Asciidoctor User Manual

Sarah White
@carbonfray (https://twitter.com/carbonfray)
Dan Allen
@mojavelinux (https://github.com/mojavelinux)

Table of Contents

This document is under active development and discussion!

If you find errors or omissions in this document, please don't hesitate to submit an issue or open a pull request (https://github.com/asciidoctor/asciidoctor.org/issues) with a fix. We also encourage you to ask questions and discuss any aspects of the project on the mailing list (http://discuss.asciidoctor.org) or IRC. New contributors are always welcome!

This manual assumes you are using Asciidoctor to produce and render your document. Asciidoctor implements more syntax, attributes and functions than the legacy AsciiDoc.py processor. Appendix A lists which features are available to the Asciidoctor and AsciiDoc processors.

Introduction to Asciidoctor

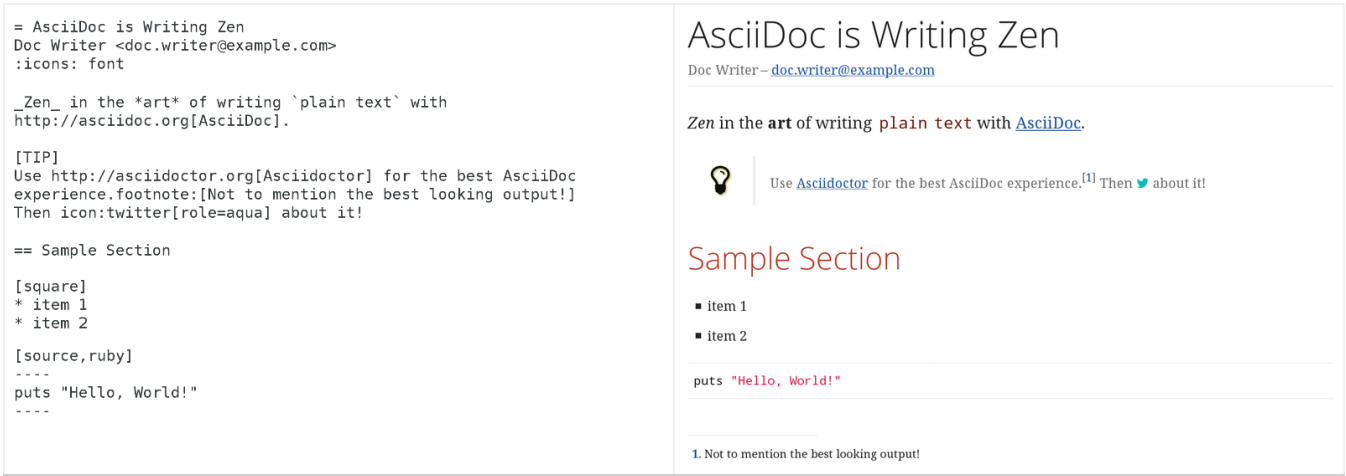
Section Pending

1. What is Asciidoctor?

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

1.1. The Big Picture

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



AsciiDoc source | Rendered HTML

1.2. Asciidoctor on the JVM

You can run Asciidoctor on the JVM using JRuby. You can also use AsciidoctorJ (https://github.com/asciidoctor/asciidoctorj) to invoke Asciidoctor's APIs from Java and other JVM languages.

1.3. Asciidoctor.js

Asciidoctor can be used in JavaScript. Opal (http://opalrb.org) is used to transcompile the code from Ruby to JavaScript to make Asciidoctor.js (https://github.com/asciidoctor/asciidoctor.js), which can be used whereever JavaScript runs, such as in a web browser or on Node.js.

1.4. Asciidoctor's most notable benefits

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

Built-in and custom templates

Asciidoctor uses a set of built-in ERB templates to generate HTML 5 and DocBook output that is structurally equivalent to what AsciiDoc produces. Any of these templates can be replaced by a custom template written in any template language available in the Ruby ecosystem. Custom template rendering is handled by the Tilt (https://github.com/rtomayko/tilt) template abstraction library. Tilt is one of the most popular gems in the Ruby ecosystem.

Parser and object model

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

Performance and security

No coverage of Asciidoctor would be complete without mention of its *speed*. Despite not being an original goal of the project, Asciidoctor has proven startlingly fast. It loads, parses and renders documents **25 times as fast** as the Python implementation. That's good news for developer productivity and good news for GitHub or any server-side application that needs to render AsciiDoc markup. Asciidoctor also offers several levels of security, further justifying its suitability for server-side deployments.

Beyond Ruby

Asciidoctor's usage is not limited to the Ruby community. Thanks to JRuby (http://jruby.org), a port of Ruby to the JVM, Asciidoctor can be used inside Java applications as well. Plugins are available for Apache Maven (https://github.com/asciidoctor/asciidoctor-maven-plugin), Gradle (https://github.com/asciidoctor/asciidoctor-gradle-plugin), and Rewrite (https://github.com/ocpsoft/rewrite/tree/master/transform-markup). These plugins are based on the AsciidoctorJ (https://github.com/asciidoctor/asciidoctorj) for Asciidoctor.

Asciidoctor also ships with a command line interface (CLI). The Asciidoctor CLI, asciidoctor (http://asciidoctor.org/man/asciidoctor), is a drop-in replacement for the `asciidoc.py` script from the AsciiDoc Python distribution.

1.4.1. AsciiDoc syntax processing

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

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

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

1.5. Compared to AsciiDoc

Section Pending

1.6. Compared to MarkDown

Section Pending

Quick Starts

Section Pending

2. Installation Quick Start

Section Pending

3. Usage Quick Start

Section Pending

3.1. Using the Command Line Interface

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

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

```
$ asciidoctor --version
```

The following information should be output in your terminal:

```
Asciidoctor 0.1.4 [http://asciidoctor.org]
```

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

```
$ asciidoctor <asciidoc_file>
```

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

There are many other options available and full help is provided via:

```
$ asciidoctor --help
```

or in the man page (http://asciidoctor.org/man/asciidoctor).

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

3.2. Using the Ruby API

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

Asciidoctor also has a Java API that mirrors the Ruby API. The Java API calls through to the Ruby API using an embedded JRuby runtime. See the Asciidoctor Java integration project (http://asciidoctor.org/docs/install-and-use-asciidoctor-java-integration) for more information.

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

```ruby
require 'asciidoctor'
```

With that in place, you can start processing AsciiDoc documents. To parse a file into an `Asciidoctor::Document` object:

```ruby
doc = Asciidoctor.load_file 'sample.adoc'
```

You can get information about the document:

```ruby
puts doc.doctitle
puts doc.attributes
```

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

```ruby
Asciidoctor.convert_file 'sample.adoc', :in_place => true
```

The command will output to the file `sample.html` in the same directory.

You can render the file to DocBook 5.0 by setting the `:backend` option to *`docbook`*:

```ruby
Asciidoctor.convert_file 'sample.adoc', :in_place => true, :backend => 'docbook'
```

The command will output to the file `sample.xml` in the same directory. If you're on Linux, you can view the file using Yelp (https://wiki.gnome.org/action/show/Apps/Yelp).

You can also use the API to render strings and load custom templates.

4. Syntax Quick Start

Section Pending

5. Custom Output Quick Start

Section Pending

Getting Started

Section Pending

6. System Requirements

Asciidoctor works on Linux, Mac and Windows.

Asciidoctor requires one of the following implementations of Ruby:

- Ruby 1.8.7
- Ruby 1.9.3
- Ruby 2.0 (or better)
- JRuby 1.7.5 (Ruby 1.8 and 1.9 modes)
- Rubinius 2.0 (Ruby 1.8 and 1.9 modes)
- Opal (Javascript)

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

7. Installing the Asciidoctor Ruby Gem

Asciidoctor can be installed via the `gem` command, bundler, or popular Linux package managers.

To install Asciidoctor using the `gem` command:

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

```
$ gem install asciidoctor
```

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

```
$ asciidoctor --version
```

The following information should be output in your terminal:

```
Asciidoctor 1.5.2 [http://asciidoctor.org]
Runtime Environment (ruby 2.2.1p85 [x86_64-linux]) (lc:UTF-8 fs:UTF-8 in:- ex:UTF-8)
```

Two API Flavors

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

Asciidoctor also has a Java API that mirrors the Ruby API. The Java API calls through to the Ruby API using an embedded JRuby runtime. See the Asciidoctor Java integration project (http://asciidoctor.org/docs/install-and-use-asciidoctor-java-integration) for more information.

7.1. Install with Bundler

To install Asciidoctor using Bundler:

1. Open your system Gemfile
2. Add the `asciidoctor` gem to your Gemfile using the following text

```
source 'https://rubygems.org'
gem 'asciidoctor'
```

3. Save the Gemfile
4. Open a terminal
5. Install the gem with bundler

```
$ bundle install
```

7.2. Install with `yum`

To install Asciidoctor on Fedora or other RPM-based systems:

1. Open a terminal
2. Run the installation command

Fedora 21 or earlier

```
$ sudo yum install asciidoctor
```

Fedora 22 or later

```
$ sudo dnf install asciidoctor
```

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

7.3. Install with `apt-get`

To install Asciidoctor on Debian Sid or Ubuntu Saucy or greater:

1. Open a terminal
2. Type the `apt-get` command

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

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

8. Upgrading the Asciidoctor Ruby Gem

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

```
$ gem update asciidoctor
```

If you accidentally use `gem install` instead of `gem update` then you will have both versions installed. If you wish to remove the older version use the `gem` command:

```
$ gem cleanup asciidoctor
```

On Fedora, you can update it using:

```
$ sudo yum update rubygem-asciidoctor
```

Your Fedora system may be configured to automatically update packages, in which case no further action is required by you. Refer to the Fedora docs (http://docs.fedoraproject.org) if you are unsure.

On Debian or Ubuntu, you can update it using:

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

The Fedora, Debian and Ubuntu packages will not be available right away after a release of the RubyGem. It may take several weeks before the packages become available for a new release. If you need the latest version immediately, use the `gem install` option.

9. Extensions and Integrations

See Extensions.

Terms and Concepts

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

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

10. Elements

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

A document can include the following block elements:

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

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

Inline elements include:

- Quotes

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

11. Macros

Section pending

12. Attributes

Attributes are one of the features that sets Asciidoctor apart from other lightweight markup languages. Attributes activate features and store predefined styles, options, and replacement content.

In Asciidoctor, attributes are classified as:

- Intrinsic Attributes
- Attribute Entrys
- Attribute Lists
- BlockId Elements
- Macros
- API and Command Line Attributes

12.1. Attribute assignment precedence

By default, the attribute assignment precedence, from highest to lowest, is as follows:

- Attribute passed to the API or CLI
- Attribute defined in the document
- Intrinsic attribute value (default values)

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

The intrinsic value for the doctype attribute is `article`. Therefore, if doctype is not set and assigned a value in the document, API or CLI it will be assigned the article value (i.e. its default value). However, if doctype is set in the document and assigned a new value, such as `book`, the book value will override the intrinsic value. Finally, a value assigned to doctype via the API or CLI, will overrule the value in the document.

You can adjust the precedence of attribute values passed to the API or CLI. By adding an `@` symbol to the end of an attribute value passed to the API or CLI, it makes that assignment have a lower precedence than an assignment in the document.

Let's add that to the precedence list defined earlier.

- Attribute passed to the API or CLI that does not end in `@`
- Attribute defined in the document
- Attribute passed to the API or CLI that ends in `@`
- Intrinsic attribute value (default values)

12.2. Using attributes: set, assign, and reference

Before you can use an attribute in your document, you must **set** it. Sometimes, setting an attribute is also referred to as **toggling on**. To set an attribute, you must add its name to the document.

Once set, many attributes can be assigned a value. Values are the substitute text, content type, or operation that will replace any references to the attribute that the processor finds in the text.

An attribute reference is an inline element that calls the attribute entry with the same name. An attribute's reference is its name enclosed in curly brackets. The reference is replaced with the attribute's value when Asciidoctor processes the document.

A number of attributes and values are automatically used when the Asciidoctor processor renders a document. These attributes are called intrinsic attributes. For example, you do not need to set the `normal` attribute on a basic paragraph because it is set by an intrinsic attribute. But when you want a paragraph to have a different style, such as `TIP`, you set the TIP attribute on a paragraph and TIP will override the intrinsic normal attribute. Intrinsic attributes can be overridden or unset from the command line or within the document.

Attributes can be unset with the bang symbol (`!`). The `!` can be placed before or after the attribute's name.

For example, both:

```
:sectnums!:
```

and

```
:!sectnums:
```

means unset the `sectnums` attribute and therefore "do not number" the sections.

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

12.3. Setting attributes on a document

When an attribute is set in the header of a document, it is called an attribute entry.

Anatomy of an attribute entry

```
:attribute name: value
```

An attribute entry consists of the attribute's name and its value. The attribute's name is preceded and followed by a colon. After the second colon, there is at least one space and then the value. The value is optional.

Once set, an attribute entry and its value are available for use throughout your entire document. Attribute entries are used to toggle settings on and off or to set configuration variables that control the output generated by the AsciiDoc processor. For example, to include a table of contents in your document, you can turn on (i.e., set) the `toc` attribute entry.

```
:toc:
```
ASCIIDOC

When the value field following an attribute is left empty, as it is in the example above, the default value will be assigned. The default value for `toc` is `auto`; therefore, the table of contents will be placed on below the document's title when it is rendered. If you wanted the table of contents to be placed on the right side of the document, you must assign the attribute a new value.

```
:toc: right
```
ASCIIDOC

The `right` value will override the default value. The value assigned to an attribute in the document header will replace the intrinsic value.

Attribute entries are also used to store URLs. Here's an example:

```
:fedpkg: https://apps.fedoraproject.org/packages/asciidoc
```
ASCIIDOC

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

```
Information about the AsciiDoc package in Fedora is found at {fedpkg}.
```
ASCIIDOC

You can also set the base path to images (default: *empty*), icons (default: `./images/icons`), stylesheets (default: `./stylesheets`) and JavaScript files (default: `./javascripts`):

```
:imagesdir: ./images
:iconsdir: ./icons
:stylesdir: ./styles
:scriptsdir: ./js
```
ASCIIDOC

When you find yourself typing the same text repeatedly, or text that often needs to be updated, consider assigning it to a document attribute and use that instead.

Attribute entries have the following characteristics:

- can be assigned to a document via the CLI or in the document's header and body
- have default values
  - their default values are set when the user leaves the value field empty
- some have alternate values
- they can be unset (turned off) with a `!`
- they override config files, intrinsic attributes
- they do **not** override attributes issued from the command line
  - a number of entry attributes can also be used on the command line

12.3.1. Attribute entry substitutions

The `header` substitution group is applied to the header of your document. Text substitution elements replace characters, markup, attribute references, and macros with converter specific styles and values. When Asciidoctor processes a document it uses a set of six text substitution elements. In the header, only special characters and attribute references are replaced.

However, if you require other substitutions to be applied to an attribute's value, use the pass inline macro. This macro has special meaning in an attribute entry. It allows the substitutions to be applied at the time the attribute is defined.

The pass inline macro accepts a list of substitutions in the `target` slot. In the next example, we'll apply the quotes substitution to an attribute entry's value.

```
:app-name: pass:quotes[MyApp^(C)^]
```
ASCIIDOC

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

```
:app-name: pass:q[MyApp^(C)^]
```
ASCIIDOC

Another approach is to change the order of substitutions that are applied where the attribute is referenced.

```
:app-name: MyApp^(C)^

[subs="specialcharacters,attributes,quotes,replacements,macros,post_replacements"]
The application is called {app-name}.
```
ASCIIDOC

12.4. Built-in data attributes

Asciidoctor includes numerous intrinsic attributes that are assigned values when the document is rendered. The values of these built-in data attributes are derived from how the document is processed and when and where is it processed. These attributes can be referenced anywhere in the document.

Built-in data attributes

Attribute	Description
`asciidoctor`	Calls the processor
`asciidoctor-version`	Version of the processor
`backend`	Backend used to render document
`docdate`	Last modified date
`docdatetime`	Last modified date and time
`docdir`	Name of document directory
`docfile`	Name of document file
`doctime`	Last modified time

`doctitle`	The title of the document
`doctype`	Document's doctype (e.g., article)
`localdate`	Local date when rendered
`localdatetime`	Local date and time when rendered
`localtime`	Local time when rendered

12.5. Setting attributes on an element

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

Anatomy of an attribute list

```
[positional attribute,positional attribute,named attribute="value"]
```

Attribute lists:

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

12.5.1. Positional attribute

in an attribute list

not named

the first positional attribute in the list on inline quoted text is referred to as the role attribute

the first positional attribute in the list on blocks and macros is known as the style attribute

12.5.2. Named attribute

Named attributes are assigned a value with an `=` in an attribute list.

To undefine a named attribute, set the value to `none`.

12.5.3. Style

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

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

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

- comment
- example
- sidebar
- source

12.5.4. Id

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

An id has two purposes:

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

Block assignment

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

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

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

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

```
[quote#think, Donald Trump]
____
As long as you're going to be thinking anyway, think big.
____
```
ASCIIDOC

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

Inline assignment

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

Quoted text block with id assignment using Asciidoctor shorthand

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

12.5.5. Role

Section introduction pending

An element can be assigned numerous roles.

