit the AsciiDoc processor to read content from a URL.

Lists

Example 372. Unordered list

```
* List item
** Nested list item
*** Deeper nested list item
* List item
 ** Another nested list item
* List item
```

▼ *View result of Example 372*

- List item
  - Nested list item
    - Deeper nested list item
- List item
  - Another nested list item
- List item

An empty line is required before and after a list to separate it from other blocks. You can force two adjacent lists apart by adding an empty attribute list (i.e., `[]`) above the second list or by inserting an empty line followed by a line comment after the first list. If you use a line comment, the convention is to use `//-` to provide a hint to other authors that it's serving as a list divider.

Example 373. Unordered list max level nesting

```
* Level 1 list item
** Level 2 list item
*** Level 3 list item
**** Level 4 list item
***** Level 5 list item
****** etc.
* Level 1 list item
```

▼ *View result of Example 373*

- Level 1 list item
  - Level 2 list item
    - Level 3 list item
      - Level 4 list item
        - Level 5 list item
          - etc.
- Level 1 list item

The unordered list marker can be changed using a list style (e.g., `square`).

Example 374. Ordered list

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

▼ *View result of Example 374*

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

Example 375. Ordered list max level nesting

```
. Level 1 list item
.. Level 2 list item
... Level 3 list item
.... Level 4 list item
..... Level 5 list item
. Level 1 list item
```

▼ *View result of Example 375*

1. Level 1 list item
    a. Level 2 list item
        i. Level 3 list item
            A. Level 4 list item
                I. Level 5 list item
2. Level 1 list item

Ordered lists support numeration styles such as `lowergreek` and `decimal-leading-zero`.

Example 376. Checklist

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

▼ *View result of Example 376*

- ☑ checked
- ☑ also checked

- ☐ not checked

  normal list item

Example 377. Description list

```
First term:: The description can be placed on the same line
as the term.
Second term::
Description of the second term.
The description can also start on its own line.
```

▼ *View result of Example 377*

First term

The description can be placed on the same line as the term.

Second term

Description of the second term. The description can also start on its own line.

Example 378. Question and answer list

```
[qanda]
What is the answer?::
This is the answer.

Are cameras allowed?::
Are backpacks allowed?::
No.
```

▼ *View result of Example 378*

1. *What is the answer?*

   This is the answer.

2. *Are cameras allowed?*

   Are backpacks allowed?

   No.

Example 379. Mixed

```
Operating Systems::
  Linux:::
    . Fedora
      * Desktop
    . Ubuntu
      * Desktop
      * Server
  BSD:::
    . FreeBSD
    . NetBSD
```

```
Cloud Providers::
  PaaS:::
    . OpenShift
    . CloudBees
  IaaS:::
    . Amazon EC2
    . Rackspace
```

▼ *View result of Example 379*

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

Lists can be indented. Leading whitespace is not significant.

Example 380. Complex content in outline lists

```
* Every list item has at least one paragraph of content,
  which may be wrapped, even using a hanging indent.
+
Additional paragraphs or blocks are adjoined by putting
a list continuation on a line adjacent to both blocks.
+
list continuation:: a plus sign (`{plus}`) on a line by itself

* A literal paragraph does not require a list continuation.

 $ cd projects/my-book
```

```
* AsciiDoc lists may contain any compound content.
+
|===
|Column 1, Header Row |Column 2, Header Row

|Column 1, Row 1
|Column 2, Row 1
|===
```

▼ *View result of Example 380*

- Every list item has at least one paragraph of content, which may be wrapped, even using a hanging indent.

  Additional paragraphs or blocks are adjoined by putting a list continuation on a line adjacent to both blocks.

  list continuation
  : a plus sign (+) on a line by itself

- A literal paragraph does not require a list continuation.

  ```
  $ cd projects/my-book
  ```

- AsciiDoc lists may contain any compound content.

Column 1, Header Row	Column 2, Header Row
Column 1, Row 1	Column 2, Row 1

Images

You can use the imagesdir attribute to avoid hard coding the common path to your images in every image macro. The value of this attribute can be an absolute path, relative path, or base URL. If the image target is a relative path, the attribute's value is prepended (i.e., it's resolved relative to the value of the `imagesdir` attribute). If the image target is a URL or absolute path, the attribute's value is *not* prepended.

Example 381. Block image macro

```
image::sunset.jpg[]

image::sunset.jpg[Sunset]

.A mountain sunset
[#img-sunset,caption="Figure 1: ",link=https://www.flickr.com/photos/javh/5448336655]
image::macros:sunset.jpg[Sunset,200,100]
```

```
image::https://asciidoctor.org/images/octocat.jpg[GitHub mascot]
```

▼ *View result of Example 381*

Figure 1: A mountain sunset

Two colons following the image keyword in the macro (i.e., `image::`) indicates a block image (aka figure), whereas one colon following the image keyword (i.e., `image:`) indicates an inline image. (All macros follow this pattern). You use an inline image when you need to place the image in a line of text. Otherwise, you should prefer the block form.

Example 382. Inline image macro

```
Click image:play.png[] to get the party started.

Click image:pause.png[title=Pause] when you need a break.
```

▼ *View result of Example 382*

Click  to get the party started.

Click  when you need a break.

Example 383. Inline image macro with positioning role

```
image:sunset.jpg[Sunset,150,150,role=right] What a beautiful sunset!
```

▼ *View result of Example 383*

 What a beautiful sunset!

Example 384. Embedded

```
= Document Title
:data-uri:
```

When the `data-uri` attribute is set, all images in the document—including admonition icons—are embedded into the document as data URIs. You can also pass it as a command line argument using `-a data-uri`.

Audio

Example 385. Block audio macro

```
audio::ocean-waves.wav[]

audio::ocean-waves.wav[start=60,opts=autoplay]
```

You can control the audio settings using additional attributes and options on the macro.

Videos

Example 386. Block video macro

```
video::video-file.mp4[]

video::video-file.mp4[width=640,start=60,opts=autoplay]
```

Example 387. Embedded YouTube video

```
video::RvRhUHTV_8k[youtube]
```

Example 388. Embedded Vimeo video

```
video::67480300[vimeo]
```

You can control the video settings using additional attributes and options on the macro.

Keyboard, button, and menu macros

You must set the `experimental` attribute in the document header to enable these macros.

Example 389. Keyboard macro

```
|===
|Shortcut |Purpose

|kbd:[F11]
|Toggle fullscreen

|kbd:[Ctrl+T]
|Open a new tab
|===
```

▼ *View result of Example 389*

Shortcut	Purpose
F11	Toggle fullscreen
Ctrl + T	Open a new tab

Example 390. Menu macro

```
To save the file, select menu:File[Save].

Select menu:View[Zoom > Reset] to reset the zoom level to the default setting.
```

▼ *View result of Example 390*

To save the file, select **File › Save**.

Select **View › Zoom › Reset** to reset the zoom level to the default setting.

Example 391. Button macro

```
Press the btn:[OK] button when you are finished.

Select a file in the file navigator and click btn:[Open].
```

▼ *View result of Example 391*

Press the **[OK]** button when you are finished.

Select a file in the file navigator and click **[Open]**.

Literals and source code

Example 392. Inline literal monospace

```
Output literal monospace text, such as `+{backtick}+` or `+http://localhost:8080+`, by
enclosing the text in a pair of pluses surrounded by a pair of backticks.
```

▼ *View result of Example 392*

Output literal monospace text, such as `{backtick}` or `http://localhost:8080`, by enclosing the text in a pair of pluses surrounded by a pair of backticks.

Example 393. Literal paragraph

```
Normal line.

 Indent line by one space to create a literal line.

Normal line.
```

▼ *View result of Example 393*

Normal line.

```
Indent line by one space to create a literal line.
```

Normal line.