Block assignment

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

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

```
[.summary]
* Review 1
* Review 2
```
ASCIIDOC

```asciidoc
[role="summary"]
* Review 1
* Review 2
```

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

```asciidoc
[.summary.incremental]
* Review 1
* Review 2
```

```asciidoc
[role="summary,incremental"]
* Review 1
* Review 2
```

Inline assignment

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

Quoted text with role assignments using traditional AsciiDoc syntax

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

Quoted text with role assignments using Asciidoctor shorthand

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

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

Role-playing for text enclosed in backticks

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

Given:

```asciidoc
[rolename]`escaped monospace text`
```

the following HTML is produced:

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

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

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

which produces:

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

Keep in mind that text enclosed is not subject to other inline substitutions, but rather passed through as is.

12.5.6. Options

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

It can also be defined globally with an attribute entry.

Block assignment

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

1. Prefixing the value with a percent sign (`%`).

2. Specifying the value with `opts=<name>`
3. Specifying the value with `options=<name>`.

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

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

Shorthand Asciidoctor syntax

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

Here's how the options are assigned to the table using `options`.

Traditional AsciiDoc syntax

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

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

Shorthand Asciidoctor block syntax

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

Traditional AsciiDoc block syntax

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

12.6. Assigning document variables inline

Document variables can be assigned using the following syntax:

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

For example:

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

It's effectively the same as:

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

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

12.7. Attribute conventions

Section Pending

12.7.1. Catch a missing or undefined attribute

By default, the original AsciiDoc processor drops the entire line if it contains a reference to a missing attribute (e.g., `{bogus}`). This "feature" was added for use in templates written for the original processor, which also used the AsciiDoc syntax.

This behavior is not needed in Asciidoctor since templates are written in a dedicated template language (e.g., ERB, Haml, Slim, etc). More critically, the behavior is frustrating for the author, editor or reader. To them, it's not immediately apparent when a line goes missing. Discovering its absence often requires a full (and tedious) read-through of the document or section.

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

Missing attribute

The attribute attribute-missing controls how missing references are handled. By default, missing references are left intact so it's clear to the author when one hasn't been satisfied since, likely, the intent is for it to be replaced.

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 (compliant behavior)

`warn`
: print a warning about the missing attribute

Consider the following line of AsciiDoc:

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

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

`skip`
: Hello, {name}!

`drop`
: Hello, !

`drop-line`
:

`warn`
: ```
  WARNING: skipping reference to missing attribute: XYZ
  ```

*Undefined attribute*

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

This attribute has two possible values:

`drop`
: substitute the expression with an empty string after processing it

`drop-line`
: drop the line that contains this expression (default setting and compliant behavior)

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

Consider the following declaration:

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

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

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

Building a Document

Introduction Pending

13. Text Editor

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

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

The most popular application for editing plain text on Mac OS X is **TextMate**. A similar choice on Linux is **GEdit**. On Windows, stay away from Notepad and Wordpad because they produce plain text which is not cross-platform friendly. Opt instead for a competent text editor like **Notepad++**. If you're a programmer (or a writer with an inner geek), you'll likely prefer **Vim**, **Emacs**, or **Sublime Text**, all of which are available cross-platform. The key feature all these editors share is syntax highlighting for AsciiDoc (http://asciidoc.org/#_editor_support).

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

14. Document Types

Article

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

Book

The same as articles with the additional ability to use a top level title as part titles, includes the appendix, dedication, preface, bibliography, glossary, index, and colophon.

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.

The rules for the inline doctype are as follows:

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

Given the following input:

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

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

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

The Asciidoctor processor is now able to cover the full range of output, from unstructured (inline) text to a full, standalone document!

Asciidoctor only.

Man page

Special title and section naming conventions used to generate roff format UNIX manual pages. Corresponds to the DocBook refentry document type.

15. Basic Document Anatomy

Section pending

16. Header

The document header contains title, author and revision information as well as any document-wide attributes set and defined by the user. The header is *not required* when the `doctype` is `article` or `book`. If absent, Asciidoctor will render the content that is present.

A header must be present when the document type is `manpage`. The requirements for a manual page (man page) are described in the man pages section.

When the header is present, it must start with a document title. The title can be immediately followed by two optional lines of text defining author and revision information. Finally, any document-wide settings are configured using attribute entries. The header can not contain any blank lines, but it can contain comments.

The header is processed when rendering a full document. This means that the header of a document called via an include directive will be processed and rendered. When you do not want the header of a document to be rendered, set the `noheader` or `no-header-footer` attribute in the document's header or 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 render the content. Asciidoctor has a number of attributes available to correctly handle front matter. See the static website generators section to learn how Asciidoctor integrates with static website generators.

Now let's look at the document title in detail.

16.1. Document title

A document title must be placed on the first line of the document. It is preceded by one equal sign followed by at least one space (`= `), and then the text of the title.

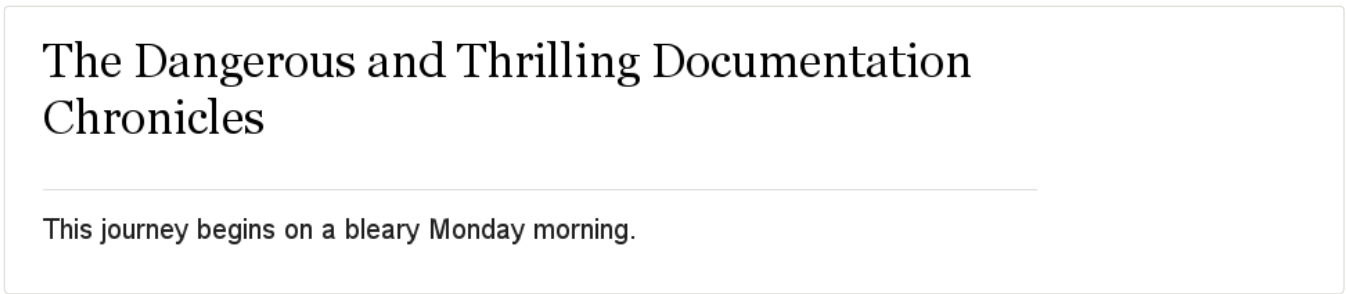
Here's an example of a document title followed by an abbreviated paragraph.

Document with a title

```
= The Dangerous and Thrilling Documentation Chronicles

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

ASCIIDOC

Result: Rendered document title



When `doctype` is set to `article` or `manpage`, a document can have only one section level 0 title. The `book` document type can have multiple section level 0 titles. The first section level 0 title, located in the header, is the document's title, and subsequent section level 0 titles are considered the titles of parts.

A document's title is assigned to the built-in `doctitle` attribute. Its value is identical to the value returned by `Document#doctitle` and `sect0`. The doctitle attribute can be referenced anywhere in a document and displays the document's title when rendered.

Referencing the doctitle attribute

```
= The Dangerous and Thrilling Documentation Chronicles

{doctitle} begins on a bleary Monday morning.
```

ASCIIDOC

Result: doctitle output

You can control whether or not the title is displayed when the document is rendered with the attributes `notitle` and `showtitle`. If you don't want the title to be rendered, set the `notitle` attribute in the header or CLI. On the other hand, the title is not displayed when a document's header and footer are disabled. The header and footer are disabled when a document is embedded. In this case, set the `showtitle` attribute in order to render the title.

Let's see how to add additional information to the header, including an author and her email address.

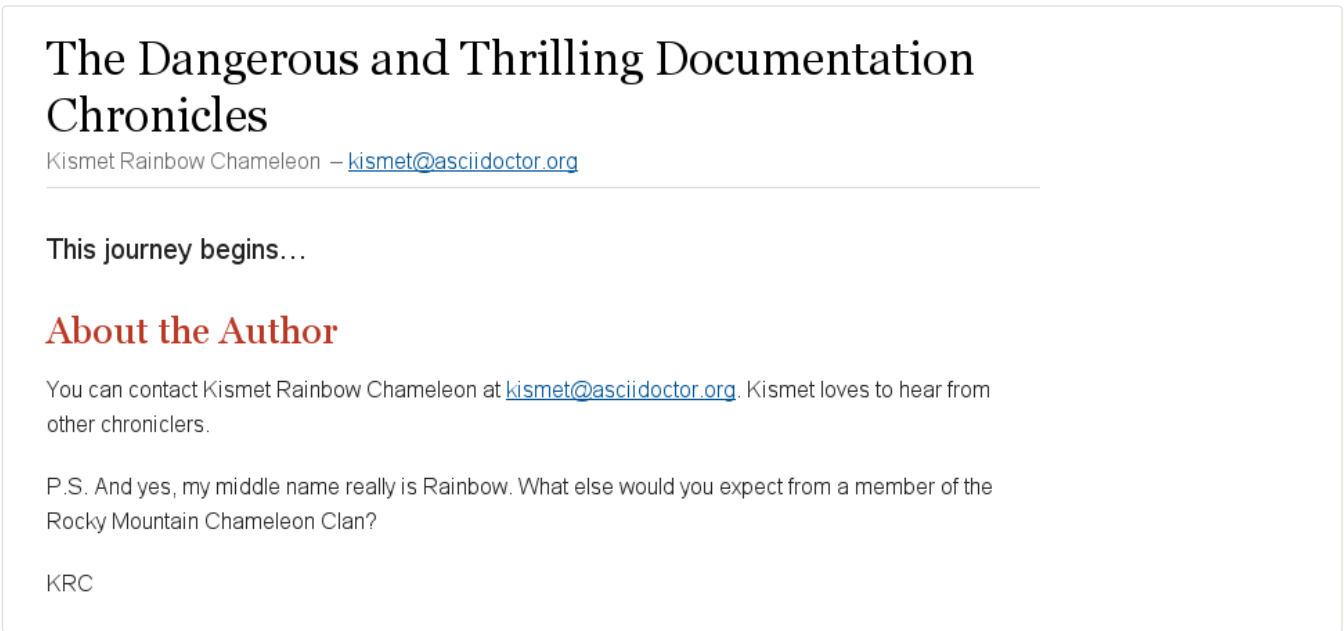
16.2. Author and email

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

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

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

This journey begins...

== About the Author

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

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

{authorinitials}
```
ASCIIDOC

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



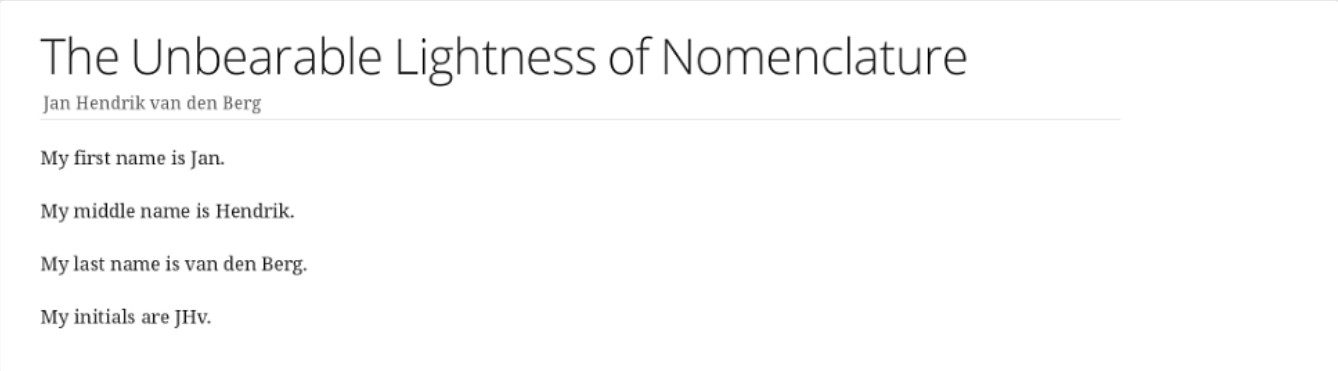
As you can see in the example above, Asciidoctor uses the author's name and email to assign values to a number of built-in attributes that can be used throughout the document's body. These attributes include:

`author`	The author's full name, which includes all of the characters or words prior to a semicolon (`;`), angle bracket (`<`) or the end of the line.
`firstname`	The first word in the author attribute.
`lastname`	The last word in the author attribute.
`middlename`	If a firstname and lastname are present, any remaining words or characters found between these attributes are assigned to the middlename attribute.
`authorinitials`	The first character of the firstname, middlename, and lastname attributes.
`email`	An email address, delimited by angle brackets (`<>`).

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

```asciidoc
= The Unbearable Lightness of Nomenclature
Jan Hendrik van_den_Berg

My first name is {firstname}.

My middle name is {middlename}.

My last name is {lastname}.

My initials are {authorinitials}.
```

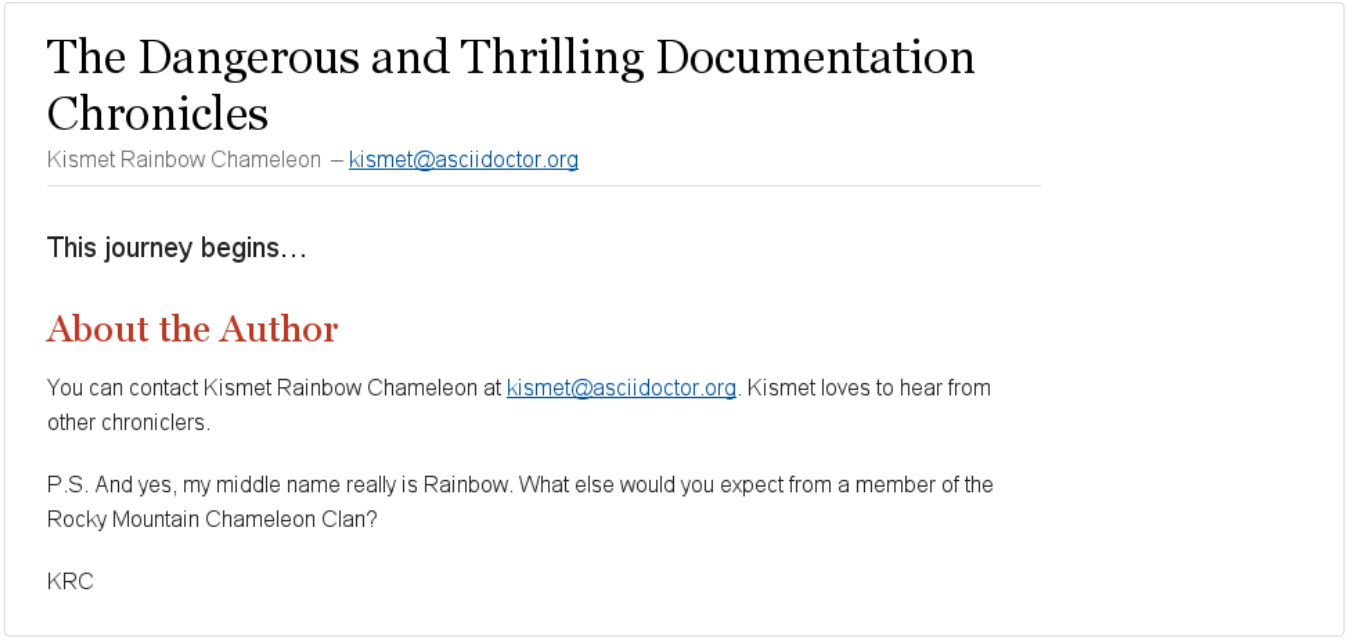
Result: Rendered author information when author has a compound name



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

```asciidoc
= The Dangerous and Thrilling Documentation Chronicles
:author: Kismet Rainbow Chameleon
:email: kismet@asciidoctor.org

This journey begins...

== About the Author

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

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

{authorinitials}
```

Result: Rendered author information when author and email attributes are explicitly set

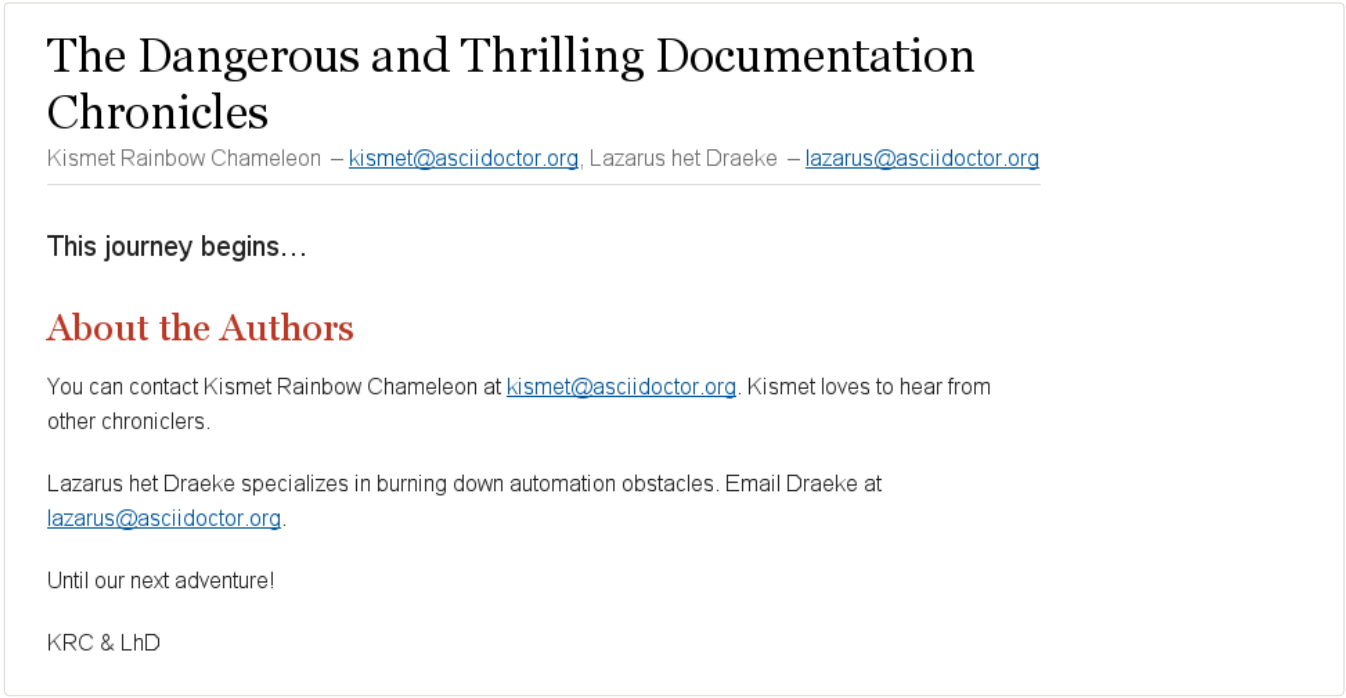


The `html5` and `docbook` converters can render documents with multiple authors. Multiple authors and their emails are separated by semicolons (`;`) when they're listed on the same line.

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

This journey begins...

== About the Authors

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

{author_2} specializes in burning down automation obstacles. (1)
Email {lastname_2} at {email_2}.

Until our next adventure!

{authorinitials} & {authorinitials_2}
```

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

Result: Rendered author information when document has multiple authors



16.3. Revision number, date, and remark

A document's revision information contains three optional attributes.

`revnumber` The document's version number which must contain at least one numeric character. Any letters or symbols preceding the numeric character will not be rendered. If the `revdate` is absent, the `revnumber` must be followed by a comma (e.g., `v1.0,`) or (as of Asciidoctor 1.5.3) begin with a "v" character (e.g., `v1.0`). If the `revdate` is present, it must be separated from the `revnumber` by a comma.

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

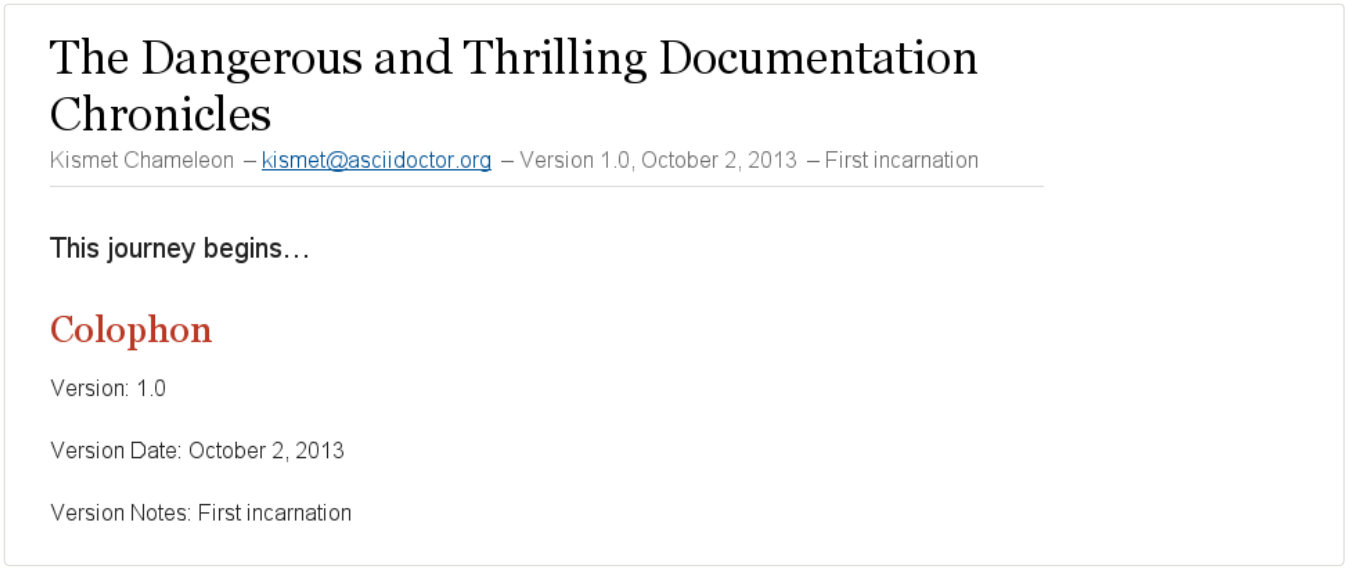
`revremark` Information about this version of the document.

The revision information is listed on the third line of the header, beneath the author information line.

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

This journey begins...

== Colophon

Version: {revnumber}

Version Date: {revdate}

Version Notes: {revremark}
```
ASCIIDOC

1. revnumber and revdate must be separated by a comma (`,`).
2. revdate can contain words, letters, numbers, and symbols.
3. The revremark attribute must be preceded by a colon (`:`), regardless of whether revnumber or revdate are set.

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



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

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

This journey begins...

== Colophon

Version: {revnumber}

Version Date: {revdate}

Version Notes: {revremark}
```

ASCIIDOC

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

(2) The revremark can contain attribute references.

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

Result: Rendered revision information when revision attributes are explicitly set

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

Revision extraction information and an extraction example are pending.

16.4. Subtitle partitioning

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

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

A document title that contains a subtitle

```
= Main Title: Subtitle
```
ASCIIDOC

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

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

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

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

A document title with an explicit title separator

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

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

A document title with an explicit title separator

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

or from the API or CLI (shown here):

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

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

Retrieving a partitioned document title

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

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

Retrieving a partitioned document title with a custom separator

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

16.5. Metadata

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

16.5.1. Description

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

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

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

ASCIIDOC

(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 rendered to HTML, the document description value is assigned to the HTML `<meta>` tag.

HTML output

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<meta name="generator" content="Asciidoctor 0.1.4">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="A story chronicling the inexplicable hazards and vicious beasts a documentation tear
<title>The Dangerous and Thrilling Documentation Chronicles</title>
<style>
```

XML

16.5.2. Keywords

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

```
= The Dangerous and Thrilling Documentation Chronicles
Kismet Rainbow Chameleon <kismet@asciidoctor.org>; Lazarus het_Draeke <lazarus@asciidoctor.org>
:keywords: documentation, team, obstacles, journey, victory

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

ASCIIDOC

HTML output

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<meta name="generator" content="Asciidoctor 0.1.4">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="keywords" content="documentation, team, obstacles, journey, victory">
<title>The Dangerous and Thrilling Documentation Chronicles</title>
<style>
```

XML

16.5.3. Alternate title

By default, the document title is used as the HTML `<title>` tag value. However, you can set the `title` attribute in the document's header to override this behavior.

16.5.4. Custom metadata, styles and functions

You can add content, such as custom metadata, stylesheet, and script information, to the header of a rendered document with document information (`docinfo`) files. The docinfo file section details what these files can contain and how to use them.

16.6. Header summary

Header attributes and values

Attribute	Values	Description	Notes	Converters
`author`		Author's full name		all
`authorinitials`		First character of each word in the `author` attribute		all
`description`		Text describing the document		html
`docinfo`, `docinfo1`, `docinfo2`		Adds content from a docinfo file to header		html, docbook
`doctitle`	Text entered by user	Title of document	Identical to the value returned by `Document#doctitle`	all
`email`		Email address		all
`firstname`		First word of `author` attribute		all
`keywords`		A list of comma-separated values that describe the document		html
`lastname`		Last word of `author` attribute		all
`middlename`		Middle word of `author` attribute		all
`no-header-footer`, `-s`		Suppresses the rendering of the header and footer		all
`noheader`		Suppresses the rendering of the header		all
`notitle`		Toggles the display of a document's title		all
`revdate`		Date of document version		all
`revnumber`		Version number of the document		all
`revremark`		Version comments		all
`showtitle`		Toggles the display of an embedded document's title		all
`title`		Alternative title of the document		html
`version-label`	`Version`, User defined	The label preceding the `revnumber` in a render document's byline		html

17. Preamble

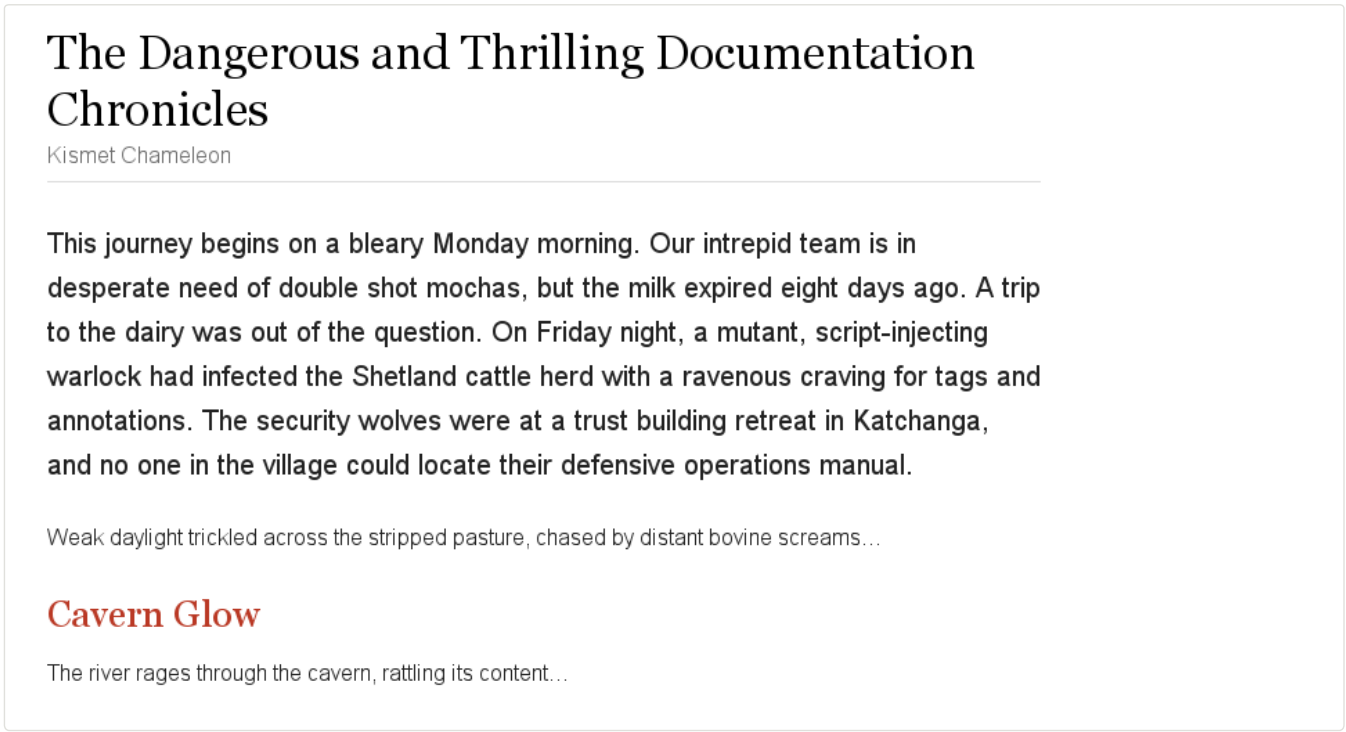
Content between the document header and the first section level title is called the preamble. A preamble is optional.

Preamble

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

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

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

== Cavern Glow

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

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

Result: Rendered preamble



18. Sections

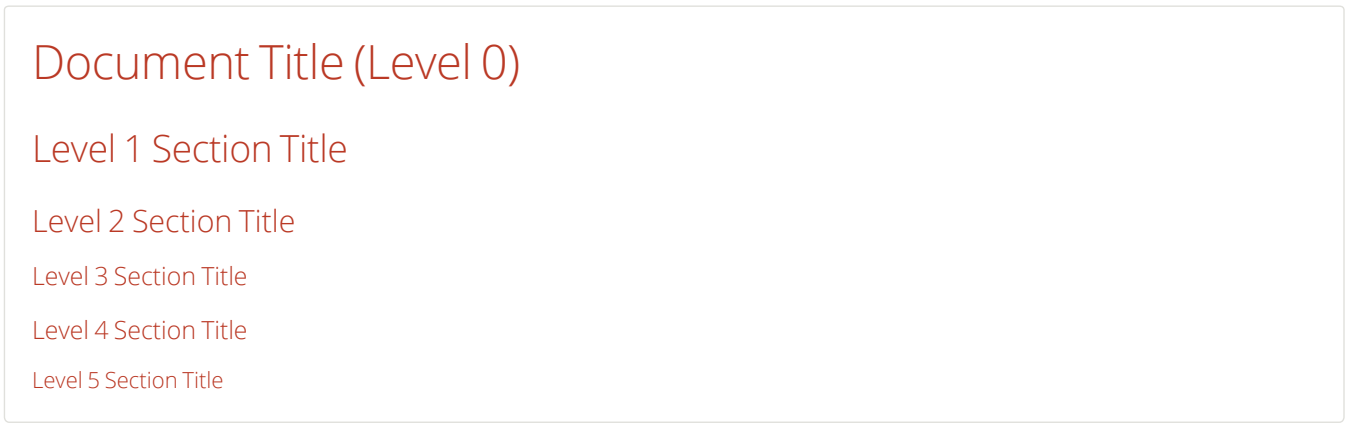
Sections partition the document into a content hierarchy. In Asciidoctor, sections are defined using section titles.

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

Section titles available in an article doctype

```asciidoc
= Document Title (Level 0)

== Level 1 Section Title

=== Level 2 Section Title

==== Level 3 Section Title

===== Level 4 Section Title

====== Level 5 Section Title

== Another Level 1 Section Title
```

Render section titles



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

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

1. A document can only have multiple level 0 sections if the doctype is set to book.
2. Section levels cannot be skipped when nesting sections (i.e., you can't nest a level 5 section directly in a level 3 section)

For example, the following syntax is illegal:

```asciidoc
= 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
```
ASCIIDOC

18.1. Titles as HTML headings

When the document is rendered as 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.

18.2. Auto-generated IDs

Section IDs are generated from the section title. By default, the ID is prefixed with an underscore (`_`) and each word in the title is separated by an underscore. You can change the prefix using the `idprefix` attribute.

```
:idprefix: -
```
ASCIIDOC

If you want to remove the prefix, assign it to an empty value:

```
:idprefix:
```
ASCIIDOC

You can change the word separator by using the `idseparator` attribute.

```
:idseparator: -
```
ASCIIDOC

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

```
:sectids!:
```
ASCIIDOC

Asciidoctor permits all valid UTF-8 characters in section IDs. If you are generating a PDF from AsciiDoc using a2x and dblatex, see Using UTF-8 titles with a2x (http://aerostitch.github.io/misc/asciidoc/asciidoc-title_uft8.html) to learn about the required `latex.encoding=utf8` switch.

18.3. Custom IDs

You can also assign a custom ID and reference text label to a section (see Defining an Anchor). This can be useful when you define a cross reference pointing to the section.

```
[[tigers-section, Tigers]]
=== Subspecies of Tiger
```
ASCIIDOC

18.4. 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.

18.5. Anchors

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.

18.6. Numbering

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

```
:sectnums:
```
ASCIIDOC

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

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

```
:sectnums!:

== Unnumbered Section
```
ASCIIDOC

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

Given:

```
= Document Title

:sectnums!:

== Colophon Section

== Another Colophon Section

== Last Colophon Section

:sectnums:

== Section One

== Section Two

== Section Three
```
ASCIIDOC

The sections will be numbered as follows:

```
Colophon Section

Another Colophon Section

Last Colophon Section

1. Section One

2. Section Two

3. Section Three
```
ASCIIDOC

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

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

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

18.7. Discrete or floating section titles

The `discrete` attribute can be applied to any section titles that start with two to six equal signs (`==`). A discrete title is styled like a section title but is not part of the content hierarchy (i.e., it ignores section nesting rules). Nor will it be included in the ToC.

```
[discrete] (1)
== Discrete Title for a Sidebar (2)

**** (3)
Discrete titles are useful for labeling large sidebar and admonition blocks.
****
```
ASCIIDOC

1. Set the `discrete` attribute above the title
2. The title must be designated by two to six equal signs
3. Delimiter specifying the beginning of a sidebar block

You can also use the `float` attribute to create a discrete title. However, the content in the discrete section will not be repositioned (i.e., float) to the left or right of other content blocks.

18.8. Section styles

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

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

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

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

```
[abstract]
== Documentation Abstract

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

The structure and usage rules for each section style is explained in Structuring, Navigating, and Referencing Your Content.