Example 394. Literal block

```
....
error: 1954 Forbidden search
absolutely fatal: operation lost in the dodecahedron of doom
```

```
Would you like to try again? y/n
....
```

▼ *View result of Example 394*

```
error: 1954 Forbidden search
absolutely fatal: operation lost in the dodecahedron of doom

Would you like to try again? y/n
```

Example 395. Listing block with title

```
.Gemfile.lock
----
GEM
  remote: https://rubygems.org/
  specs:
    asciidoctor (2.0.15)

PLATFORMS
  ruby

DEPENDENCIES
  asciidoctor (~> 2.0.15)
----
```

▼ *View result of Example 395*

Listing 1. Gemfile.lock

```
GEM
  remote: https://rubygems.org/
  specs:
    asciidoctor (2.0.15)

PLATFORMS
  ruby

DEPENDENCIES
  asciidoctor (~> 2.0.15)
```

Example 396. Source block with title and syntax highlighting

```
.Some Ruby code
[source,ruby]
----
require 'sinatra'
```

```
get '/hi' do
  "Hello World!"
end
----
```

▼ *View result of Example 396*

Listing 1. Some Ruby code

```ruby
require 'sinatra'

get '/hi' do
  "Hello World!"
end
```

You must enable source highlighting by setting the `source-highlighter` attribute in the document header, CLI, or API.

```
:source-highlighter: rouge
```

See Syntax Highlighting to learn which values are accepted when using Asciidoctor.

Example 397. Source block with callouts

```
[source,ruby]
----
require 'sinatra' // <1>

get '/hi' do // <2>
  "Hello World!" // <3>
end
----
<1> Library import
<2> URL mapping
<3> HTTP response body
```

▼ *View result of Example 397*

```ruby
require 'sinatra' ①

get '/hi' do ②
  "Hello World!" ③
end
```

① Library import

② URL mapping

③ HTTP response body

Example 398. Make callouts non-selectable

```
----
line of code // <1>
line of code # <2>
line of code ;; <3>
line of code <!--4-->
----
<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.
<4> A callout behind a line comment for XML or SGML languages like HTML.
```

▼ *View result of Example 398*

```
line of code ①
line of code ②
line of code ③
line of code ④
```

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.

④ A callout behind a line comment for XML or SGML languages like HTML.

Example 399. Source block content included from a file

```
[,ruby]
----
include::app.rb[]
----
```

Example 400. Source block content included from file relative to source directory

```
:sourcedir: src/main/java

[source,java]
----
include::{sourcedir}/org/asciidoctor/Asciidoctor.java[]
----
```

Example 401. Strip leading indentation from partial file content

```
[source,ruby]
----
include::lib/app.rb[tag=main,indent=0]
```

```
----
```

The indent attribute is frequently used when including source code by tagged region or lines. It can be specified on the include directive itself or the enclosing literal, listing, or source block.

When indent is 0, the leading block indent is stripped.

When indent is greater than 0, the leading block indent is first stripped, then a block is indented by the number of columns equal to this value.

Example 402. Source paragraph (no empty lines)

```
[source,xml]
<meta name="viewport"
  content="width=device-width, initial-scale=1.0">

This is normal content.
```

▼ *View result of Example 402*

```
<meta name="viewport"
  content="width=device-width, initial-scale=1.0">
```

This is normal content.

Admonitions

Example 403. Admonition paragraph

```
NOTE: An admonition draws the reader's attention to auxiliary information.

Here are the other built-in admonition types:

IMPORTANT: Don't forget the children!

TIP: Look for the warp zone under the bridge.

CAUTION: Slippery when wet.

WARNING: The software you're about to use is untested.

IMPORTANT: Sign off before stepping away from your computer.
```

▼ *View result of Example 403*

An admonition draws the reader's attention to auxiliary information.

Here are the other built-in admonition types:

Don't forget the children!

Look for the warp zone under the bridge.

Slippery when wet.

The software you're about to use is untested.

Sign off before stepping away from your computer.

Example 404. Admonition block

```
[NOTE]
====
An admonition block may contain compound content.

.A list
- one
- two
- three

Another paragraph.
====
```

▼ *View result of Example 404*

An admonition block may contain compound content.

A list

- one
- two
- three

Another paragraph.

More delimited blocks

Any block can have a title. A block title is defined using a line of text above the block that starts with a dot. That dot cannot be followed by a space. For block images, the title is displayed below the block. For all other blocks, the title is typically displayed above it.

Example 405. Sidebar block

```
.Optional Title
****
```

```
Sidebars are used to visually separate auxiliary bits of content
that supplement the main text.
****
```

▼ *View result of Example 405*

Optional Title

Sidebars are used to visually separate auxiliary bits of content that supplement the main text.

Example 406. Example block

```
====
Here's a sample AsciiDoc document:

----
= Title of Document
Doc Writer
:toc:

This guide provides...
----

The document header is useful, but not required.
====
```

▼ *View result of Example 406*

Here’s a sample AsciiDoc document:

```
= Title of Document
Doc Writer
:toc:

This guide provides...
```

The document header is useful, but not required.

Example 407. Blockquotes

```
[quote,Abraham Lincoln,Address delivered at the dedication of the Cemetery at
Gettysburg]
____
Four score and seven years ago our fathers brought forth
on this continent a new nation...
____
```

```
[quote,Albert Einstein]
A person who never made a mistake never tried anything new.


____
A person who never made a mistake never tried anything new.
____

[quote,Charles Lutwidge Dodgson,'Mathematician and author, also known as
https://en.wikipedia.org/wiki/Lewis_Carroll[Lewis Carroll]']
____
If you don't know where you are going, any road will get you there.
____

"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
```

▼ *View result of Example 407*

> Four score and seven years ago our fathers brought forth on this continent a new nation…
>
> — Abraham Lincoln, Address delivered at the dedication of the Cemetery at Gettysburg

> A person who never made a mistake never tried anything new.
>
> — Albert Einstein

> A person who never made a mistake never tried anything new.

> If you don’t know where you are going, any road will get you there.
>
> — Charles Lutwidge Dodgson, Mathematician and author, also known as Lewis Carroll

> 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

Example 408. Open blocks

```
--
An open block can be an anonymous container,
or it can masquerade as any other block.
--

[source]
--
```

```
puts "I'm a source block!"
--
```

▼ *View result of Example 408*

An open block can be an anonymous container, or it can masquerade as any other block.

```
puts "I'm a source block!"
```

Example 409. Passthrough block

```
++++
<p>
Content in a passthrough block is passed to the output unprocessed.
That means you can include raw HTML, like this embedded Gist:
</p>

<script src="https://gist.github.com/mojavelinux/5333524.js">
</script>
++++
```

▼ *View result of Example 409*

```
<p>
Content in a passthrough block is passed to the output unprocessed.
That means you can include raw HTML, like this embedded Gist:
</p>

<script src="https://gist.github.com/mojavelinux/5333524.js">
</script>
```

Example 410. Customize block substitutions

```
:release-version: 2.4.3

[source,xml,subs=attributes+]
----
<dependency>
  <groupId>org.asciidoctor</groupId>
  <artifactId>asciidoctorj</artifactId>
  <version>{release-version}</version>
</dependency>
----
```

▼ *View result of Example 410*

```
<dependency>
  <groupId>org.asciidoctor</groupId>
  <artifactId>asciidoctorj</artifactId>
  <version>2.4.3</version>
```

```
</dependency>
```

Tables

Example 411. Table with a title, two columns, a header row, and two rows of content

```
.Table Title
|===
|Column 1, Header Row |Column 2, Header Row ①
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
|===
```

① Unless the `cols` attribute is specified, the number of columns is equal to the number of cell separators on the first (non-empty) line.

② When an empty line immediately follows a non-empty line at the start of the table, the cells in the first line get promoted to the table header.

▼ *View result of Example 411*

Table 1. Table Title

Column 1, Header Row	Column 2, Header 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

Example 412. Table with two columns, a header row, and two rows of content

```
[%header,cols=2*] ①
|===
|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
|===
```

① The `*` in the `cols` attribute is the repeat operator. It means repeat the column specification across the remaining columns. In this case, we are repeating the default formatting across 2 columns. When the cells in the header are not defined on a single line, you must use the `cols` attribute to set the number of columns in the table and the `%header` option (or `options=header` attribute) to promote the first row to the table header.