18.9. Sections summary

Section attributes and values

Attribute	Value(s)	Example Syntax	Comments
`sectid`	N/A	`:sectid!:`	Intrinsic; set to autogenerate by default
`idprefix`	_, or user defined text	`:idprefix: -`	Intrinsic; set to autogenerate a _ by default
`sectanchors`	N/A	`:sectanchors:`	Asciidoctor only
`sectlinks`	N/A	`:sectlinks:`	Asciidoctor only
`sectnums`	N/A	`:sectnums:`	Section numbering is off by default. Can be toggled on or off through document.

19. Blocks

Section Pending

19.1. Title

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

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

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

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

19.2. Metadata

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

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

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

```asciidoc
.Gettysburg Address (1)
[[gettysburg]] (2)
[quote, Abraham Lincoln, Address delivered at the dedication of the Cemetery at Gettysburg] (3) (4) (5)
____
Four score and seven years ago our fathers brought forth
on this continent a new nation...

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

(1) Title: Gettysburg Address

(2) ID: gettysburg, see Defining an Anchor

(3) Block name: quote

(4) attribution: Abraham Lincoln (Named block attribute)

(5) citetitle: Address delivered at the dedication of the Cemetery at Gettysburg (Named block attribute)

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

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

19.3. Delimited blocks

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

Here's an example of a sidebar block:

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

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

19.3.1. Delimiter lines

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

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

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

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

```
****
valid example block
****
```

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

19.3.2. Optional delimiters

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

This format is often used for single-line listings:

```
[listing]
sudo yum install asciidoc
```
ASCIIDOC

or single-line quotes:

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

19.3.3. Masquerading blocks

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

19.4. Built-in blocks summary

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

Block	Block Name	Delimiter	Purpose	Substitutions
Admonition	`[<LABEL>]`	Any delimiter	Content labeled with a tag or icon, can masquerade as any delimited block type	Varies
Comment	N/A	`////`	Private notes that are not displayed in the output	None
Example	`[example]`	`====`	Designates example content or defines an	Normal

			admonition block	
Fenced	N/A	```	Source code or keyboard input is displayed as entered	Verbatim
Listing	`[listing]`	`----`	Source code or keyboard input is displayed as entered	Verbatim
Literal	`[literal]`	`....`	Output text is displayed exactly as entered	Verbatim
Open	Most block names	`--`	Anonymous block that can act as any block except *passthrough* or *table* blocks	Varies
Passthrough	`[pass]`	`++++`	Unprocessed content that is sent directly to the output	None
Quote	`[quote]`	`____`	A quotation with optional attribution	Normal
Sidebar	`[sidebar]`	`****`	Aside text and content rendered outside the flow of the document	Normal
Source	`[source]`	`----`	Source code or keyboard input to be displayed as entered	Verbatim
Stem	`[stem]`	`++++`	Unprocessed content that is sent directly to an interpreter (such as AsciiMath or LaTeX math)	None
Table	N/A	`|===`	Displays tabular content	Varies
Verse	`[verse]`	`____`	A verse with optional attribution	Normal

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

Substitution groups

Group	Special characters	Quotes	Attributes	Replacements	Macros	Post replacements
Header	✔	✖	✔	✖	✖	✖
None	✖	✖	✖	✖	✖	✖
Normal	✔	✔	✔	✔	✔	✔
Pass	✖	✖	✖	✖	✖	✖
Verbatim	✔	✖	✖	✖	✖	✖

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

20. Paragraph

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

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

Two paragraphs in an AsciiDoc document

```
This journey begins one late Monday afternoon in Antwerp.
Our team desperately needs coffee, but none of us dare open the office door.

To leave means code dismemberment and certain death.
```
ASCIIDOC

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

This journey begins one late Monday afternoon in Antwerp. Our team desperately needs coffee, but none of us dare open the office door.

To leave means code dismemberment and certain death.

20.1. Line breaks

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

However, if you want the line breaks in a paragraph to be preserved, you can either use a plus sign (`+`) or the `hardbreaks` attribute. This results in a visible line break (e.g., `<br>`) folowing each line.

Line breaks preserved using the plus sign (+)

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

Rubies are red,
Topazes are blue.

Line breaks preserved using the hardbreaks attribute

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

Ruby is red.
Java is black.

Alternatively, you can preserve line breaks throughout your whole document by adding the `hardbreaks` attribute to the document's header.

Line breaks preserved throughout the document using the hardbreaks attribute

```
= Line Break Doc Title
:hardbreaks:

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

You can also preserve line breaks using literal blocks, listing blocks, and verses.

20.2. Lead style

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

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

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

Rendered lead paragraph

This is the ultimate paragraph.

When you render a document using the default backend and stylesheet, the first paragraph of the preamble is automatically styled as a lead paragraph.

21. Text Formatting

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

Text formatting versus quotes

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

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

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

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

21.1. Bold and italic

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

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

Bold and italic text formatting syntax

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

Result: bold and italic text

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

21.2. Quotation marks and apostrophes

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

Single and double curved quotation marks syntax

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

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

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

Rendered curved quotation marks

> “What kind of charm?” Lazarus asked. “An odoriferous one or a mineral one?”
>
> Kizmet shrugged. “The note from Olaf’s desk says ‘wormwood and licorice,’ but these could be normal groceries for werewolves.”

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

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

Curved apostrophe syntax

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

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

Rendered apostrophes

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

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

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

Rendered escaped apostrophe

> Olaf's desk ⋯

21.3. Subscript and superscript

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

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

Subscript

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

Superscript

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

```
"`Well the H~2~O formula written on their whiteboard could be part of a shopping list, but I don't think the local bodega s
```
ASCIIDOC

> “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.

You can write and render equations and formulas in AsciiDoc documents using MathJax.

21.4. Monospace

Monospace text formatting is often used to emulate how source code appears in computer terminals, simple text editors, and integrated development environments (IDEs). In Asciidoctor 1.5, inline content is rendered using a fixed-width font, i.e. monospaced font, when it is enclosed in a single set of backticks (`` ` ``). A character bounded by other characters must be enclosed in a double set of backticks (`` ``).

```
"`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 scent of science`; `_smell like a genius_`.
```
ASCIIDOC

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

Rendered monospace formatted text

> “Wait!” Indigo plucked a small vial from her desk’s top drawer and held it toward us. The vial’s label read: $E=mc^2$; `the scent of science`; `*_smell like a genius_*`.

21.5. Custom styling with attributes

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

Highlighted style syntax

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

Highlighted text HTML output

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

Rendered highlighted text

> Werewolves are allergic to cinnamon.

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

Built-in CSS class syntax

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

[big]##O##nce upon an infinite loop.
```
ASCIIDOC

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

Rendered CSS class styled text

> Do werewolves believe in small print?
>
> Once upon an infinite loop.

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

Custom CSS class syntax

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

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

HTML output

```
<span class="userinput">asciidoctor</span>
```
HTML

22. Unordered Lists

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

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

```
* Edgar Allen Poe
* Sheri S. Tepper
* Bill Bryson
```
ASCIIDOC

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

Rendered unordered list

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

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

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

Rendered unordered list with a title

Kizmet's Favorite Authors

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

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

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

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

Separating Lists

If you have adjacent lists, they have the tendency to want to fuse together. To force the lists apart, place a line comment between them (`//`), offset on either side by a blank line (i.e., an end of list marker). Here's an example, where the `^` is dummy text that indicates this line serves as an "end of list" marker:

```
* Apples
* Oranges

//^

* Walnuts
* Almonds
```
ASCIIDOC

22.1. Nested

To nest an item, just add another asterisk (`*`) in front of it.

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

Rendered nested, unordered list

Possible DefOps manual locations

- West wood maze
  - Maze heart
    - Reflection pool
  - Secret exit
- Untracked file in git repository

You can have up to five levels of nesting.

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

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

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

22.2. Complex list content

Aside from nested lists, all of the list items you've seen only have one line of text. A list item can hold any type of AsciiDoc content, including paragraphs, listing blocks and even tables. You just need to *attach* them to the list item.

Like with regular paragraph text, the text in a list item can wrap across any number of lines, as long as all the lines are adjacent. The wrapped lines can be indented and they will still be treated as normal paragraph text. For example:

```
* The header in AsciiDoc is optional, but if
it is used it must start with a document title.

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

* The document header must be separated from
  the remainder of the document by one or more
  blank lines and cannot contain blank lines.
```
ASCIIDOC

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

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

If you want to attach additional paragraphs to a list item, you "add" them together using a *list continuation*. A list continuation is a `+` symbol on a line by itself, immediately adjacent to the two blocks it's connecting. Here's an example:

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

* Optional Author and Revision information
  immediately follows the header title.
```

ASCIIDOC

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

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

Using the list continuation, you can attach any type of block element and you can use the list continuation any number of times in a single list item.

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

```
* The header in AsciiDoc must start with a
  document title.
+
----
= Document Title
----
+
NOTE: The header is optional.

* Optional Author and Revision information
  immediately follows the header title.
+
----
= Document Title
Doc Writer <doc.writer@asciidoc.org>
v1.0, 2013-01-01
----
```

ASCIIDOC

Here's how the source is rendered:

A list with complex content

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

  ```
  = Document Title
  ```

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

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

22.3. Custom markers

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

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

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

These styles are supported by the default Asciidoctor stylesheet.

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

For HTML

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

For DocBook

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

Here's an unordered list that has square bullets:

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

- one
- two
- three

22.4. Checklist

List items can be marked complete using checklists.

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

Checklist syntax

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

Rendered checklist

- ☑ checked
- ☑ also checked
- ☐ not checked

normal list item

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

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

Since HTML generated by Asciidoctor is usually static, the checkbox is set as disabled to make it appear as a simple mark. If you want to make the checkbox interactive (i.e., clickable), add the `interactive` option to the checklist:

Checklist with interactive checkboxes

```
[options=interactive]
- [*] checked
- [x] also checked
- [ ] not checked
-     normal list item
```

Rendered checklist with interactive checkboxes

- [x] checked
- [x] also checked
- [ ] not checked

normal list item

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

Checklist with font-based checkboxes

```
[options=interactive]
- [*] checked
- [x] also checked
- [ ] not checked
-     normal list item
```

22.5. Summary

Pending

23. Ordered Lists

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

```
1. Protons
2. Electrons
3. Neutrons
```
ASCIIDOC

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
```
ASCIIDOC

1. Protons
2. Electrons
3. Neutrons

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

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

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

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

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
```
ASCIIDOC

Parts of an atom

1. Protons
2. Electrons
3. Neutrons

23.1. Nested

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

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

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

A nested ordered list

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

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

> *# of dots = level of nesting*

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

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

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

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

1. Linux
    - Fedora
    - Ubuntu
    - Slackware
2. BSD
    - FreeBSD
    - NetBSD

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

```
. Linux

  * Fedora
  * Ubuntu
  * Slackware

. BSD

  * FreeBSD
  * NetBSD
```

The labeled list section demonstrates how to combine all three list types.

23.2. Numbering styles

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

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

Asciidoctor ordered list numeration styles

Block style	CSS list-style-type
arabic	decimal
decimal*	decimal-leading-zero
loweralpha	lower-alpha
upperalpha	upper-alpha
lowerroman	lower-roman
upperroman	upper-roman
lowergreek*	lower-greek

* 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]*	[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

* 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
```

ASCIIDOC

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

23.3. Summary

Pending

24. Labeled List

Labeled lists are useful when you need to include a description or supporting text for each item in a list. Each item in a labeled list consists of a term or phrase followed by:

- a separator (typically a double colon, `::`)
- at least one space or endline
- the item's content

Here's an example of a labeled list that identifies parts of a computer:

```
ASCIIDOC
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 labeled list

CPU
: The brain of the computer.

Hard drive
: Permanent storage for operating system and/or user files.

RAM
: Temporarily stores information the CPU uses during operation.

Keyboard
: Used to enter text or control items on the screen.

Mouse
: Used to point to and select items on your computer screen.

Monitor
: Displays information in visual form using text and graphics.

If you want the label and content to appear on the same line, add the horizontal style to the list.

```
ASCIIDOC
[horizontal]
CPU:: The brain of the computer.
Hard drive:: Permanent storage for operating system and/or user files.
RAM:: Temporarily stores information the CPU uses during operation.
```

CPU	The brain of the computer.
Hard drive	Permanent storage for operating system and/or user files.
RAM	Temporarily stores information the CPU uses during operation.

The content of a labeled list can be any AsciiDoc element. For instance, we could rewrite the grocery list from above so that each aisle is a label rather than a parent outline list item.

```
Dairy::
* Milk
* Eggs
Bakery::
* Bread
Produce::
* Bananas
```

Dairy

- Milk
- Eggs

Bakery

- Bread

Produce

- Bananas

Labeled lists are quite lenient about whitespace, so you can spread the items out and even indent the content if that makes it more readable for you:

```
Dairy::

  * Milk
  * Eggs

Bakery::

  * Bread

Produce::

  * Bananas
```

Finally, you can mix and match the three list types within a single hybrid list. Asciidoctor works hard to infer the relationships between the items that are most intuitive to us humans.

Here's a list that mixes labeled, ordered, and unordered list items:

```
Operating Systems::
  Linux:::
    . Fedora
      * Desktop
    . Ubuntu
      * Desktop
      * Server
  BSD:::
    . FreeBSD
    . NetBSD

Cloud Providers::
  PaaS:::
    . OpenShift
    . CloudBees
  IaaS:::
    . Amazon EC2
    . Rackspace
```

Here's how the list is rendered:

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 complex content in a list item as well.

24.1. Question and answer style list

Q&A

```
[qanda]
What is Asciidoctor?::
  An implementation of the AsciiDoc processor in Ruby.
What is the answer to the Ultimate Question?:: 42
```

1. *What is Asciidoctor?*

   An implementation of the AsciiDoc processor in Ruby.

2. *What is the answer to the Ultimate Question?*

   42

24.2. Summary

Section pending

25. Tables

Tables are one of the most refined areas of the AsciiDoc syntax. They are easy to create, easy to read in raw form and also remarkably sophisticated.

Tables are delimited by `|===` and made up of cells. The default table data format is PSV (Prefix Separated Values), which means that the processor creates a new cell each time it encounters a vertical bar (`|`). The cells are grouped into rows. Each row must share the same number of cells, taking into account any column or row spans. Then, each consecutive cell in a row is placed in a separate column. The simple table example below consists of two columns and three rows.

Simple table

```
|=== (1)
(2)
| Cell in column 1, row 1 | Cell in column 2, row 1 (3)
(4)
| Cell in column 1, row 2 | Cell in column 2, row 2

| Cell in column 1, row 3 | Cell in column 2, row 3

|=== (1)
```

(1) The table's content boundaries are defined by a vertical bar followed by three equal signs (`|===`).

(2) A blank line separates the table delimiter and the first row so the first row is not styled as the table's header.

(3) The new cell is marked by a vertical bar (`|`).

(4) Rows are separated by blank lines.

Result: Rendered simple table

Cell in column 1, row 1	Cell in column 2, row 1
Cell in column 1, row 2	Cell in column 2, row 2
Cell in column 1, row 3	Cell in column 2, row 3

The leading and trailing spaces around the content in a cell are stripped; therefore, they don't affect the table's layout when it is rendered. The two examples below illustrate how leading and trailing spaces don't change the rendered table's layout.

Cell content adjacent to the |

```
|===

|Cell in column 1, row 1|Cell in column 2, row 1

|===
```

Result: Rendered table when cell content was entered adjacent to the |

Cell in column 1, row 1	Cell in column 2, row 1

Cell content with varying leading and trailing spaces

```
|===

| Cell in column 1, row 1        |          Cell in column 2, row 1

|===
```

Result: Rendered table when cell content was bounded by varying leading and trailing spaces

Cell in column 1, row 1	Cell in column 2, row 1

There are multiple ways to group cells into a row. The cells in a row can be placed on:

a. the same line

b. consecutive, individual lines

c. a combination of a and b

Cells on the same line

```asciidoc
|===

|Cell in column 1, row 1 |Cell in column 2, row 1 |Cell in column 3, row 1

|Cell in column 1, row 2 |Cell in column 2, row 2 |Cell in column 3, row 2

|===
```

Result: Rendered table when multiple cells where entered on the same line

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

When the cells of a row are individually entered on consecutive lines, the `cols` attribute is needed to specify the number of columns in the table. If the `cols` attribute is not set, the first non-blank line inside the block delimiter (`|===`) determines the number of columns.

Cells on consecutive, individual lines

```asciidoc
[cols="3*"] (1)
|===
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

(1) The `cols` attribute states that this table has three columns. The `*` is a repeat operator which is explained in the column specifiers section.

Result: Rendered table when cells where listed on consecutive, individual lines

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

Rows can be formed from adjacent lines of individual cells and cells listed on the same line.

Cells on the same line and individual lines

```asciidoc
[cols="3*"]
|===
|Cell in column 1, row 1 |Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2 |Cell in column 3, row 2
|===
```

Result: Cells on the same line and individual lines

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

The next sections describe and demonstrate the variety of ways you can customize table cells, rows and columns.

25.1. Columns

The number of columns in a table is determined by the number of cells found in the first non-blank line after the table delimiter (`|===`) or by the values assigned to the `cols` attribute.

For example, the syntax in the two examples below will both render a table with two columns.

```
|===

|Cell in column 1, row 1 |Cell in column 2, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2

|===
```

ASCIIDOC

Result: Rendered table with two columns as defined by the number of cells in the first row

Cell in column 1, row 1	Cell in column 2, row 1
Cell in column 1, row 2	Cell in column 2, row 2

```
[cols="2*"]
|===

|Cell in column 1, row 1
|Cell in column 2, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2

|===
```

ASCIIDOC

Result: Rendered table with two columns as defined by the cols attribute

Cell in column 1, row 1	Cell in column 2, row 1
Cell in column 1, row 2	Cell in column 2, row 2

When a single number is assigned to the cols attribute, its value indicates the number of columns. Each column will be the same width. However, the number of columns can also be assigned as a comma delimited list. The number of entries in the list determines the number of columns.

The comma delimited list below creates a table with four columns of equal width.

```
[cols="1,1,1,1"]
```

ASCIIDOC

This syntax provides that same result:

```
[cols="4*"]
```

ASCIIDOC

Now, let's talk about that asterisk in the syntax above.

25.2. Column formatting

The AsciiDoc syntax provides a variety of ways to control the size, style and layout of content within columns. These **specifiers** can be applied to whole columns.

To apply a specifier to a column, you must set the `cols` attribute and assign it a value. A column specifier can contain any of the following components:

- multiplier
- align
- width
- style

Each component is optional.

The multiplier operator (`*`) is used when you want a specifier to apply to more than one consecutive column. If used, the multiplier must always be placed at the beginning of the specifier.

For example:

```
[cols="3*"] (1)
|===
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

(1) The table will consist of three columns, as indicated by the **3**. The `*` operator ensures that the default layout and style will be applied to all of the columns.

Result: Rendered table with multiplier applied

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

The alignment component allows you to horizontally or vertically align a column's content. Content can be horizontally aligned left (`<`), center (`^`), or right (`>`).

To horizontally center the content in all of the columns, add the `^` operator after the multiplier.

```
[cols="3*^"]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

Result: Rendered table with horizontal, center alignment applied to all columns

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

What if you only want to center the content in the last column? Assign the default styles to the preceding columns, and `^` to the last column in a comma separated list.

```
[cols="2*,^"]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

Result: Rendered table with horizontal, center alignment applied to last column

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

Let's specify a different horizontal alignment for each column.

```
[cols="<,^,>"]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

Result: Rendered table with a different horizontal alignment for each column

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

You'll notice that the content in the examples above is only centered on the horizontal. It can also be aligned vertically when the alignment operator is prefixed with a dot (`.`). Content can be vertically aligned to the top (`<`), middle (`^`), or bottom (`>`) of a cell.

To vertically align the content to the middle of the cells in all of the columns, add a `.` and then the `^` operator after the multiplier.

```
[cols="3*.^"]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

Result: Rendered table with vertical, middle alignment applied to all columns

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

If you only want to align the content to the bottom of each cell in the last column, you'd assign the default styles to the preceding columns, and `>` to the last column in a comma separated list.

```
[cols="2*,.>"]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

Result: Rendered table with vertical, bottom alignment applied to last column

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

Let's specify a different vertical alignment for each column.

```
[cols=".<,.^,.>"]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

Result: Rendered table with a different vertical alignment for each column

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

Finally, we'll also horizontally center the content in the last column.

```
[cols=".<,.^,^.>"]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

Result: Rendered table with a different vertical alignment for each column and horizontal, center alignment in the last column

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

When both a horizontal and vertical alignment is assigned to a column, the horizontal alignment operator must precede the vertical operator.

The width component sets the width of a column. Its value can be a proportional integer (the default is 1) or a percentage (1 to 99).

```
[cols="1,2,6"]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

Result: Table rendered with column sizes adjusted by a proportional integer

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

When assigning percentage values to the cols attribute, you do not need to include the percent sign (`%`).

```
[cols="50,20,30"]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

Result: Table rendered with column sizes adjusted by a percentage

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

Let's create a table with custom widths and alignments.

```
[cols=".<2,.^5,^.>3"]
|===
|Cell in column 1, row 1 with lots and lots and lots and lots of content
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2 and another bucket of content, and then a jelly roll of content
|===
```

Result: Rendered table with variable widths and alignments

Cell in column 1, row 1 with lots and lots and lots and lots of content	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2 and another bucket of content, and then a jelly roll of content

The style component must always be located at the end of the specifier. When no style name is provided column contents will be processed as regular inline text.

The column styles are described in the table below.

Style Name	**Value**	**Description**
AsciiDoc	`a`	Any block-level elements (paragraphs, delimited blocks and block macros) may be contained within the column. The elements will be processed and rendered.
Emphasis	`e`	Text is italicized
Header	`h`	Header styles are applied to the column
Literal	`l`	Column content is treated as if it were inside a literal block
Monospaced	`m`	Text is rendered in monospaced font
None (default style)	`d`	Text is handled like a normal paragraph. Supports all markup (i.e., inline formatting, inline macros) that is permitted in a paragraph.
Strong	`s`	Text is bolded
Verse	`v`	Column content is treated as if it were inside a verse block

Let's apply the header style to the first column, the monospaced style to the second, the strong style to the third, and the emphasis style to the fourth.

```
[cols="h,m,s,e"]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1
|Cell in column 4, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|Cell in column 4, row 2
|===
```
ASCIIDOC

Result: Rendered table with a header, monospaced, and strong styled column

Cell in column 1, row 1	`Cell in column 2, row 1`	**Cell in column 3, row 1**	*Cell in column 4, row 1*
Cell in column 1, row 2	`Cell in column 2, row 2`	**Cell in column 3, row 2**	*Cell in column 4, row 2*

Specifiers can also be applied to individual cells.

25.3. Cell Formatting

In addition to sharing many of the column specifier capabilities, cell specifiers allow cells to span rows and columns. Like a column specifier, a cell specifier is made up of components. These components, listed and defined below, are all optional.

- span
- align
- style

A cell specifier is prefixed directly to the cell delimiter (`|`) preceding the content you want to customize.

The span component can duplicate a cell or have it span multiple rows or columns.

To duplicate a cell in multiple, consecutive columns, prefix the `|` with the multiplication factor and the `*` operator.

Cell duplicated across three columns

```
|===

|Cell in column 1, row 1 |Cell in column 2, row 1 |Cell in column 3, row 1

3*|Same cell content in columns 1, 2, and 3

|Cell in column 1, row 3
|Cell in column 2, row 3
|Cell in column 3, row 3

|===
```

Result: Rendered table where cell was duplicated across three columns

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Same cell content in columns 1, 2, and 3	Same cell content in columns 1, 2, and 3	Same cell content in columns 1, 2, and 3
Cell in column 1, row 3	Cell in column 2, row 3	Cell in column 3, row 3

To have a cell span multiple, consecutive columns, prefix the `|` with the span factor and the `+` operator.

Cell spanning three columns

```
|===

|Cell in column 1, row 1 |Cell in column 2, row 1 |Cell in column 3, row 1

3+|Content in a single cell that spans columns 1, 2, and 3

|Cell in column 1, row 3
|Cell in column 2, row 3
|Cell in column 3, row 3

|===
```

Result: Rendered table where cell spans three columns

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Content in a single cell that spans columns 1, 2, and 3		
Cell in column 1, row 3	Cell in column 2, row 3	Cell in column 3, row 3

If you want to have a cell span multiple, consecutive rows, prefix the span factor with a dot (`.`).

Cell spanning two rows

```
|===

|Cell in column 1, row 1 |Cell in column 2, row 1 |Cell in column 3, row 1

.2+|Content in a single cell that spans rows 2 and 3
|Cell in column 2, row 2
|Cell in column 3, row 2

|Cell in column 2, row 3
|Cell in column 3, row 3

|===
```

Result: Rendered table where a cell spans two rows

Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Content in a single cell that spans rows 2 and 3	Cell in column 2, row 2	Cell in column 3, row 2
	Cell in column 2, row 3	Cell in column 3, row 3

The alignment component for cells works the same as the column specifier alignment component.

Cells aligned horizontally, vertically, and across a span of three columns

```
[cols="3"]
|===
^|Prefix the +{vbar}+ with +{caret}+ to center content horizontally
<|Prefix the +{vbar}+ with +<+ to align the content to the left horizontally
>|Prefix the +{vbar}+ with +>+ to align the content to the right horizontally

.^|Prefix the +{vbar}+ with a +.+ and +{caret}+ to center the content in the cell vertically
.<|Prefix the +{vbar}+ with a +.+ and +<+ to align the content to the top of the cell
.>|Prefix the +{vbar}+ with a +.+ and +>+ to align the content to the bottom of the cell

3+^.^|This content spans three columns (+3{plus}+) and is centered horizontally (+{caret}+) and vertically (+.{caret}+) wit

|===
```

Result: Rendered cells aligned horizontally, vertically, and across a span of three columns

Prefix the {vbar} with {caret} to center content horizontally	Prefix the {vbar} with < to align the content to the left horizontally	Prefix the {vbar} with > to align the content to the right horizontally

Prefix the {vbar} with a . and {caret} to center the content in the cell vertically	Prefix the {vbar} with a . and < to align the content to the top of the cell	Prefix the {vbar} with a . and > to align the content to the bottom of the cell
This content spans three columns (3{plus}) and is centered horizontally ({caret}) and vertically (.{caret}) within the cell.		

To use a pipe (`|`) within the content of a cell without creating a new cell, you must use the `{vbar}` attribute.

The style component can also be applied to individual cells. For example, you can apply the AsciiDoc element styles to an individual cell by prefixing the vertical bar with an `a`.

Comparing cells using AsciiDoc styles and no AsciiDoc styles

```
[cols="2"]
|===

a|This cell is prefixed with an +a+ so the following list is rendered with the AsciiDoc list element styles.

* List item 1
* List item 2
* List item 3
|This cell *is not* prefixed with an +a+ so the following list is not rendered with the AsciiDoc list element styles.

* List item 1
* List item 2
* List item 3

a|This cell is prefixed with an +a+ so the following paragraph is rendered with the +lead+ style.

[.lead]
I am a paragraph styled with the lead attribute.
|This cell *is not* prefixed with an +a+ so the following paragraph is not rendered with the +lead+ style.

[.lead]
I am a paragraph styled with the lead attribute.

|===
```

Result: Rendered table comparing cells using AsciiDoc styles and no AsciiDoc styles

This cell is prefixed with an a so the following list is rendered with the AsciiDoc list element styles. • List item 1 • List item 2 • List item 3	This cell **is not** prefixed with an a so the following list is not rendered with the AsciiDoc list element styles. * List item 1 * List item 2 * List item 3
This cell is prefixed with an a so the following paragraph is rendered with the lead style. I am a paragraph styled with the lead attribute.	This cell **is not** prefixed with an a so the following paragraph is not rendered with the lead style. [.lead] I am a paragraph styled with the lead attribute.

For the grand finale, let's apply a variety of specifiers to individual cells.

Building a variety of cell specifiers

```
|===

2*>m|This content is duplicated across two columns.

It is aligned right horizontally.

And it is monospaced.

.3+^.>s|This cell spans 3 rows. The content is centered horizontally, aligned to the bottom of the cell, and strong.
e|This content is emphasized.

.^l|This content is aligned to the top of the cell and literal.

v|This cell contains a verse
that may one day expound on the
wonders of tables in an
epic sonnet.

|===
```

Result: Rendered table featuring a variety of cell specifiers

This content is duplicated across two columns. It is aligned right horizontally. And it is monospaced.	This content is duplicated across two columns. It is aligned right horizontally. And it is monospaced.
This cell spans 3 rows. The content is centered horizontally, aligned to the bottom of the cell, and strong.	*This content is emphasized.*
	`This content is aligned to the top of the cell and literal.`
	This cell contains a verse that may one day expound on the wonders of tables in an epic sonnet.

25.4. Header row

The first row of a table can be rendered as a header row with the `header` value. The header value is assigned by setting the `options` attribute or based on the table's layout.

The `options` attribute is set in the table's attribute list and assigned the value `header` in the example below.

Table with the header value assigned to the options attribute

ASCIIDOC

```
[cols="2*", options="header"]
|===
|Name of Column 1
|Name of Column 2

|Cell in column 1, row 1
|Cell in column 2, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|===
```

Result: Rendered table when the header value is assigned to the options attribute

Name of Column 1	Name of Column 2
Cell in column 1, row 1	Cell in column 2, row 1
Cell in column 1, row 2	Cell in column 2, row 2

Alternatively, you can define the header row based on how you layout the table. Asciidoctor use the following conventions to determine when the first row should become the header row:

1. The first line of content inside the table block delimiters is not empty.
2. The second line of content inside the table block delimiters is empty.
3. The `options` attribute has not been specified in the block attributes.

As seen in the result of the example below, if all of these rules hold, then the first row of the table is treated as a header.

Table that has an implicit header row

```
|===
|Name of Column 1 |Name of Column 2

|Cell in column 1, row 1
|Cell in column 2, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|===
```

Result: Rendered table when the header row was assigned implicitly

Name of Column 1	Name of Column 2
Cell in column 1, row 1	Cell in column 2, row 1
Cell in column 1, row 2	Cell in column 2, row 2

Notice that when the implicit method of assigning the header row is used, it is not necessary to set the `cols` attribute.

We're considering using a similar convention for enabling the footer in the future. Thus, if you rely on this convention to enable the header row, it's advised that you not put all the cells in the last row on the same line unless you intend on making it the footer row.

25.5. Footer Row

The last row of a table can be styled as a footer by assigning the `footer` value to the `options` attribute.

```
[options="footer"]
|===
|Name of Column 1 |Name of Column 2

|Cell in column 1, row 1
|Cell in column 2, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2

|Footer in column 1, row 3
|Footer in column 2, row 3
|===
```

Result: Table rendered with a footer

Name of Column 1	Name of Column 2
Cell in column 1, row 1	Cell in column 2, row 1
Cell in column 1, row 2	Cell in column 2, row 2
Footer in column 1, row 3	Footer in column 2, row 3

Remember that when the options attribute is set, and you want the first row of the table to be displayed as a header, you must assign the header value.

25.6. Table lines, placement, and size

There are several attributes available for customizing the placement, size and lines of a table.

By default, a table will stretch to 100 percent of the page width. To reduce the width of the table, set the `width` attribute in the table's attribute list. The width value can be any number between 1 and 99.

Table with width set to 65%

```
[width="65"]
|===
|Name of Column 1 |Name of Column 2 |Name of Column 3

|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

Result: Render table with a width of 65%

Name of Column 1	Name of Column 2	Name of Column 3
Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

The border around a table can be changed by setting the `frame` attribute. By default, the frame attribute is assigned the `all` value, which draws a border on each side of the table. If you set the frame attribute, you can override the default value with `topbot`, `sides` or `none`.

The `topbot` value draws a border on the top and bottom of the table.

```
[frame="topbot"]
|===
|Name of Column 1 |Name of Column 2 |Name of Column 3

|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

Result: Table rendered with topbot value

Name of Column 1	Name of Column 2	Name of Column 3
Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

The `sides` value draws a border on the right and left side of the table.

```
[frame="sides"]
|===
|Name of Column 1 |Name of Column 2 |Name of Column 3

|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

Result: Table rendered with sides value

Name of Column 1	Name of Column 2	Name of Column 3
Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

The `none` value removes the borders around the table.

```
[frame="none"]
|===
|Name of Column 1 |Name of Column 2 |Name of Column 3

|Cell in column 1, row 1
|Cell in column 2, row 1
|Cell in column 3, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2
|Cell in column 3, row 2
|===
```

Result: Table rendered with none value

Name of Column 1	Name of Column 2	Name of Column 3
Cell in column 1, row 1	Cell in column 2, row 1	Cell in column 3, row 1
Cell in column 1, row 2	Cell in column 2, row 2	Cell in column 3, row 2

At this time, Asciidoctor has not implemented the `align`, `halign`, or `valign` attributes.

25.7. CSV and DSV Data Formats

Tables can be created from Comma Separated Values (CSV) and Delimiter Separated Values (DSV). To use these type of datasets, set the `format` attribute and assign it the `csv` or `dsv` value. Tables using these alternate data formats are structured and formatted exactly like `psv` tables, the data separators are just different.

When the format is set to csv the data separator is a comma (`,`), as seen in the table below.

```
[format="csv", options="header"]
|===
Artist,Track,Genre
Baauer,Harlem Shake,Hip Hop
The Lumineers,Ho Hey,Folk Rock
|===
```
ASCIIDOC

Result: Rendered CSV table

Artist	Track	Genre
Baauer	Harlem Shake	Hip Hop
The Lumineers	Ho Hey	Folk Rock

When the format is dsv, the data separator is a colon (`:`).

```
[format="dsv", options="header"]
|===
Artist:Track:Genre
Robyn:Indestructable:Dance
The Piano Guys:Code Name Vivaldi:Classical
|===
```
ASCIIDOC

Result: Rendered DSV table

Artist	Track	Genre

Robyn	Indestructable	Dance
The Piano Guys	Code Name Vivaldi	Classical

Asciidoctor also provides shorthand notation for setting CSV and DSV table formats. The first position of the table block delimiter (i.e., `|===`) can be replaced by the data delimiter to set the table format.