▼ *View result of Example 412*

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

Example 413. Table with three columns, a header row, and two rows of content

```
.Applications
[cols="1,1,2"] ①
|===
|Name |Category |Description

|Firefox
|Browser
|Mozilla Firefox is an open source web browser.
It's designed for standards compliance,
performance, portability.

|Arquillian
|Testing
|An innovative and highly extensible testing platform.
Empowers developers to easily create real, automated tests.
|===
```

① In this example, the `cols` attribute has two functions. It specifies that this table has three columns, and it sets their relative widths.

▼ *View result of Example 413*

Table 1. Applications

Name	Category	Description
Firefox	Browser	Mozilla Firefox is an open source web browser. It's designed for standards compliance, performance, portability.
Arquillian	Testing	An innovative and highly extensible testing platform. Empowers developers to easily create real, automated tests.

Example 414. Table with column containing AsciiDoc content

```
[cols="2,2,5a"]
|===
|Firefox
|Browser
|Mozilla Firefox is an open source web browser.

It's designed for:
```

```
* standards compliance
* performance
* portability

https://getfirefox.com[Get Firefox]!
|===
```

▼ *View result of Example 414*

Firefox	Browser	Mozilla Firefox is an open source web browser. It's designed for: • standards compliance • performance • portability Get Firefox!

Example 415. Table from CSV data using shorthand

```
,===
Artist,Track,Genre

Baauer,Harlem Shake,Hip Hop
,===
```

▼ *View result of Example 415*

Artist	Track	Genre
Baauer	Harlem Shake	Hip Hop

Example 416. Table from CSV data

```
[%header,format=csv]
|===
Artist,Track,Genre
Baauer,Harlem Shake,Hip Hop
The Lumineers,Ho Hey,Folk Rock
|===
```

▼ *View result of Example 416*

Artist	Track	Genre
Baauer	Harlem Shake	Hip Hop
The Lumineers	Ho Hey	Folk Rock

Example 417. Table from CSV data included from file

```
,===
include::customers.csv[]
,===
```

Example 418. Table from DSV data using shorthand

```
:===
Artist:Track:Genre

Robyn:Indestructible:Dance
:===
```

▼ *View result of Example 418*

Artist	Track	Genre
Robyn	Indestructible	Dance

Example 419. Table with formatted, aligned and merged cells

```
[cols="e,m,^,>s",width="25%"]
|===
|1 >s|2 |3 |4
^|5 2.2+^.^|6 .3+<.>m|7
^|8
|9 2+>|10
|===
```

▼ *View result of Example 419*

1	**2**	3	**4**
5	6		
8			
9		10	7

IDs, roles, and options

Example 420. Shorthand method for assigning block ID (anchor) and role

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

- To specify multiple roles using the shorthand syntax, delimit them by dots.
- The order of `id` and `role` values in the shorthand syntax does not matter.

Example 421. Formal method for assigning block ID (anchor) and role

```
[id="goals",role="incremental"]
* Goal 1
* Goal 2
```

Example 422. Explicit section ID (anchor)

```
[#null-values]
== Primitive types and null values
```

Example 423. Assign ID (anchor) and role to inline formatted text

```
[#id-name.role-name]`monospace text`

[#free-world.goals]*free the world*
```

Example 424. Shorthand method for assigning block options

```
[%header%footer%autowidth]
|===
|Header A |Header B
|Footer A |Footer B
|===
```

Example 425. Formal method for assigning block options

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

// options can be shorted to opts
[opts="header,footer,autowidth"]
|===
|Header A |Header B
|Footer A |Footer B
|===
```

Comments

Example 426. Line and block comments

```
// A single-line comment

////
A multi-line comment.
```

```
Notice it's a delimited block.
////
```

Breaks

Example 427. Thematic break (aka horizontal rule)

```
before

'''

after
```

▼ *View result of Example 427*

before

---

after

Example 428. Page break

```
<<<
```

Attributes and substitutions

Example 429. Attribute declaration and usage

```
:url-home: https://asciidoctor.org
:link-docs: https://asciidoctor.org/docs[documentation]
:summary: AsciiDoc is a mature, plain-text document format for \
       writing notes, articles, documentation, books, and more. \
       It's also a text processor & toolchain for translating \
       documents into various output formats (i.e., backends), \
       including HTML, DocBook, PDF and ePub.
:checkedbox: pass:normal[{startsb}&#10004;{endsb}]

Check out {url-home}[Asciidoctor]!

{summary}

Be sure to read the {link-docs} too!

{checkedbox} That's done!
```

▼ *View result of Example 429*

Check out Asciidoctor!

AsciiDoc is a mature, plain-text document format for writing notes, articles, documentation, books, and more. It's also a text processor & toolchain for translating documents into various output formats (i.e., backends), including HTML, DocBook, PDF and ePub.

Be sure to read the documentation too!

[] That's done!

To learn more about the available attributes and substitution groups see:

- Document Attributes Reference
- Character Replacement Attributes Reference
- Substitution Groups

Example 430. Counter attributes

```
.Parts{counter2:index:0}
|===
|Part Id |Description

|PX-{counter:index}
|Description of PX-{index}

|PX-{counter:index}
|Description of PX-{index}
|===
```

▼ *View result of Example 430*

Table 1. Parts

Part Id	Description
PX-3	Description of PX-3
PX-4	Description of PX-4

Text replacements

Textual symbol replacements

Name	Syntax	Unicode Replacement	Rendered	Notes
Copyright	(C)	©	©	
Registered	(R)	®	®	
Trademark	(TM)	™	™	

Name	Syntax	Unicode Replacement	Rendered	Notes
Em dash	--	—	—	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 ( ). Otherwise, the em dash is followed by a zero-width space () to provide a break opportunity.
Ellipsis	...	…	…	The ellipsis is followed by a zero-width space () to provide a break opportunity.
Single right arrow	->	→	→	
Double right arrow	=>	⇒	⇒	
Single left arrow	<-	←	←	
Double left arrow	<=	⇐	⇐	
Typographic apostrophe	Sam's	Sam’s	Sam’s	The typewriter apostrophe is replaced with the typographic (aka curly or smart) apostrophe.

Any named, numeric or hexadecimal XML character reference is supported.

Escaping substitutions

Example 431. Backslash

```
In /items/\{id}, the id attribute isn't replaced.
The curly braces around it are preserved.

\*Stars* isn't displayed as bold text.
The asterisks around it are preserved.

\&sect; appears as an entity reference.
It's not converted into the section symbol (&#167;).

\=> The backslash prevents the equals sign followed by a greater
than sign from combining to form a double arrow character (=>).

\[[Word]] is not interpreted as an anchor.
The double brackets around it are preserved.

[\[[Word]]] is not interpreted as a bibliography anchor.
The triple brackets around it are preserved.
```

```
\((DD AND CC) OR (DD AND EE)) is not interpreted as a flow index term.
The double brackets around it are preserved.

The URL \https://example.org isn't converted into an active link.
```

▼ *View result of Example 431*

In /items/{id}, the id attribute isn’t replaced. The curly braces around it are preserved.

Stars isn’t displayed as bold text. The asterisks around it are preserved.

§ appears as an entity reference. It’s not converted into the section symbol (§).

=> The backslash prevents the equals sign followed by a greater than sign from combining to form a double arrow character (⇒).

[[syntax-quick-reference:::Word]] is not interpreted as an anchor. The double brackets around it are preserved.

[[[syntax-quick-reference:::Word]]] is not interpreted as a bibliography anchor. The triple brackets around it are preserved.

((DD AND CC) OR (DD AND EE)) is not interpreted as a flow index term. The double brackets around it are preserved.

The URL https://example.org isn’t converted into an active link.

Example 432. Single and double plus inline passthroughs

```
A word or phrase between single pluses, such as +/user/{id}+,
is not substituted.
However, the special characters like +<+ and +>+ are still
escaped in the output.

An attribute reference within a word, such as dev++{conf}++,
is not replaced.

A plus passthrough will escape standalone formatting marks,
like +``+, or formatting marks within a word, like all-natural++*++.
```

▼ *View result of Example 432*

A word or phrase between single pluses, such as /user/{id}, is not substituted. However, the special characters like < and > are still escaped in the output.

An attribute reference within a word, such as dev{conf}, is not replaced.

A plus passthrough will escape standalone formatting marks, like ``, or formatting marks within a word, like all-natural*.

Example 433. Triple plus inline passthrough and inline pass macro

```
+++<del>strike this</del>+++ is marked as deleted.

pass:[<del>strike this</del>] is also marked as deleted.
```

▼ *View result of Example 433*

~~strike this~~ is marked as deleted.

~~strike this~~ is also marked as deleted.

Bibliography

Example 434. 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,gang]]] Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides.
Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. 1994.
```

▼ *View result of Example 434*

The Pragmatic Programmer [syntax-quick-reference:::pp] should be required reading for all developers. To learn all about design patterns, refer to the book by the “Gang of Four” [gang].

References

- [syntax-quick-reference:::pp] Andy Hunt & Dave Thomas. The Pragmatic Programmer: From Journeyman to Master. Addison-Wesley. 1999.
- [gang] Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. 1994.

Footnotes

Example 435. Normal and reusable footnotes

```
A statement.footnote:[Clarification about this statement.]

A bold statement!footnote:disclaimer[Opinions are my own.]

Another bold statement.footnote:disclaimer[]
```

▼ *View result of Example 435*

A statement.[1]

A bold statement![2 - Footnotes]

Another bold statement.[2 - Footnotes]

Markdown compatibility

Markdown compatible syntax is an optional feature of the AsciiDoc language and is currently only available when using Asciidoctor.

Example 436. Markdown-style headings

```
# Document Title (Level 0)

## Section Level 1

### Section Level 2

#### Section Level 3

##### Section Level 4

###### Section Level 5
```

▼ *View result of Example 436*

Document Title (Level 0)

Section Level 1

Section Level 2

Section Level 3

Section Level 4

Section Level 5

Example 437. Fenced code block with syntax highlighting

```
```ruby
require 'sinatra'

get '/hi' do
```
```

````
  "Hello World!"
end
```
````

▼ *View result of Example 437*

```ruby
require 'sinatra'

get '/hi' do
  "Hello World!"
end
```

Example 438. 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
```

▼ *View result of Example 438*

> 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

Example 439. 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.
```

▼ *View result of Example 439*

> > 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.

Example 440. Markdown-style thematic breaks

```
---

- - -

***

* * *
```

▼ *View result of Example 440*

Frequently Asked Questions (FAQ)

Does AsciiDoc only support ASCII text?

No. AsciiDoc provides full Unicode support (UTF-8 by default, UTF-16 with a BOM).

The “Ascii” in AsciiDoc (and Asciidoctor) merely refers to the range of characters used to define the language syntax (e.g., block delimiters, section markers, list markers, attribute list boundaries, built-in attribute and block names, etc.). In other words, you only have to use characters from US-ASCII in order to express the structure of an AsciiDoc document. The content itself, which includes paragraphs, titles, verbatim blocks, attribute names and values, custom block names, and so forth, may contain characters from any character range in Unicode.

An AsciiDoc processor assumes the input is encoded in UTF-8 and it encodes output documents in UTF-8 as well. The include directive allows the encoding to be specified if the include file is not encoded in UTF-8.

What’s the relationship between a converter and a backend?

A **converter** is the software that performs conversion from AsciiDoc to a publishable format. A **backend** is an identifier for the intended output format, and thus tells the AsciiDoc processor

which converter to use. You can think of the backend as an alias for a converter.

The backend represents the user's intent to transform the AsciiDoc document to a given format (e.g., `html5` for HTML 5). That backend also serves as an identifier that tells the processor which converter to use. More than one converter can bind to (i.e., stake claim to) the same backend in order to provide the user with alternatives for generating a given output format. For example, the backend `pdf` could be satisfied by Asciidoctor PDF, but it may also be mapped to a different implementation. The last converter that registers itself with a backend wins.

What's the media type (aka MIME type) for AsciiDoc?

A media type, or MIME type, is a code for identifying file formats and content formats transmitted over the Internet. As of yet, there's no official media type registered for AsciiDoc. However, the AsciiDoc Working Group, which oversees the specification for the AsciiDoc language, has plans to submit a proposal to register an official the media type for AsciiDoc. See asciidoctor#2502.

The proposed media type for AsciiDoc is as follows:

```
name: text/asciidoc
extensions: .adoc, .asciidoc
```

The name `text/asciidoc` follows the convention used for Markdown. The `.adoc` extension is the preferred one. The `.asciidoc` extension is only included for backwards compatibility with existing documents.

See tools.ietf.org/html/rfc7763 for details about naming a media type.

Why is my document attribute being ignored?

If the document attribute is a header-only attribute, make sure it is defined in the document header or passed in via a CLI or API. Otherwise, the document attribute will not have any affect.

Recall that the document header ends at the first empty line or block, whichever comes first. If you have an empty line somewhere in what you intend to be the document header, the attribute entries that fall after that empty line are going to be defined in the body, not the header. That likely explains your problem. If you remove the empty line(s), your attribute should be recognized.

If the document attribute is not a header-only attribute, make sure it is being defined (using an attribute entry) outside of any delimited block and offset from other blocks by at least one empty line.

Part way through the document, the blocks stop rendering correctly. What went wrong?

When content does not display as you expect in the later part of a document, it's usually due to a delimited block missing its closing delimiter line. The parsing rules inside a delimited block are different. If left open, it can impact how the AsciiDoc processor interprets the document structure. For example, the AsciiDoc processor will stop recognizing section titles from that point forward.

To solve this problem, first look for missing delimiter lines. An AsciiDoc processor must warn you when this situation is detected. Syntax highlighting in your text editor can also help with this. Also look at the rendered output to see if the block styles are extending past where you intended.

The most sly culprit is the open block. Although an open block doesn't have any special styling, it does apply delimited block semantics to its contents. You can add a role that applies a custom style, such as a red outline, so you can see its boundaries.

Why don't links to URLs that contain an underscore or caret work?

An AsciiDoc processor applies normal substitutions to paragraph content, including the target of a URL or link macro. It's up to the author to escape this syntax. See Troubleshooting Complex URLs to find techniques you can use to address this problem.

Compare AsciiDoc 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*. While that's certainly the goal of both AsciiDoc and Markdown, and both are very approachable for newcomers, this page explores why AsciiDoc is a more suitable alternative to Markdown as your content grows and evolves.

Starting with Markdown

The most prevalent lightweight markup language is Markdown. (At least, Markdown is what you call it at first). The main advantage of Markdown lies in its primitive syntax: its manual and cheat-sheet 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.

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.

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)	`**bold**`	`*bold*`
Bold (unconstrained)	`**b**old`	`**b**old`
Italic (constrained)	`*italic*`	`_italic_`
Italic (unconstrained)	*n/a*	`__i__talic`
Monospace (constrained)	`` `monospace` ``	`` `monospace` ``
Monospace (unconstrained)	`` `m`onospace ``	``` ``m``onospace ```
Literal monospace	`` `http://localhost:8080` `/issue/{id}` ``	`` `+http://localhost:8080+` `+/issue/{id}+` ``
Link with label	`[Asciidoctor](https://asciidoctor.org)`	`https://asciidoctor.org[Asciidoctor]`

Language Feature	Markdown	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 [Usage](#_usage).` `<h2 id="_usage">Usage</h2>`	`See <<_usage>>.` `== Usage`
Block ID (aka anchor)	`<h2 id="usage">Usage</h2>`	`[#usage]` `== Usage`
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.` `____`

Language Feature	Markdown	AsciiDoc
Literal block	<pre> $ gem install asciidoctor</pre>	*Example 441. Indented (by 1 or more spaces)* <pre> $ gem install asciidoctor</pre> *Example 442. Delimited* <pre>.... $ gem install asciidoctor ....</pre>
Code block*	<pre>```java public class Person { private String name; public Person(String name) { this.name = name; } } ```</pre>	<pre>[source,java] ---- public class Person { private String name; public Person(String name) { this.name = name; } } ----</pre>
Unordered list	<pre>* apples * orange * temple * navel * bananas</pre>	<pre>* apples * oranges ** temple ** navel * bananas</pre>
Ordered list	<pre>1. first 2. second 3. third</pre>	<pre>. first . second . third</pre>
Thematic break (aka horizontal rule)*	<pre>*** * * * --- - - - ___ _ _ _</pre>	<pre>'''</pre>

Language Feature	Markdown	AsciiDoc
Typographic quotes (aka “smart quotes”)	Enabled through an extension switch, but offer little control in how they are applied.	``` The ``'90s popularized a new form of music known as "`grunge`" rock. It'll turn out to have an impact that extended well beyond music. ```
Document header	*Example 443. Slapped on as “front matter”* ``` --- layout: docs title: Writing posts prev_section: defining-frontmatter next_section: creating-pages permalink: /docs/writing-posts/ --- ```	*Example 444. Native support!* ``` = Writing posts :page-layout: base :showtitle: :prev_section: defining-frontmatter :next_section: creating-pages ```
Admonitions	*n/a*	``` TIP: You can add line numbers to source listings by adding the word `numbered` in the attribute list after the language name. ```
Sidebars	*n/a*	``` .Lightweight Markup **** Writing languages that let you type less and express more. **** ```
Block titles	*n/a*	``` .Grocery list * Milk * Eggs * Bread ```
Includes	*n/a*	``` include::intro.adoc[] ```
URI reference	``` Go to the [home page][home]. [home]: https://example.org ```	``` :home: https://example.org Go to the {home}[home page]. ```

Language Feature	Markdown	AsciiDoc
Custom CSS classes	*n/a*	`[.path]_Gemfile_`

* Asciidoctor 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 some of the common parts of the Markdown syntax, like headings, blockquotes and fenced code blocks, simplifying the migration from Markdown to AsciiDoc. For details, see Markdown compatibility.

Document Attributes Reference

Document attributes are used either to configure behavior in the processor or to relay information about the document and its environment. This page catalogs all the built-in (i.e., reserved) document attributes in AsciiDoc.

Unless otherwise marked, these attributes can be modified (set or unset) from the API using the `:attributes` option, from the CLI using the `-a` option, or in the document (often in the document header) using an attribute entry.

Use the follow legend to understand the columns and values in the tables on this page.

Set By Default — The attribute is automatically set and assigned a default value by the AsciiDoc processor. The default value is indicated in **bold** (e.g., `skip`).

Allowable Values — Allowable values for the attribute. Numeric values and values shown in *italic* are instructional and indicate a value type (e.g., *any*, *empty*, *number*, 1–5, etc.).

- *any* — Any value is accepted.
- *empty* — Indicates the attribute doesn't require an explicit value. The

attribute is simply turned on by being set.

- *empty*[=`effective`] — In some cases, an empty value is interpreted by the processor as one of the allowable non-empty values. This effective value is prefixed with an equals sign and enclosed in square brackets (e.g., [=`auto`]). An attribute reference will resolve to an empty value rather than the effective value.
- (`implied`) — Built-in attributes that are not set may have an implied value. The implied value is enclosed in parentheses (e.g., (`attributes`)). An implied value can't be resolved using an attribute reference.

If the attribute doesn't accept *any* or *empty*, than you must only assign one of the allowable values or specified value type.

Header Only — The attribute must be set or unset by the end of the document header (i.e., set by the API, CLI, or in the document header). Otherwise, the assignment won't have any effect on the document. If an attribute is not marked as *Header Only*, it can be set anywhere in the document, assuming the attribute is not locked by the API or CLI. However, changing an attribute only affects behavior for content that follows the assignment (in document order).

Intrinsic attributes

Intrinsic attributes are set automatically by the processor. These attributes provide information about the document being processed (e.g., `docfile`), the security mode under which the processor is running (e.g., `safe-mode-name`), and information about the user's environment (e.g., `user-home`).

Many of these attributes are read only, which means they can't be modified (with some exceptions). Attributes which are not are marked as modifiable. Attributes marked as both modifiable and API/CLI Only can only be set from the API or CLI. All other attributes marked as modifiable must be set by the end of the header (i.e., Header Only).

Name	Allowable Values	Modifiable	API/CLI Only	Notes