Instead of specifying the csv format using a block attribute, you can simply replace the leading pipe (`|`) with a comma (`,`):

```
ASCIIDOC
,===
Artist,Track,Genre

Baauer,Harlem Shake,Hip Hop
,===
```

Result: Rendered CSV table using shorthand syntax

Artist	Track	Genre
Baauer	Harlem Shake	Hip Hop

In the same way, the dsv format can be specified by replacing the leading pipe (`|`) with a colon (`:`).

```
ASCIIDOC
:===
Artist:Track:Genre

Robyn:Indestructable:Dance
:===
```

Result: Rendered DSV table using shorthand syntax

Artist	Track	Genre
Robyn	Indestructable	Dance

DSV and CSV datasets can also be inserted into a table using the include directive.

```
ASCIIDOC
[format="csv", options="header"]
|===
include::tracks.csv[]
|===
```

Result: Rendered table of CSV data pulled from a separate file

Artist	Track	Genre
Baauer	Harlem Shake	Hip Hop
The Lumineers	Ho Hey	Folk Rock
Robyn	Indestructable	Dance
The Piano Guys	Code Name Vivaldi	Classical

25.8. Summary

Table attributes and values

Attribute	Description	Value	Description	Notes
`format`	data format of the table's contents	`psv`	Prefixed Separated Values, default, cells are delimited by a |	
		`dsv`		

			Delimiter Separated Values, cells are delimited by a colon (`:`)	
		`csv`	Comma Separated Values, cells are delimited by a comma (`,`)	
`separator`	cell separator	`|`, `:`, `,`, `!`, or a user defined attribute	`|` is the default, `!` separates cells in nested PSV tables	
`frame`	draws a border around the table	`all`	default, border on all sides	
		`topbot`	border on top and bottom	
		`none`	no borders	
		`sides`	border on left and right sides	
`grid`	draws boundary lines between rows and columns	`all`	default, draws boundary lines around each cell	
		`cols`	draws boundary lines between columns	
		`rows`	draws boundary lines between rows	
		`none`	no boundary lines	
`align`	horizontally aligns table within page width	`left`	default, aligns to left side of page	Not yet implemented in Asciidoctor. AsciiDoc HTML output only. The `align` and `float` attributes are mutually exclusive.
		`right`	aligns to right side of page	
		`center`	horizontally aligns to center of page	
`float`	aligns the table to the side of the page in conjunction with the table's `width` attribute	`left`	default, aligns the table with the left side of the page	AsciiDoc HTML output only. The `float` and `align` attributes are mutually exclusive.
		`right`	aligns the table with the right side of the page	
`halign`	horizontally aligns all of the cell contents in a table	`left`	default, aligns the contents of the cells to the left	Not yet implemented in Asciidoctor. AsciiDoc HTML output only. Overridden by column and cell specifiers.
		`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`	default, aligns the cell contents to the top of the cell	Not yet implemented in Asciidoctor. AsciiDoc HTML output only. Overridden by column and cell specifiers.
		`bottom`	aligns the cell contents to the bottom of the cell	
		`middle`	aligns the cell contents to the middle of the cell	

`options`	comma separated list of table values	`header`	styles the first table row	By default header and footer rows are omitted
		`footer`	styles the last table row	
`cols`	comma separated list of column specifiers	specifiers		
`width`	the table width relative to the available page width	user defined value	a percentage between 1 and 99	
`filter`	a shell command for content within a cell			

26. Horizontal Rules

Horizontal rule syntax

```asciidoc
'''
```

Result: Horizontal rule

26.1. Markdown-style horizontal rules

Asciidoctor recognizes Markdown horizontal rules. The motivation here is to ease migration of Markdown documents to AsciiDoc documents.

To avoid conflicts with AsciiDoc's block delimiter syntax, only 3 repeating characters (`-` or `*`) are recognized. As with Markdown, whitespace between the characters is optional.

Markdown-style horizontal rule syntax

```markdown
---

- - -

***

* * *
```

Result: Markdown-style horizontal rules

A macro definition for the Markdown horizontal rules is included in the AsciiDoc compatibility file so they can be recognized by the `asciidoc` command as well.

27. Page break

Page break syntax

```asciidoc
<<<
```

28. URLs

A Uniform Resource Link (URL) represents the location of a resource on the web. Typical URLs contain a scheme, domain name, file name, and extension.

Asciidoctor recognizes the following common schemes without the help of any markup.

- http
- https
- ftp
- irc
- mailto
- email@email.com

Since the URL in the example below begins with *http*, Asciidoctor will automatically turn it into a hyperlink when it is processed.

```
The homepage for the Asciidoctor Project is http://www.asciidoctor.org. (1)
```
ASCIIDOC

(1) The trailing period will not get caught up in the link.

To prevent automatic linking of an URL, prepend it with a backslash (`\`).

If you prefer URLs to be rendered without a visible scheme, set the `hide-uri-scheme` attribute in the document's header.

```
:hide-uri-scheme:

http://asciidoctor.org
```
ASCIIDOC

When the hide-uri-scheme attribute is set, the above URL will render as follows:

```
<a href="http://asciidoctor.org">asciidoctor.org</a>
```
XML

Note the absence of *http* inside the `<a>` element.

To attach a URL to text, enclose the text in square brackets at the end of the URL.

```
Chat with other Asciidoctor users in the irc://irc.freenode.org/#asciidoctor[Asciidoctor IRC channel].
```

Additionally, you can format the linked text.

```
Ask questions on the http://discuss.asciidoctor.org/[*mailing list*].
```

Rendered URLs

The homepage for the Asciidoctor Project is http://asciidoctor.org.

Chat with other Asciidoctor users in the Asciidoctor IRC channel.

Ask questions on the **mailing list** (http://discuss.asciidoctor.org/).

When a URL does not start with one of the common schemes, you must use the `link` macro. The URL is preceded by `link:` and followed by square brackets. The square brackets can include optional link text, target preference, and roles.

Anatomy of a link macro

```
link:url[optional link text, optional target attribute, optional role attribute]
```

First, let's look at an example of a link macro that contains link text.

```
Let's view the raw HTML of the link:view-source:asciidoctor.org[Asciidoctor homepage].
```

Let's view the raw HTML of the Asciidoctor homepage.

Troubleshooting Complex URLs

A URL may not render correctly when it contains characters such as underscores (`_`) or carets (`^`). This problem occurs because the markup parser interprets parts of the URL (i.e., the link target) as valid text formatting markup. Most lightweight markup languages have this issue because they don't use a grammar-based parser. Asciidoctor plans to handle URLs more carefully in the future (see issue #281 (https://github.com/asciidoctor/asciidoctor/issues/281)), which may be solved by moving to a grammar-based parser (see issue #61 (https://github.com/asciidoctor/asciidoctor/issues/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: http://www.asciidoctor.org/now_this__link_works.html

This URL has repeating underscores {link-with-underscores}.
```
ASCIIDOC

Asciidoctor won't break links with underscores when they are assigned to an attribute because inline formatting markup is substituted before attributes. The URL remains hidden while the rest of the document is being formatted (strong, emphasis, monospace, etc).

Solution B

Another way to solve formatting glitches is to explicitly specify the formatting you want to have applied to a span of text. This can be done by using the inline pass macro. If you want to display a URL, and have it preserved, put it inside the pass macro and enable the macros substitution, which is what substitutes links.

```
This URL has repeating underscores pass:macros[http://www.asciidoctor.org/now_this__link_works.html].
```
ASCIIDOC

The pass macro removes the URL from the document, applies the `macros` substitution to the URL, and then restores the processed URL to its original location once the substitutions are complete on the whole document.

Alternatively, you can use `++` around the URL only. However, when you use this approach, Asciidoctor won't recognize it as a URL anymore, so you have to use the explicit `link` prefix.

```
This URL has repeating underscores link:++http://www.asciidoctor.org/now_this__link_works.html++[].
```
ASCIIDOC

For more information, see issue #625 (https://github.com/asciidoctor/asciidoctor/issues/625).

Next, we'll add a target and role to a link macro.

When you set attributes in the link macro, you must also set the `linkattrs` attribute in the document's header because Asciidoctor does not parse attributes in the link macro by default. Once you've set linkattrs in the header, you can then specify the name of the target window using the `window` attribute.

```
= Asciidoctor Document Title
:linkattrs:

Let's view the raw HTML of the link:view-source:asciidoctor.org[Asciidoctor homepage, window="_blank"].
```
ASCIIDOC

Let's view the raw HTML of the Asciidoctor homepage.

Since `_blank` is the most common window name, we've introduced shorthand for it. Just end the link text with a caret (`^`):

```
Let's view the raw HTML of the link:view-source:asciidoctor.org[Asciidoctor homepage^].
```
ASCIIDOC

If you use the caret syntax more than once in a single paragraph, you may need to escape the first occurrence with a backslash.

When linkattrs is set, you can add a role (i.e., CSS class) to the link.

```
Chat with other Asciidoctor users in the irc://irc.freenode.org/#asciidoctor[Asciidoctor IRC channel] or on the http://disc
```
ASCIIDOC

Chat with other Asciidoctor users in the Asciidoctor IRC channel or on the **mailing list** (http://discuss.asciidoctor.org/).

Links with attributes (including the subject and body segments on mailto links) are a feature unique to Asciidoctor. When they are enabled, you must surround the link text in double quotes if it contains a comma.

28.1. Link to relative files

If you want to link to an external file relative to the current document, use the `link` macro in front of the file name.

```
link:protocol.json[Open the JSON file]
```
ASCIIDOC

If your file is an HTML file, you can link directly to a section in the document, append a hash (`#`) followed by the section's ID to the end of the file name.

```
link:external.html#livereload[LiveReload]
```
ASCIIDOC

28.2. Summary

Link attributes and values

Attribute	Value(s)	Example Syntax	Comments
`linkattrs`		`:linkattrs:`	Must be set in the header to parse link macro attributes.
`window`	blank	http://discuss.asciidoctor.org[Discuss Asciidoctor, window="_blank"]	The blank value can also be specified using `^`. Requires `linkattrs`.
`window`	`^`	http://example.org["Google, Yahoo, Bing^"] and http://discuss.asciidoctor.org[Discuss Asciidoctor^]	Requires `linkattrs`.

`role`	CSS classes available to inline elements	http://discuss.asciidoctor.org[Discuss Asciidoctor, role="teal"]	Requires `linkattrs`.
`id`	name of element, custom link text	<<section-title,cross reference text>>	Applies to cross references

29. Cross references

A link to another location within an AsciiDoc document or between AsciiDoc documents is called a *cross reference* (also referred to as an *xref*). To be able to create a cross reference, you first need to define the location where the reference will point to. This is the anchor definition.

29.1. Defining an Anchor

An anchor can be defined anywhere: header, image, listing or text block. The anchor's ID is defined between two square brackets.

Defining an inline anchor ID

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

Inline anchors make arbitrary content referenceable.

It is possible to customize the text that will be used in the cross reference link (called `xreflabel`). If not defined, Asciidoctor does it best to find suitable text (the solution differs from case to case). In case of an image, the image caption will be used. In case of a section header, the text of the section's title will be used.

To define the `xreflabel`, add it in the anchor definition right after the ID (separated by a comma).

An anchor ID with a defined xreflabel. The caption will not be used as link text.

```
[[tiger-image,Image of a tiger]]
.This image represents a Bengal tiger also called the Indian tiger
image::tiger.png[]
```

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

Setting an anchor ID using the macro form

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

29.2. Internal cross references

In Asciidoctor, the xref inline macro is used to create cross references (i.e. links) to sections, blocks or phrases that have an ID (explicit or auto-generated).

An implicit cross reference is created by enclosing the ID of the target block or section (or the path of another document with an optional anchor) in double angled brackets.

Cross reference using the ID of the target section

```
The section <<images>> describes how to insert images into your document.
```
ASCIIDOC

Rendered cross reference using the ID of the target section

The section Images describes how to insert images into your document.

You can also link to a block or section using the title by referencing its title. However, the title must begin with an uppercase letter (in basic Latin) without any leading formatting marks.

Cross reference using a section's title

```
Refer to <<Internal cross references>>.
```
ASCIIDOC

Rendered cross reference using a section's title

Refer to Internal cross references.

Some converters, such as the HTML converter, will use the `xreflabel` as the default text of the link. However, you can also customize this text. After the ID, add a comma and then enter the custom text you want the cross reference to display.

Cross reference with custom xreflabel text

```
Learn how to <<link-macro-attributes,use attributes within the link macro>>.
```
ASCIIDOC

Rendered cross reference using custom xreflabel text

Learn how to use attributes within the link macro.

You can also use the inline xref macro as an alternative to the double angled bracket form.

Inline xref macro

```
Learn how to xref:link-macro-attributes[use attributes within the link macro].
```
ASCIIDOC

29.3. Inter-document cross references

The xref inline 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 Asciidoctor:

```
The section <<images>> describes how to insert images into your document.
```

This cross reference creates a link to the section with the ID *images*.

Let's assume the cross reference is defined in the document *document-a.adoc*. If the target section is in a separate document, *document-b.adoc*, the author may be tempted to write:

```
Refer to link:document-b.html#section-b[Section B] for more information.
```

However, this link is coupled to HTML output. What's worse, if *document-b.adoc* is included in the same master as *document-a.adoc*, then the link will refer to a document that doesn't even exist!

These problems can be alleviated by using an inter-document xref:

```
Refer to <<document-b.adoc#section-b,Section B>> for more information.
```

The ID of the target is now placed behind a hash symbol (`#`). Preceding the hash is the name of the reference document (the file extension is optional). We've also added a label since Asciidoctor cannot (yet) resolve the section title in a separate document.

When Asciidoctor generates the link for this cross reference, it first checks to see if *document-b.adoc* is included in the same master as *document-a.doc*. 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 master 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.

29.3.1. Navigating between source files

In certain environments, such as GitHub and the browser extensions, you view the generated HTML from the AsciiDoc source URL. Since the default suffix for relative links is `.html`, the inter-document cross references end up pointing to non-existent HTML files. You need, instead, to get inter-document cross references to refer to other AsciiDoc source files. You can achieve this behavior by setting the `outfilesuffix` attribute to the value as `.adoc`, as the example below shows.

```
= Document Title
ifdef::env-github,env-browser[:outfilesuffix: .adoc]

See the <<README#,README>>.

We could also write the link as link:README{outfilesuffix}[README].
```

The links in the generated document will now point to *README.adoc* instead of the default, *README.html*.

You probably don't want to set `outfilesuffix` to `.adoc` without the `ifdef` condition as it could result in Asciidoctor overwriting input files when you run it locally (though there's some protection against this).

30. Images

To include an image on its own line (i.e., a *block image*), use the `image` prefix in front of the file name and square brackets after it.

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

If you want to specify alt text, include it inside the square brackets:

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

You can also give the image an id, a title, set its dimensions and make it a link.

```
[[img-sunset]] (1)
.A mountain sunset (2)
image::sunset.jpg[Sunset, 300, 200, link="http://www.flickr.com/photos/javh/5448336655"] (3) (4) (5)
```
ASCIIDOC

1. ID, see Defining an Anchor.
2. The title of a block image is displayed underneath the image when rendered.
3. The first positional attribute, *Sunset*, is the image's alt text.
4. Image width and height
5. `link` makes the image a link

A hyperlinked image with caption


(http://www.flickr.com/photos/javh/5448336655)

Figure 1: A mountain sunset

Block images are prefixed by a caption label (Figure) and number automatically. To turn off figure caption labels and numbers, add the `figure-caption` attribute to the document header and unset it.

```
:figure-caption!:
```

Automatic image scaling

The default Asciidoctor stylesheet implements responsive images (using width-wise scaling). If the width of the screen is smaller than the width of the image, the image will be scaled down to fit. To support this feature, the original aspect ratio of the image is preserved at all sizes. Thus, when you set the dimensions, the values should reflect the original aspect ratio of the image as this will be enforced. If the values don't match the aspect ratio, the height is ignored by the browser.

If you want to include an image inline, use the `image` prefix instead (notice there is only one colon):

```
Click image:icons/play.png[Play, title="Play"] to get the party started.

Click image:icons/pause.png[title="Pause"] when you need a break.
```
ASCIIDOC

Click  to get the party started.

Click  when you need a break.

For inline images, the optional title is displayed as a tooltip.

30.1. Set the images directory

Images are resolved relative to the value of the `imagesdir` document attribute, which defaults to an empty value. The imagesdir attribute can be an absolute path, relative path or base URL. If the image target is a URL or an absolute path, the imagesdir prefix is *not* added.

You should use the imagesdir attribute to avoid hard coding the shared path to your images in every image macro.

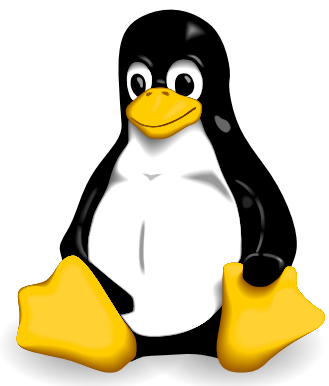
30.1.1. Include images by full URL

Asciidoctor supports remote (i.e., images with a URL target) block and inline images. You can reference images served from any URL (e.g., your blog, an image hosting service, your docs server, etc.) and never have to worry about downloading the images and putting them somewhere locally.

Here are a few examples of images that have a URL target:

Block image with a URL target

```
image::http://upload.wikimedia.org/wikipedia/commons/3/35/Tux.svg[Tux,250,350]
```
ASCIIDOC



Inline image with a URL target

```
You can find image:http://upload.wikimedia.org/wikipedia/commons/3/35/Tux.svg[Linux,25,35] everywhere these days.
```
ASCIIDOC

You can find  everywhere these days.

The value of imagesdir is ignored when the image target is a URI.

If you want to avoid typing the URL prefix for every image, and all the images are located on the same server, you can use the `imagesdir` attribute to set the base URL:

Using a URL as the base URL for images

```
:imagesdir-old: {imagesdir}
:imagesdir: http://upload.wikimedia.org/wikipedia/commons

image::3/35/Tux.svg[Tux,250,350]

:imagesdir: {imagesdir-old}
```

ASCIIDOC

This time, the imagesdir *is* used since the image target is not a URL (the imagesdir just happens to be one).

This feature is included in the AsciiDoc compatibility file so that AsciiDoc gets it right too.

30.2. Put images in their place

Images are a great way to enhance the text, whether its 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

30.2.1. Positioning attributes

Asciidoctor supports the `align` attribute on block images to align the image within the block (e.g., left, right or center). The named attribute `float` can be applied to both the block and inline image macros. Float pulls the image to one side of the page or the other and wraps block or inline content around it, respectively.

Here's an example of a floating block image. The paragraphs or other blocks that follow the image will float up into the available space next to the image. The image will also be positioned horizontally in the center of the image block.

A block image pulled to the right and centered within the block

```
image::tiger.png[Tiger,200,200,float="right",align="center"]
```

ASCIIDOC

Here's an example of a floating inline image. The image will float into the upper-right corner of the paragraph text.

An inline image pulled to the right of the paragraph text

```
image:linux.png[Linux,150,150,float="right"]
You can find Linux everywhere these days!
```

ASCIIDOC

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.

30.2.2. Positioning roles

Here are the examples from above, now configured to use roles that map to CSS classes in the default Asciidoctor stylesheet:

Block image macro using positioning roles

```
[.right.text-center]
image::tiger.png[Tiger,200,200]
```

ASCIIDOC

Inline image macro using positioning role

```
image:sunset.jpg[Sunset,150,150,role="right"] What a beautiful sunset!
```

ASCIIDOC

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	✔	✔	✔	✔	✔
Inline Image	✔	✔	✖	✖	✖

Merely setting the float direction on an image is not sufficient for proper positioning. That's because, by default, no space is left between the image and the text. To alleviate this problem, we've added sensible margins to images that use either the positioning named attributes or roles.

If you want to customize the image styles, perhaps to customize the margins, you can provide your own additions to the stylesheet (either by using your own stylesheet that builds on the default stylesheet or by adding the styles to a docinfo file).

The shorthand syntax for a role (`.`) can not yet be used with image styles.

30.2.3. Framing roles

It's common to frame the image in a border to further offset it from the text. You can style any block or inline image to appear as a thumbnail using the `thumb` role (or `th` for short).

The `thumb` role doesn't alter the dimensions of the image. For that, you need to assign the image a height and width.

Here's a common example for adding an image to a blog post. The image floats to the right and is framed to make it stand out more from the text.

```
image:logo.png[role="related thumb right"] Here's text that will wrap around the image to the left.
```

ASCIIDOC

Notice we added the `related` role to the image. This role isn't technically required, but it gives the image semantic meaning.

30.2.4. Control the float

When you start floating images, you may discover that too much content is floating around the image. What you need is a way to clear the float. That is provided using another role, `float-group`.

Let's assume that we've floated two images so that they are positioned next to each other and we want the next paragraph to appear below them.

```
[.left]
.Image A
image::a.png[A,240,180]

[.left]
.Image B
image::b.png[B,240,180,title="Image B"]

Text below images.
```

ASCIIDOC

When this example is rendered and viewed a browser, the paragraph text appears to the right of the images. To fix this behavior, you just need to "group" the images together in a block with self-contained floats. Here's how it's done:

```
[.float-group]
--
[.left]
.Image A
image::a.png[A,240,180]

[.left]
.Image B
image::b.png[B,240,180]
--

Text below images.
```

ASCIIDOC

This time, the text will appear below the images where we want it.

30.3. Summary

Block and inline image attributes and values

Attribute	Value(s)	Example Syntax	Commer
`imagesdir`	empty, absolute path, relative path or base URL	`:imagesdir: /file`	If the ima target is a URL or a absolute path, the `imagesd` prefix is added; Default is empty value
`id`	User defined text	or `image::sunset.jpg[Sunset,id=sunset-img]`	
`alt`	User defined text in first position of attribute list or named attribute	`image::sunset.jpg[Brilliant sunset]` or `image::sunset.jpg[align=left,alt=Sunset]`	
`title`	User defined text	`.A mountain sunset` or `image::sunset.jpg[Sunset,title=A mountain sunset]`	Blocks: ti displayed below image; Inline: tit displayed as tooltip
`caption`	User defined text	`[caption="Figure 8: "]`	Insert below blo title and above image macro; M

			only appl to block images
`width`	User defined size in pixels	`image::sunset.jpg[Sunset, width=300]`	
`height`	User defined size in pixels	`image::sunset.jpg[Sunset,height=200]`	
`link`	User defined location of external file	`image::sunset.jpg[Sunset,link="http://www.flickr.com/photos/javh/5448336655"]`	
`scaledwidth`	User defined percentage by which to scale image's width	`image::sunset.jpg[Sunset,scaledwidth=25%]`	Only applies to block images rendered DocBook and then converte to PDF
`scale`	TBD	TBD	DocBook only
`align`	left, center, right	`image::sunset.jpg[Sunset,align=left]`	Block images only; Alig and float attributes are mutually exclusive
`float`	left, right	`image::sunset.jpg[Sunset,float=right]`	Block images only; floa and align attributes are mutually exclusive To stop a floating image, us `unfloat` `[]`
`role`	left, right, th, thumb, related, rel	`image::foo.png[role="thumb right"]`	Inline images ca use role t float images le

			and right Role shorthan (.) can n be used c images

31. Video

The `video` macro supports displaying videos stored relatively as well as externally.

```
video::video_file.mp4[]
```
ASCIIDOC

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.

```
video::video_file.mp4[width=640, start=60, end=140, options=autoplay]
```
ASCIIDOC

31.1. YouTube and Vimeo videos

The video macro supports videos from external video hosting services like YouTube and Vimeo. To use this feature, put the video ID in the macro target and the name of the hosting service in the first positional attribute. Asciidoctor will generate the correct embed code and insert the video in the HTML output.

Embedding a YouTube video

```
video::rPQoq7ThGAU[youtube]
```
ASCIIDOC

Embedding a Vimeo video

```
video::67480300[vimeo]
```
ASCIIDOC

31.2. Summary

Video attributes and values

Attribute	Value(s)	Example Syntax	Comments
`title`	User defined text	`.An ocean sunset`	
`poster`	User defined URL or file of a video	`video::ocean_sunset.mp4[poster=sunset.jpg]`	
`width`	User defined size in pixels	`video::ocean_sunset.mp4[width=400]`	
`height`	User defined size in pixels	`video::ocean_sunset.mp4[height=400]`	
`options`	autoplay, loop, controls, nocontrols	`video::ocean_sunset.mp4[options=loop]`	The controls value is enabled by default
`start`	User defined playback start time in seconds	`video::ocean_sunset.mp4[start=30]`	
`end`	User defined playback end time in seconds	`video::ocean_sunset.mp4[end=90]`	

32. Audio

Pending

32.1. Summary

Audio attributes and values

Attribute	Value(s)	Example Syntax	Comments
`options`	`autoplay`, `loop`, `controls`, `nocontrols`	`audio::ocean_waves.mp3[options="autoplay,loop"]`	The controls value is enabled by default

33. Admonition

There are certain statements that you may want to draw attention to by taking them out of the content's flow and labeling them with a priority. These are called admonitions. It's rendered style is determined by the assigned label (i.e., value). Asciidoctor provides five admonition style labels:

- `NOTE`
- `TIP`
- `IMPORTANT`
- `CAUTION`
- `WARNING`

When you want to call attention to a single paragraph, start the first line of the paragraph with the label you want to use. The label must be uppercase and followed by a colon (`:`).

Admonition paragraph syntax

```asciidoc
WARNING: Wolpertingers are known to nest in server racks. (1) (2)
Enter at your own risk.
```

(1) The label must be uppercase and immediately followed by a colon (`:`).

(2) Separate the first line of the paragraph from the label by a single space.

Result: Admonition paragraph

Wolpertingers are known to nest in server racks. Enter at your own risk.

When you want to apply an admonition to complex content, set the label as a style attribute on a block. As seen in the next example, admonition labels are commonly set on example blocks. This behavior is referred to as **masquerading**. The label must be uppercase when set as an attribute on a block.

Admonition block syntax

```asciidoc
[IMPORTANT] (1)
.Feeding the Werewolves
==== (2)
While werewolves are hardy community members, keep in mind the following dietary concerns:

. They are allergic to cinnamon.
. More than two glasses of orange juice in 24 hours makes them howl in harmony with alarms and sirens.
. Celery makes them sad.
====
```

(1) Set the label in an attribute list on a delimited block. The label must be uppercase.

(2) Admonition styles are commonly set on example blocks. Example blocks are delimited by four equal signs (`====`).

Result: Admonition block

Feeding the Werewolves

While werewolves are hardy community members, keep in mind the following dietary concerns:

1. They are allergic to cinnamon.
2. More than two glasses of orange juice in 24 hours makes them howl in harmony with alarms and sirens.
3. Celery makes them sad.

In the examples above, the admonition is rendered in a callout box with the style label in the gutter. You can replace the rendered labels with icons by setting the `icons` attribute on the document. This is how the WARNING admonition paragraph renders when icons is set and assigned the `font` value.

Admonition paragraph with icons set

```
WARNING: Wolpertingers are known to nest in server racks.
Enter at your own risk.
```
ASCIIDOC

Result: Admonition paragraph with icons set

Wolpertingers are known to nest in server racks. Enter at your own risk.

Learn more about using Font Awesome or custom icons with admonitions in the icons section.

34. Sidebar

Use a sidebar for ancillary content that doesn't fit into the flow of the document's narrative. A sidebar can be titled and contain any type of content such as source code and images.

Sidebar syntax

```asciidoc
.AsciiDoc history (1)
**** (2)
AsciiDoc was first released in Nov 2002 by Stuart Rackham.
It was designed from the start to be a shorthand syntax
for producing professional documents like DocBook and LaTeX.
****
```

(1) A title is optional.

(2) A sidebar is delimited by four asterisks (`**`).

Result: Sidebar with title

AsciiDoc history

AsciiDoc was first released in Nov 2002 by Stuart Rackham. It was designed from the start to be a shorthand syntax for producing professional documents like DocBook and LaTeX.

35. Example

Example syntax

```asciidoc
.Sample document
====
Here's a sample AsciiDoc document:

[listing]
....
= Title of Document
Doc Writer
:toc:

This guide provides...
....

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

Result: Example block

Sample document

Here's a sample AsciiDoc document:

```
= Title of Document
Doc Writer
:toc:

This guide provides...
```

The document header is useful, but not required.

36. Prose Excerpts, Quotes and Verses

Prose excerpts, quotes and verses share the same syntax structure, including:

- block name, either `quote` or `verse`
- name of who the content is attributed to
- biblographical information of the book, speech, play, poem, etc., where the content was drawn from
- excerpt text

36.1. Quote

For content that doesn't require the preservation of line breaks, set the `quote` attribute in the first position of the attribute list. Next, set the attribution and relevant citation information. However, these positional attributes are optional.

Anatomy of a basic quote

```
[quote, attribution, citation title and information]
Quote or excerpt text
```

If the quote is a single line or paragraph, you can place the attribute list directly on top of the text.

Quote paragraph syntax

```
.After landing the cloaked Klingon bird of prey in Golden Gate park: (1)
[quote, Captain James T. Kirk, Star Trek IV: The Voyage Home] (2) (3) (4)
Everybody remember where we parked. (5)
```

(1) Mark lead-in text explaining the context or setting of the quote using a period (`.`). (optional)

(2) For content that doesn't require the preservation of line breaks, set `quote` in the first position of the attribute list.

(3) The second position contains who the excerpt is attributed to. (optional)

(4) Enter additional citation information in the third position. (optional)

(5) Enter the excerpt or quote text on the line immediately following the attribute list.

Result: Quote paragraph

After landing the cloaked Klingon bird of prey in Golden Gate park:

> *Everybody remember where we parked.*
>
> — *Captain James T. Kirk*
> *Star Trek IV: The Voyage Home*

If the quote or excerpt is more than one paragraph, place the text between delimiter lines consisting of four underscores (`____`).

Quote block syntax

```
[quote, Monty Python and the Holy Grail]
____
Dennis: Come and see the violence inherent in the system. Help! Help! I'm being repressed!

King Arthur: Bloody peasant!

Dennis: Oh, what a giveaway! Did you hear that? Did you hear that, eh? That's what I'm on about! Did you see him repressing
____
```

Result: Quote block

> *Dennis: Come and see the violence inherent in the system. Help! Help! I'm being repressed!*
>
> *King Arthur: Bloody peasant!*
>
> *Dennis: Oh, what a giveaway! Did you hear that? Did you hear that, eh? That's what I'm on about! Did you see him repressing me? You saw him, Didn't you?*
>
> — *Monty Python and the Holy Grail*

Asciidoctor also provides three alternative ways to markup quotes and prose excerpts.

36.1.1. Quoted paragraph

You can turn a single paragraph into a blockquote by:

1. surrounding it with double quotes
2. adding an optional attribution (prefixed with two dashes) below the quoted text

Quoted paragraph syntax

```
"I hold it that a little rebellion now and then is a good thing,
and as necessary in the political world as storms in the physical."
-- Thomas Jefferson, Papers of Thomas Jefferson: Volume 11
```
ASCIIDOC

Result: Quoted paragraph

> *I hold it that a little rebellion now and then is a good thing, and as necessary in the political world as storms in the physical.*
>
> — *Thomas Jefferson*
> *Papers of Thomas Jefferson: Volume 11*

36.1.2. Air quotes

Air quotes (http://en.wikipedia.org/wiki/Air_quotes) are two double quotes on each line, emulating the gesture of making quote marks with two fingers on each hand.

```
[, Richard M. Nixon]
""
When the President does it, that means that it's not illegal.
""
```
ASCIIDOC

Result: Air quotes

> *When the President does it, that means that it's not illegal.*
>
> — *Richard M. Nixon*

36.1.3. Markdown-style blockquotes

Asciidoctor supports Markdown-style blockquotes.

Markdown-style blockquote syntax

```
> I hold it that a little rebellion now and then is a good thing,
> and as necessary in the political world as storms in the physical.
> -- Thomas Jefferson, Papers of Thomas Jefferson: Volume 11
```
MARKDOWN

Result: Markdown-style blockquote

> *I hold it that a little rebellion now and then is a good thing, and as necessary in the political world as storms in the physical.*
>
> — *Thomas Jefferson*

Papers of Thomas Jefferson: Volume 11

Like Markdown, Asciidoctor supports block content inside the blockquote, including nested blockquotes.

Markdown-style blockquote containing block content

```markdown
> > What's new?
>
> I've got Markdown in my AsciiDoc!
>
> > Like what?
>
> * Blockquotes
> * Headings
> * Fenced code blocks
>
> > Is there more?
>
> Yep. AsciiDoc and Markdown share a lot of common syntax already.
```

Here's how this conversation renders.

Result: Markdown-style blockquote with block content

> *What's new?*
>
> *I've got Markdown in my AsciiDoc!*
>
> > *Like what?*
>
> - *Blockquotes*
> - *Headings*
> - *Fenced code blocks*
>
> > *Is there more?*
>
> *Yep. AsciiDoc and Markdown share a lot of common syntax already.*

36.2. Verse

When you need to preserve indents and line breaks, use the `verse` block name. Verses are defined by setting `verse` on a paragraph or an excerpt block delimited by four underscores (`____`).

Verse paragraph syntax

```asciidoc
[verse, Carl Sandburg, two lines from the poem Fog]
The fog comes
on little cat feet.
```

Result: Verse paragraph

> The fog comes
> on little cat feet.
>
> — *Carl Sandburg*
> *two lines from the poem Fog*

When the verse content includes blank or indented lines, enclose it in an excerpt block.

Verse delimited block syntax

```
[verse, Carl Sandburg, Fog]
____
The fog comes
on little cat feet.

It sits looking
over harbor and city
on silent haunches
and then moves on.
____
```

Result: verse block

> The fog comes
> on little cat feet.
>
> It sits looking
> over harbor and city
> on silent haunches
> and then moves on.
>
> — *Carl Sandburg*
> *Fog*

37. Comments

Line

```
// A single-line comment.
```

Single-line comments can be used to divide elements, such as two adjacent lists.