`backend`	*any* *ex.* html5	Yes	No	The backend used to select and activate the converter that creates the output file. Usually named according to the output format (e.g., `html5`)

Name	Allowable Values	Modifiable	API/CLI Only	Notes
backend-<backend>	*empty*	No	n/a	A convenience attribute for checking which backend is selected. <backend> is the value of the backend attribute (e.g., backend-html5). Only one such attribute is set at time.
basebackend	*any* *ex.* html	No	n/a	The generic backend on which the backend is based. Typically derived from the backend value minus trailing numbers (e.g., the basebackend for docbook5 is docbook). May also indicate the internal backend employed by the converter (e.g., the basebackend for pdf is html).
basebackend-<basebackend>	*empty*	No	n/a	A convenience attribute for checking which basebackend is active. <basebackend> is the value of the basebackend attribute (e.g., basebackend-html). Only one such attribute is set at time.
docdate	*date (ISO)* *ex.* 2019-01-04	Yes	No	Last modified date of the source document.[1,2]
docdatetime	*datetime (ISO)* *ex.* 2019-01-04 19:26:06 UTC	Yes	No	Last modified date and time of the source document.[1,2]
docdir	*directory path* *ex.* /home/user/docs	If input is a string	Yes	Full path of the directory that contains the source document. Empty if the safe mode is SERVER or SECURE (to conceal the file's location).
docfile	*file path* *ex.* /home/user/docs/userguide.adoc	If input is a string	Yes	Full path of the source document. Truncated to the basename if the safe mode is SERVER or SECURE (to conceal the file's location).

Name	Allowable Values	Modifiable	API/CLI Only	Notes
`docfilesuffix`	*file extension* *ex.* .adoc	If input is a string	Yes	File extension of the source document, including the leading period.
`docname`	*file stem basename* *ex.* userguide	If input is a string	Yes	Root name of the source document (no leading path or file extension).
`doctime`	*time (ISO)* *ex.* 19:26:06 UTC	Yes	No	Last modified time of the source document.[1,2]
`doctype-<doctype>`	*empty*	No	n/a	A convenience attribute for checking the doctype of the document. <doctype> is the value of the `doctype` attribute (e.g., `doctype-book`). Only one such attribute is set at time.
`docyear`	*integer* *ex.* 2025	Yes	No	Year that the document was last modified.[1,2]
`embedded`	*empty*	No	n/a	Only set if content is being converted to an embedded document (i.e., body of document only).
`filetype`	*any* *ex.* html	If input is a string	Yes	File extension of the output file name (without leading period).
`filetype-<filetype>`	*empty*	No	n/a	A convenience attribute for checking the filetype of the output. <filetype> is the value of the `filetype` attribute (e.g., `filetype-html`). Only one such attribute is set at time.
`htmlsyntax`	`html` `xml`	No	n/a	Syntax used when generating the HTML output. Controlled by and derived from the backend name (html=html or xhtml=html).
`localdate`	*date (ISO)* *ex.* 2019-02-17	Yes	No	Date when the document was converted.[2]
`localdatetime`	*datetime (ISO)* *ex.* 2019-02-17 19:31:05 UTC	Yes	No	Date and time when the document was converted.[2]

Name	Allowable Values	Modifiable	API/CLI Only	Notes
`localtime`	*time (ISO)* *ex.* 19:31:05 UTC	Yes	No	Time when the document was converted.[2]
`localyear`	*integer* *ex.* 2025	Yes	No	Year when the document was converted.[2]
`outdir`	*directory path* *ex.* /home/user/docs/dist	No	n/a	Full path of the output directory. (Cannot be referenced in the content. Only available to the API once the document is converted).
`outfile`	*file path* *ex.* /home/user/docs/dist/userguide.html	No	n/a	Full path of the output file. (Cannot be referenced in the content. Only available to the API once the document is converted).
`outfilesuffix`	*file extension* *ex.* .html	Yes	No	File extension of the output file (starting with a period) as determined by the backend (`.html` for `html`, `.xml` for `docbook`, etc.).
`safe-mode-level`	`0` `1` `10` `20`	No	n/a	Numeric value of the safe mode setting. (0=UNSAFE, 1=SAFE, 10=SERVER, 20=SECURE).
`safe-mode-name`	`UNSAFE` `SAFE` `SERVER` `SECURE`	No	n/a	Textual value of the safe mode setting.
`safe-mode-unsafe`	*empty*	No	n/a	Set if the safe mode is UNSAFE.
`safe-mode-safe`	*empty*	No	n/a	Set if the safe mode is SAFE.
`safe-mode-server`	*empty*	No	n/a	Set if the safe mode is SERVER.
`safe-mode-secure`	*empty*	No	n/a	Set if the safe mode is SECURE.
`user-home`	*directory path* *ex.* /home/user	No	n/a	Full path of the home directory for the current user. Masked as `.` if the safe mode is SERVER or SECURE.

[1] Only reflects the last modified time of the source document file. It does not consider the last modified time of files which are included.

[2] 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 the source date epoch specification 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.

Compliance attributes

Name	Allowable Values	Set By Default	Header Only	Notes
`attribute-missing`	`drop` `drop-line` **`skip`** `warn`	Yes	No	Controls how missing attribute references are handled.
`attribute-undefined`	`drop` **`drop-line`**	Yes	No	Controls how attribute unassignments are handled.
`compat-mode`	*empty*	No	No	Controls when the legacy parsing mode is used to parse the document.
`experimental`	*empty*	No	Yes	Enables *Button and Menu UI Macros* and the *Keyboard Macro*.
`reproducible`	*empty*	No	Yes	Prevents last-updated date from being added to HTML footer or DocBook info element. Useful for storing the output in a source code control system as it prevents spurious changes every time you convert the document. Alternately, you can use the SOURCE_DATE_EPOCH environment variable, which sets the epoch of all source documents and the local datetime to a fixed value.
`skip-front-matter`	*empty*	No	Yes	Consume YAML-style front-matter at top of document and store it in `front-matter` attribute.

Localization and numbering attributes

Name	Allowable Values	Set By Default	Header Only	Notes
appendix-caption	*any* Appendix	Yes	No	Label added before an appendix title.
appendix-number	*character* (@)	No	No	Sets the seed value for the appendix number sequence.[1]
appendix-refsig	*any* Appendix	Yes	No	Signifier added to Appendix title cross references.
caution-caption	*any* Caution	Yes	No	Text used to label CAUTION admonitions when icons aren't enabled.
chapter-number	*number* (0)	No	No	Sets the seed value for the chapter number sequence.[1] *Book doctype only*.
chapter-refsig	*any* Chapter	Yes	No	Signifier added to Chapter titles in cross references. *Book doctype only*.
chapter-signifier	*any*	No	No	Label added to level 1 section titles (chapters). *Book doctype only*.
example-caption	*any* Example	Yes	No	Text used to label example blocks.
example-number	*number* (0)	No	No	Sets the seed value for the example number sequence.[1]
figure-caption	*any* Figure	Yes	No	Text used to label images and figures.
figure-number	*number* (0)	No	No	Sets the seed value for the figure number sequence.[1]
footnote-number	*number* (0)	No	No	Sets the seed value for the footnote number sequence.[1]
important-caption	*any* Important	Yes	No	Text used to label IMPORTANT admonitions when icons are not enabled.

Name	Allowable Values	Set By Default	Header Only	Notes
`lang`	*BCP 47 language tag* (`en`)	No	Yes	Language tag specified on document element of the output document. Refer to the lang and xml:lang attributes section of the HTML specification to learn about the acceptable values for this attribute.
`last-update-label`	*any* `Last updated`	Yes	Yes	Text used for "Last updated" label in footer.
`listing-caption`	*any*	No	No	Text used to label listing blocks.
`listing-number`	*number* (`0`)	No	No	Sets the seed value for the listing number sequence.[1]
`manname-title`	*any* (`Name`)	No	Yes	Label for program name section in the man page.
`nolang`	*empty*	No	Yes	Prevents `lang` attribute from being added to root element of the output document.
`note-caption`	*any* `Note`	Yes	No	Text used to label NOTE admonitions when icons aren't enabled.
`part-refsig`	*any* `Part`	Yes	No	Signifier added to Part titles in cross references. *Book doctype only*.
`part-signifier`	*any*	No	No	Label added to level 0 section titles (parts). *Book doctype only*.
`preface-title`	*any*	No	No	Title text for an anonymous preface when `doctype` is `book`.
`section-refsig`	*any* `Section`	Yes	No	Signifier added to title of numbered sections in cross reference text.
`table-caption`	*any* `Table`	Yes	No	Text of label prefixed to table titles.
`table-number`	*number* (`0`)	No	No	Sets the seed value for the table number sequence.[1]

Name	Allowable Values	Set By Default	Header Only	Notes
`tip-caption`	*any* `Tip`	Yes	No	Text used to label TIP admonitions when icons aren't enabled.
`toc-title`	*any* `Table of Contents`	Yes	Yes	Title for table of contents.
`untitled-label`	*any* `Untitled`	Yes	Yes	Default document title if document doesn't have a document title.
`version-label`	*any* `Version`	Yes	Yes	See Version Label Attribute.
`warning-caption`	*any* `Warning`	Yes	No	Text used to label WARNING admonitions when icons aren't enabled.

Document metadata attributes

Name	Allowable Values	Set By Default	Header Only	Notes
`app-name`	*any*	No	Yes	Adds `application-name` meta element for mobile devices inside HTML document head.
`author`	*any*	Extracted from author info line	Yes	Can be set automatically via the author info line or explicitly. See Author Information.
`authorinitials`	*any*	Extracted from `author` attribute	Yes	Derived from the author attribute by default. See Author Information.
`authors`	*any*	Extracted from author info line	Yes	Can be set automatically via the author info line or explicitly as a comma-separated value list. See Author Information.
`copyright`	*any*	No	Yes	Adds `copyright` meta element in HTML document head.

Name	Allowable Values	Set By Default	Header Only	Notes
`doctitle`	*any*	Yes, if document has a doctitle	Yes	See doctitle attribute.
`description`	*any*	No	Yes	Adds description meta element in HTML document head.
`email`	*any*	Extracted from author info line	Yes	Can be any inline macro, such as a URL. See Author Information.
`firstname`	*any*	Extracted from author info line	Yes	See Author Information.
`front-matter`	*any*	Yes, if front matter is captured	n/a	If `skip-front-matter` is set via the API or CLI, any YAML-style frontmatter skimmed from the top of the document is stored in this attribute.
`keywords`	*any*	No	Yes	Adds keywords meta element in HTML document head.
`lastname`	*any*	Extracted from author info line	Yes	See Author Information.
`middlename`	*any*	Extracted from author info line	Yes	See Author Information.
`orgname`	*any*	No	Yes	Adds `<orgname>` element value to DocBook info element.
`revdate`	*any*	Extracted from revision info line	Yes	See Revision Information.

Name	Allowable Values	Set By Default	Header Only	Notes
`revremark`	*any*	Extracted from revision info line	Yes	See Revision Information.
`revnumber`	*any*	Extracted from revision info line	Yes	See Revision Information.
`title`	*any*	No	Yes	Value of `<title>` element in HTML `<head>` or main DocBook `<info>` of output document. Used as a fallback when the document title is not specified. See title attribute.

Section title and table of contents attributes

Name	Allowable Values	Set By Default	Header Only	Notes
`idprefix`	*valid XML ID start character* `_`	Yes	No	Prefix of auto-generated section IDs. See Change the ID prefix.
`idseparator`	*valid XML ID character* `_`	Yes	No	Word separator used in auto-generated section IDs. See Change the ID word separator.
`leveloffset`	[+-]0–5	No	No	Increases or decreases level of headings below assignment. A leading + or - makes the value relative.
`partnums`	*empty*	No	No	Enables numbering of parts. See Number book parts. *Book doctype only*.
`sectanchors`	*empty*	No	No	Adds anchor in front of section title on mouse cursor hover.
`sectids`	***empty***	Yes	No	Generates and assigns an ID to any section that does not have an ID. See Disable automatic ID generation.

Name	Allowable Values	Set By Default	Header Only	Notes
`sectlinks`	*empty*	No	No	Turns section titles into self-referencing links.
`sectnums`	*empty* `all`	No	No	Numbers sections to depth specified by `sectnumlevels`.
`sectnumlevels`	0–5 (3)	No	No	Controls depth of section numbering.
`title-separator`	*any*	No	Yes	Character used to separate document title and subtitle.
`toc`	*empty*[=`auto`] `auto` `left` `right` `macro` `preamble`	No	Yes	Turns on table of contents and specifies its location.
`toclevels`	0–5 (2)	No	Yes	Maximum section level to display.
`fragment`	*empty*	No	Yes	Informs parser that document is a fragment and that it shouldn't enforce proper section nesting.

General content and formatting attributes

Name	Allowable Values	Set By Default	Header Only	Notes
`asset-uri-scheme`	*empty* `http` (`https`)	No	Yes	Controls protocol used for assets hosted on a CDN.
`cache-uri`	*empty*	No	Yes	Cache content read from URIs.
`data-uri`	*empty*	No	Yes	Embed graphics as data-uri elements in HTML elements so file is completely self-contained.
`docinfo`	*empty*[=`private`] `shared` `private` `shared-head` `private-head` `shared-footer` `private-footer`	No	Yes	Read input from one or more DocBook info files.

Name	Allowable Values	Set By Default	Header Only	Notes
docinfodir	*directory path*	No	Yes	Location of docinfo files. Defaults to directory of source file if not specified.
docinfosubs	*comma-separated substitution names* (attributes)	No	Yes	AsciiDoc substitutions that are applied to docinfo content.
doctype	**article** book inline manpage	Yes	Yes	Output document type.
eqnums	*empty*[=AMS] AMS all none	No	Yes	Controls automatic equation numbering on LaTeX blocks in HTML output (MathJax). If the value is AMS, only LaTeX content enclosed in an \begin{equation}...\end{equation} container will be numbered. If the value is all, then all LaTeX blocks will be numbered. See equation numbering in MathJax.
hardbreaks-option	*empty*	No	No	Preserve hard line breaks.
hide-uri-scheme	*empty*	No	No	Hides URI scheme for raw links.
media	prepress print (screen)	No	Yes	Specifies media type of output and enables behavior specific to that media type. *PDF converter only*.
nofooter	*empty*	No	Yes	Turns off footer.
nofootnotes	*empty*	No	Yes	Turns off footnotes.
noheader	*empty*	No	Yes	Turns off header.
notitle	*empty*	No	Yes	Hides the doctitle in an embedded document. Mutually exclusive with the showtitle attribute.
outfilesuffix	***file extension*** *ex.* .html	Yes	Yes	File extension of output file, including dot (.), such as .html.

Name	Allowable Values	Set By Default	Header Only	Notes
pagewidth	*integer* (425)	No	Yes	Page width used to calculate the absolute width of tables in the DocBook output.
relfileprefix	*empty* *path segment*	No	No	The path prefix to add to relative xrefs.
relfilesuffix	***file extension*** *path segment* *ex.* .html	Yes	No	The path suffix (e.g., file extension) to add to relative xrefs. Defaults to the value of the outfilesuffix attribute. (Preferred over modifying outfilesuffix).
show-link-uri	*empty*	No	No	Prints the URI of a link after the link text. *PDF converter only*.
showtitle	*empty*	No	Yes	Displays the doctitle in an embedded document. Mutually exclusive with the notitle attribute.
stem	empty[=asciimath] asciimath latexmath	No	Yes	Enables mathematics processing and interpreter.
table-frame	(all) ends sides none	No	No	Controls default value for frame attribute on tables.
table-grid	(all) cols rows none	No	No	Controls default value for grid attribute on tables.
table-stripes	(none) even odd hover all	No	No	Controls default value for stripes attribute on tables.
tabsize	*integer* (≥ 0)	No	No	Converts tabs to spaces in verbatim content blocks (e.g., listing, literal).

Name	Allowable Values	Set By Default	Header Only	Notes
webfonts	***empty***	Yes	Yes	Control whether webfonts are loaded when using the default stylesheet. When set to empty, uses the default font collection from Google Fonts. A non-empty value replaces the family query string parameter in the Google Fonts URL.
xrefstyle	full short basic	No	No	Formatting style to apply to cross reference text.

Image and icon attributes

Name	Allowable Values	Set By Default	Header Only	Notes
iconfont-cdn	*url* (default CDN URL)	No	Yes	If not specified, uses the cdnjs.com service. Overrides CDN used to link to the Font Awesome stylesheet.
iconfont-name	*any* (font-awesome)	No	Yes	Overrides the name of the icon font stylesheet.
iconfont-remote	***empty***	Yes	Yes	Allows use of a CDN for resolving the icon font. Only relevant used when value of icons attribute is font.
icons	*empty*[=image] image font	No	Yes	Chooses images or font icons instead of text for admonitions. Any other value is assumed to be an icontype and sets the value to empty (image-based icons).
iconsdir	*directory path* *url* *ex.* ./images/icons	Yes	No	Location of non-font-based image icons. Defaults to the *icons* folder under imagesdir if imagesdir is specified and iconsdir is not specified.
icontype	jpg (png) gif svg	No	No	File type for image icons. Only relevant when using image-based icons.

Name	Allowable Values	Set By Default	Header Only	Notes
`imagesdir`	***empty*** *directory path* *url*	Yes	No	Location of image files.

Source highlighting and formatting attributes

Name	Allowable Values	Set By Default	Header Only	Notes
`coderay-css`	(`class`) `style`	No	Yes	Controls whether CodeRay uses CSS classes or inline styles.
`coderay-linenums-mode`	`inline` (`table`)	No	No	Sets how CodeRay inserts line numbers into source listings.
`coderay-unavailable`	*empty*	No	Yes	Instructs processor not to load CodeRay. Also set if processor fails to load CodeRay.
`highlightjsdir`	*directory path* *url* (default CDN URL)	No	Yes	Location of the highlight.js source code highlighter library.
`highlightjs-theme`	*highlight.js style name* (`github`)	No	Yes	Name of theme used by highlight.js.
`prettifydir`	*directory path* *url* (default CDN URL)	No	Yes	Location of non-CDN prettify source code highlighter library.
`prettify-theme`	*prettify style name* (`prettify`)	No	Yes	Name of theme used by prettify.
`prewrap`	***empty***	Yes	No	Wrap wide code listings.
`pygments-css`	(`class`) `style`	No	Yes	Controls whether Pygments uses CSS classes or inline styles.
`pygments-linenums-mode`	(`table`) `inline`	No	No	Sets how Pygments inserts line numbers into source listings.
`pygments-style`	*Pygments style name* (`default`)	No	Yes	Name of style used by Pygments.

Name	Allowable Values	Set By Default	Header Only	Notes
`pygments-unavailable`	*empty*	No	Yes	Instructs processor not to load Pygments. Also set if processor fails to load Pygments.
`rouge-css`	(`class`) `style`	No	Yes	Controls whether Rouge uses CSS classes or inline styles.
`rouge-linenums-mode`	`inline` (`table`)	No	No	Sets how Rouge inserts line numbers into source listings. *`inline` not yet supported by Asciidoctor*. See asciidoctor#3641.
`rouge-style`	*Rouge style name* (`github`)	No	Yes	Name of style used by Rouge.
`rouge-unavailable`	*empty*	No	Yes	Instructs processor not to load Rouge. Also set if processor fails to load Rouge.
`source-highlighter`	`coderay` `highlight.js` `pygments` `rouge`	No	Yes	Specifies source code highlighter. Any other value is permitted, but must be supported by a custom syntax highlighter adapter.
`source-indent`	*integer*	No	No	Normalize block indentation in source code listings.
`source-language`	*source code language name*	No	No	Default language for source code blocks.
`source-linenums-option`	*empty*	No	No	Turns on line numbers for source code listings.

HTML styling attributes

Name	Allowable Values	Set By Default	Header Only	Notes
`copycss`	***empty*** *file path*	Yes	Yes	Copy CSS files to output. Only relevant when the `linkcss` attribute is set.

Name	Allowable Values	Set By Default	Header Only	Notes
`css-signature`	*valid XML ID*	No	Yes	Assign value to `id` attribute of HTML `<body>` element. **Preferred approach is to assign an ID to document title**.
`linkcss`	*empty*	No	Yes	Links to stylesheet instead of embedding it. Can't be unset in SECURE mode.
`max-width`	CSS length (e.g. 55em, 12cm, etc)	No	Yes	Constrains maximum width of document body. **Not recommended. Use CSS stylesheet instead.**
`stylesdir`	*directory path* *url* `.`	Yes	Yes	Location of CSS stylesheets.
`stylesheet`	***empty*** *file path*	Yes	Yes	CSS stylesheet file name. An empty value tells the converter to use the default stylesheet.
`toc-class`	*valid CSS class name* (`toc`) or (`toc2`) if toc=left	No	Yes	CSS class on the table of contents container.

Manpage attributes

The attribute in this section are only relevant when using the manpage doctype and/or backend.

Name	Allowable Values	Set By Default	Header Only	Notes
`mantitle`	*any*	Based on content.	Yes	Metadata for man page output.
`manvolnum`	*any*	Based on content.	Yes	Metadata for man page output.
`manname`	*any*	Based on content.	Yes	Metadata for man page output.

Name	Allowable Values	Set By Default	Header Only	Notes
`manpurpose`	*any*	Based on content	Yes	Metadata for man page output.
`man-linkstyle`	*link format pattern* (`blue R <>`)	No	Yes	Link style in man page output.
`mansource`	*any*	No	Yes	Source (e.g., application and version) the man page describes.
`manmanual`	*any*	No	Yes	Manual name displayed in the man page footer.

Security attributes

Since these attributes deal with security, they can only be set from the API or CLI.

Name	Allowable Values	Set By Default	API/CLI Only	Notes
`allow-uri-read`	*empty*	No	Yes	Allows data to be read from URLs.
`max-attribute-value-size`	*integer* (≥ 0) `4096`	If safe mode is SECURE	Yes	Limits maximum size (in bytes) of a resolved attribute value. Default value is only set in SECURE mode. Since attributes can reference attributes, it's possible to create an output document disproportionately larger than the input document without this limit in place.
`max-include-depth`	*integer* (≥ 0) `64`	Yes	Yes	Curtail infinite include loops and to limit the opportunity to exploit nested includes to compound the size of the output document.

[1] The `-number` attributes are created and managed automatically by the AsciiDoc processor 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 Asciidoctor, setting the `-number` attributes will influence the next number used for subsequent numbered blocks of that type. However, you should not rely on this behavior as it is subject to change in future revisions of the language.

Character Replacement Attributes Reference

This page identifies built-in document attributes populated by the AsciiDoc processor that are geared towards character replacement.

This category of attributes provide portable replacements for common typographical marks (e.g., smart quotes and symbols) and non-visible characters (e.g., empty and no break space), an escaping mechanism for characters which have special meaning in AsciiDoc (e.g., plus and colon), and passthroughs for characters which get encoded by default (e.g., less than and greater than). Like all document attributes, you can insert the value of any one of these attributes in your content using an attribute reference (e.g., {nbsp}).

The AsciiDoc processor does not prevent you from reassigning these predefined attributes. However, you're encouraged to treat them as read-only. Only a converter should override these attributes if the output format requires the use of a different encoding scheme.

Built-in document attributes for character replacement

Attribute name	Replacement text	Appearance
blank[1]	*nothing*	
empty	*nothing*	
sp	*space*	
nbsp		
zwsp[2]		
wj[3]		
apos	'	'
quot	"	"
lsquo	‘	‘
rsquo	’	’
ldquo	“	“
rdquo	”	”
deg	°	°
plus	+	+
brvbar	¦	¦
vbar	|	|
amp	&	&
lt	<	<
gt	>	>
startsb	[	[

Attribute name	Replacement text	Appearance
endsb	]	]
caret	^	^
asterisk	*	*
tilde	~	~
backslash	\	\
backtick	`	`
two-colons	::	::
two-semicolons	;;	;;
cpp (deprecated)	C++	C++
cxx	C++	C++
pp	++	++

[1] An alias for the attribute empty, for those who find this terminology clearer.

[2] The Zero Width Space (ZWSP) is a code point in Unicode that shows where a long word can be split if necessary.

[3] The word joiner (WJ) is a code point in Unicode that prevents a line break at its position.

Notice that some replacement values are Unicode characters, whereas others are numeric character references (e.g., "). The numeric character reference is when the Unicode character could interfere with the AsciiDoc syntax. In this case, it's the responsibility of the converter to transform that numeric character reference into a format that is compatible with the output format. For example, in the man page converter, each character reference is replaced with a troff macro.

Thus, the abstraction of using AsciiDoc attributes for character replacements not only gives the author control over how the document is interpreted, it also helps decouple content and presentation. In other words, it's more portable to use an attribute reference in the content rather than hardcode a numeric character reference.

Glossary of Terms

This glossary is a work in progress. It does not include all the terms in AsciiDoc.

attribute reference

an expression for dereferencing the value of a document attribute.

attrlist

the source text that defines attributes for an element (i.e., block, block macro, inline macro) or an include directive.

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 element

a line-oriented chunk of content in an AsciiDoc document.

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.

block name

used to refer to custom blocks, which can map an arbitrary name to one or more contexts; has a similar role as the style for a built-in block, such as an admonition block

block style

a modifier that specializes the context of a block

built-in attribute

a document attribute that controls processing, integrations, styling, and localization.

content model

determines how the content of a block is parsed and processed (e.g., simple, compound, verbatim, raw, etc.)

context

the element's type; describes the primary function of an element (e.g., sidebar, listing, example)

converter

a software component that an AsciiDoc processor calls to convert a parsed AsciiDoc document to a given output format; the converter and output format are correlated using a backend identifier.

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.

element

discrete content in the source or output document. May be a branch (contains child elements) or a leaf (does not contain child elements).

element attribute

an attribute associated with a block, macro, formatted text, or the include directive; these attributes can affect processing of the element (or include directive) and are available in the document model; however, the value of these attributes cannot be resolved using an attribute reference.

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.

inline element

a phrase (i.e., span of content) within a block element or one of its attributes in an AsciiDoc document.

list continuation

a plus sign (+) on a line by itself that connects adjacent lines of text to a list item.

macro

a syntax for representing non-text elements or that expands into text using the provided metadata.

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.

node

an in-memory representation of a block or inline element in the parsed document model.

predefined attribute

a document attribute defined for convenience; often used for inserting special content characters.

structural container

the fixed set of reusable block enclosures (delimited regions) defined by the AsciiDoc language; implies the block's content model; characterized by a pair of matching delimited lines defining the boundaries of the block's content

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.

[1] Clarification about this statement.