Block

```
////
A multi-line comment.

Notice it's a delimited block.
////
```

Controlling Your Content

Part introduction pending

38. Text Substitutions

Text substitution elements replace characters, markup, attribute references, and macros with converter specific styles and values. When Asciidoctor processes a document it uses a set of six text substitution elements. The processor runs the text substitution elements in the following order.

1. Special characters
2. Quotes
3. Attribute references
4. Replacements
5. Inline macros
6. Post replacements

In turn, these substitutions are organized into composite value groups. The table below shows which substitution elements are included in each group.

Substitution groups

Group	Special characters	Quotes	Attributes	Replacements	Macros	Post replacements
Header	✔	✖	✔	✖	✖	✖
None	✖	✖	✖	✖	✖	✖
Normal	✔	✔	✔	✔	✔	✔
Pass	✖	✖	✖	✖	✖	✖
Verbatim	✔	✖	✖	✖	✖	✖

By default, the `normal` substitution group is applied to most block and inline elements. However, there are a few exceptions.

The `header` substitution group is applied to the header of your document. In the header, only special characters and attribute references are replaced.

Fenced, literal, listing, and source blocks are processed with the `verbatim` substitution group. Only special characters are replaced in these blocks.

The `pass` substitution group can only be applied to passthrough elements. Attribute references and macros are replaced in passthroughs.

The `none` substitution group is applied to comment blocks.

The title substitution group

The `title` substitution group includes the same text substitutions as the normal group. However, the order that the substitutions are executed is slightly different. Text substitutions are applied to titles in the following sequence:

1. Special characters
2. Quotes
3. Replacements
4. Inline macros
5. Attribute references
6. Post replacements

38.1. Special characters

When applicable, the first text substitution to occur is the replacement of any special characters. This process is handled by the `specialchars` element. The specialchars element searches for three characters (`<`, `>`, `&`) and replaces them with their character entity references.

- The less than symbol, `<`, is replaced with the `<` character entity reference.
- The greater than symbol, `>`, is replaced with the `>` character entity reference.
- An ampersand, `&`, is replaced with the `&` character entity reference.

By default, the special characters substitution occurs on all inline and block elements except for comments and certain passthroughs. The substitution of special characters can be controlled on blocks using the subs attribute and on inline elements using the passthrough macro.

Special character substitution precedes attribute substitution, so you will need to manually escape any attributes containing special characters that you set in the CLI or API. For example, on the command line, type `-a toctitle="Sections, Tables & Figures"` instead of `-a toctitle="Sections, Tables & Figures"`.

38.2. Quotes

The `quotes` substitution replaces the formatting markup on inline elements.

For example, when a document is rendered to HTML, any asterisks enclosing text are replaced with `<strong>` HTML tags.

Syntax input

```
Happy werewolves are *really* slobbery.
```

HTML output

```
Happy werewolves are <strong>really</strong> slobbery.
```

The following table shows the markup syntax that is replaced by the quotes substitution process.

HTML tags that replace AsciiDoc markup syntax

Name	Syntax	Tag
emphasis	_word_	`<em>word</em>`
strong	*word*	`<strong>word</strong>`
monospaced	`word`	`<code>word</code>`
superscript	^word^	`<sup>word</sup>`
subscript	~word~	`<sub>word</sub>`
double curved quotes	"`word`"	`&#8220;word&#8221;`
single curved quotes	'`word`'	`&#8216;word&#8217;`

The quotes substitution occurs on formatted text within title, paragraph, example, quote, sidebar, and verse blocks.

Elements subject to quotes text substitution

Element	`quotes` substitution
Attribute value	✖
Comment	✖
Example	✔
Fenced	✖

Header	✗
Literal	✗
Listing	✗
Macro	✓
Open	Varies
Paragraph	✓
Passthrough	✗
Quote	✓
Sidebar	✓
Source	✗
Special sections	✓
Table	Varies
Title	✓
Verse	✓

38.3. Attributes

Attribute references are replaced with their values when they're processed by the `attributes` substitution.

Elements subject to attributes text substitution

Element	`attributes` **substitution**
Attribute value	✓
Comment	✗
Example	✓
Fenced	✗
Header	✓
Literal	✗
Listing	✗
Macro	✓
Open	Varies
Paragraph	✓
Passthrough	✗
Quote	✓
Sidebar	✓
Source	✗
Special sections	✓

Table	Varies
Title	✔
Verse	✔

38.4. Replacements

The replacements substitution processes textual characters such as marks, arrows and dashes and replaces them with the decimal format of their Unicode code point, i.e. their numeric character reference.

Textual symbol replacements

Name	Syntax	Unicode Replacement	Rendered	Notes
Copyright	`(C)`	`&#169;`	©	
Registered	`(R)`	`&#174;`	®	
Trademark	`(TM)`	`&#8482;`	™	
Em dash	`--`	`&#8212;`	—	When space is detected on either side of the em dash, the thin space numeric character entity ( ) is also substituted into the document.
ellipses	`...`	`&#8230;`	…	
right single arrow	`->`	`&#8594;`	→	
right double arrow	`=>`	`&#8658;`	⇒	
left single arrow	`<-`	`&#8592;`	←	
left double arrow	`<=`	`&#8656;`	⇐	
apostrophe	`Sam's`	`Sam&#8217;s`	Sam’s	The vertical form apostrophe is replaced with the curved form apostrophe.

The `replacements` element depends on the substitutions completed by the `specialcharacters` element. This is important to keep in mind when applying custom substitutions to a block. See the section about applying custom substitutions for more information.

The replacements substitution also recognizes HTML and XML character entity references (http://en.wikipedia.org/wiki/List_of_XML_and_HTML_character_entity_references) as well as decimal and hexadecimal Unicode code points (http://en.wikipedia.org/wiki/List_of_Unicode_characters) and substitutes them for their corresponding decimal form Unicode code point.

For example, to render the `§` symbol you could write `&sect;`, `&#x00A7;`, or `&#167;`. When the document is processed, `replacements` will replace the section symbol reference, regardless of whether it is a character entity reference or a numeric character reference, with `&#167;`. In turn, `&#167;` will render as `§`.

Anatomy of a character entity reference and a numeric character reference

A character reference is a standard sequence of characters that is substitued for a single character when Asciidoctor processes a document. There are two types of character references: character entity references and numeric character references.

A character entity reference is the name of an entity which refers to a character. The name must be prefixed with an ampersand (`&`) and end with a semicolon (`;`).

For example:

- `&dagger;` renders as †
- `&euro;` renders as €
- `&loz;` renders 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:

- `&#x2020;` or `&#8224;` render as †
- `&#x20AC;` or `&#8364;` render as €
- `&#x25CA;` or `&#9674;` render as ◊

Developers may be more familiar with using **Unicode escape sequences** to perform text substitutions. For example, to render an `@` sign using a Unicode escape sequence, you would prefix the hexadecimal Unicode code point with a backslash (`\`) and an uppercase or lowercase `u` , i.e. `u0040` . However, Asciidoctor doesn't process and replace Unicode escape sequences at this time.

Asciidoctor also provides numerous built-in attributes for representing characters and symbols. These attributes and their corresponding output are listed in Appendix B.

The replacements substitution occurs within title, paragraph, example, quote, sidebar, and verse blocks.

Elements subject to replacements text substitution

Element	`replacements` **substitution**
Attribute value	✗
Comment	✗
Example	✓
Fenced	✗
Header	✗
Literal	✗
Listing	✗
Macro	✓
Open	Varies
Paragraph	✓
Passthrough	✗
Quote	✓
Sidebar	✓

Source	✗
Special sections	✓
Table	Varies
Title	✓
Verse	✓

38.5. Macros

Macros are processed by the `macros` element. The macros substitution replaces a macro's content with the appropriate built-in and user-defined configuration.

Elements subject to macros substitution

Element	`macros` **substitution**
Attribute value	✓
Comment	✗
Example	✓
Fenced	✗
Header	✓
Literal	✗
Listing	✗
Macro	✓
Open	Varies
Paragraph	✓
Passthrough	✓
Quote	✓
Sidebar	✓
Source	✗
Special sections	✓
Table	Varies
Title	✓
Verse	✓

38.6. Post replacements

The line break character, `+`, is replaced when the `post_replacements` process runs.

Elements subject to post replacements text substitution

Element	`post_replacements` **substitution**
Attribute value	✓

Comment	✖
Example	✔
Fenced	✖
Header	✔
Literal	✖
Listing	✖
Macro	✔
Open	Varies
Paragraph	✔
Passthrough	✖
Quote	✔
Sidebar	✔
Source	✖
Special sections	✔
Table	Varies
Title	✔
Verse	✔

38.7. Applying substitutions

Specific substitution elements can be applied to any block or paragraph by setting the `subs` attribute. The `subs` attribute can be assigned a comma separated list of the following substitution elements and groups.

`none`	Disables substitutions
`normal`	Performs all substitutions except for callouts
`verbatim`	Replaces special characters and processes callouts
`specialchars`, `specialcharacters`	Replaces `<`, `>`, and `&` with their corresponding entities
`quotes`	Applies text formatting
`attributes`	Replaces attribute references
`replacements`	Substitutes textual and character reference replacements
`macros`	Processes macros
`post_replacements`	Replaces the line break character (`+`)

Let's look at an example where you only want to process special characters, formatting markup, and callouts in a literal block. By default, literal blocks are only subject to special characters substitution. But you can change this behavior by setting the `subs` attribute in the block's attribute list.

```
[source,java,subs="verbatim,quotes"] (1)
----
System.out.println("Hello *bold* text"). (2)
----
```

(1) The subs attribute is set in the attribute list and assigned the `verbatim` and `quotes` values.

(2) The formatting markup in this line will be replaced when the quotes substitution runs.

```java
System.out.println("Hello bold text"). (1) (2)
```

(1) The `verbatim` value enables the callouts to be rendered.

(2) The `quotes` value enables the text formatting to be rendered.

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 `macros`, only macros are substituted. However, Asciidoctor also provides syntax to append or remove substitutions to a block's default substitutions.

You can add or remove a substitution from the default substitution list using the plus (`+`) and minus (`-`) modifiers.

`<substitution>+`	Prepends the substitution to the default list.
`+<substitution>`	Appends the substitution to the default list.
`-<substitution>`	Removes the substitution from the default list.

For example, you can add the `attributes` substitution to a listing block's default substitution list.

Add attributes substitution to a default substitution list

```asciidoc
[source,xml,subs="attributes+"]
----
<version>{version}</version>
----
```

Similarly, you can remove the `callouts` substitution.

Remove callouts substitution from a default substitution list

```asciidoc
[source,xml,subs="-callouts"]
.An illegal XML tag
----
<1>foo</1>
----
```

You can also specify whether the substitution is placed at the beginning or end of the substitution list. If a `+` comes before the name of the substitution, then it's added to the end of the existing list, and if a `+` comes after the name, it's added to the beginning of the list.

```
[source,xml,subs="attributes+,+replacements,-callouts"]
----
<version>{version}</version>
<copyright>(C) ACME</copyright>
(1)
content in 1 element
</1>
----
```

In the above example, the quotes, then the special characters, and then the attributes substitutions will be applied to the listing block.

If you are applying the same set of substitutions to numerous blocks, you should consider making them an attribute entry to ensure consistency.

```
:markup-in-source: verbatim,quotes

[source,java,subs="{markup-in-source}"]
----
System.out.println("Hello *bold* text").
----
```

Another way to ensure consistency and keep your documents clean and simple is to use the Treeprocessor extension.

Custom substitutions can also be applied to some inline elements, such as the pass macro.

For example, the quotes text substitution value is assigned in the inline pass macro below.

```
The markup pass:q[<u>underline *me*</u>] renders as underlined text and "`me`" is bold.
```

The markup <u>underline **me**</u> renders as underlined text and “me” is bold.

38.8. Preventing substitutions

Asciidoctor provides several approaches for preventing substitutions.

Backslash escaping

To prevent punctuation from being interpreted as formatting markup, precede it with a backslash (`\`). If the formatting punctuation begins with two characters (e.g., `__`), you need to precede it with two backslashes (`\\`). This is also how you can prevent character and attribute references from substitution. When your document is processed, the backslash is removed so it doesn’t display in your output.

```
\*Stars* will appear as *Stars*, not as bold text.

\&sect; will appear as an entity reference, not the &sect; symbol.

\\__func__ will appear as __func__, not as emphasized text.

\{two-semicolons} will appear {two-semicolons}, not resolved as ;;.
```

You can also prevent substitutions with macro and block passthroughs.

39. Literal Text and Blocks

Literal paragraphs and blocks display the text you write exactly as you enter it. Literal text is treated as preformatted text. The text is shown in a fixed-width font and endlines are preserved. Only special characters and callouts are replaced when the document is rendered.

Literal blocks are defined three ways:

1. Indenting the first line of a paragraph by one or more spaces
2. Applying the `literal` attribute to a paragraph or block
3. Using the literal block delimiter (`....`)

When a line begins with one or more spaces it is displayed as a literal paragraph. This method is a quick and easy way to insert code snippets.

Implicit literal text

```
 ~/secure/vault/defops
```
ASCIIDOC

Result: Implicit literal text

```
~/secure/vault/defops
```

When you want an entire block of text to be literal and would prefer not to indent it, set the `literal` attribute on top of the element.

Literal style paragraph syntax

```
[literal]
error: The requested operation returned error: 1954 Forbidden search for defensive operations manual
absolutely fatal: operation initiation lost in the dodecahedron of doom
would you like to die again? y/n
```
ASCIIDOC

Result: Literal style paragraph

```
error: The requested operation returned error: 1954 Forbidden search for defensive operations manual
absolutely fatal: operation initiation lost in the dodecahedron of doom
would you like to die again? y/n
```

Finally, you can surround the content you want rendered as literal by enclosing it in a set of literal block delimiters (`....`). This method is useful when the content consists of several elements that are separated by blank lines.

Literal delimited block syntax

```
....
Lazarus: Where is the *defensive operations manual*?

Computer: Calculating ...
Can not locate object that you are not authorized to know exists.
Would you like to ask another question?

Lazarus: Did the werewolves tell you to say that?

Computer: Calculating ...
....
```
ASCIIDOC

Notice in the output that the bold text formatting is not rendered nor are the three, consecutive periods replaced by the ellipsis Unicode character.

Result: Literal delimited block

```
Lazarus: Where is the *defensive operations manual*?

Computer: Calculating ...
Can not locate object that you are not authorized to know exists.
Would you like to ask another question?

Lazarus: Did the werewolves tell you to say that?

Computer: Calculating ...
```

40. Listing Blocks

Like literal blocks, the content in listing blocks is displayed exactly as you entered it. Listing block content is rendered as `<pre>` text. The content in listing blocks is only subject to special character and callout substitutions.

The `listing` block name can be applied to content two ways.

1. Set the `listing` attribute on the element.
2. Contain the content within a delimited listing block.

The listing block name is applied to an element, such as a paragraph, by setting the `listing` attribute on that element.

Listing paragraph syntax

```
[listing]
This is an example of a paragraph styled with `listing`.
Notice that the monospace markup is preserved in the output.
```

ASCIIDOC

Result: Listing paragraph

```
This is an example of a paragraph styled with `listing`.
Notice that the monospace markup is preserved in the output.
```

A delimited listing block is surrounded by lines composed of four hyphens (`----`).

Delimited listing block syntax

```
----
This is an example of a _listing block_.
The content inside is rendered as <pre> text.
----
```

ASCIIDOC

Here's how the block above appears when rendered as HTML.

Result: Listing block

```
This is an example of a _listing block_.
The content inside is rendered as <pre> text.
```

You should notice a few things about how the content is processed.

- the HTML tag `<pre>` is escaped
- then endlines are preserved
- the phrase *listing block* is not italicized, despite having underscores around it.

Listing blocks are good for displaying raw source code, especially when used in tandem with the `source` and `source-highlighter` attributes. The example below shows a listing block with `source` and the language `ruby` applied to its content.

Source block syntax

```
[[app-listing]]
[source,ruby]
.app.rb
----
require 'sinatra'

get '/hi' do
  "Hello World!"
end
----
```

ASCIIDOC

Result: Listing block with the source attribute set

```
This is an example of a _listing block_.
The content inside is rendered as <pre> text.
```

Detailed instructions for using the source and source-highlighter attributes are provided in the source code blocks section.

Listing block with custom substitutions syntax

```
:version: 0.1.4

[source,xml,subs="verbatim,attributes"]
----
<dependency>
  <groupId>org.asciidoctor</groupId>
  <artifactId>asciidoctor-java-integration</artifactId>
  <version>{version}</version>
</dependency>
----
```

Result: Listing block with custom substitutions applied

```xml
<dependency>
  <groupId>org.asciidoctor</groupId>
  <artifactId>asciidoctor-java-integration</artifactId>
  <version>0.1.4</version>
</dependency>
```

40.1. To wrap or to scroll

The default Asciidoctor stylesheet wraps long lines in listing and literal blocks by applying the CSS `white-space: pre-wrap` and `word-wrap: break-word`. The lines are wrapped at word boundaries, similar to how most text editors wrap lines. This prevents horizontal scrolling which some users considered a greater readability problem than line wrapping.

However, this behavior is configurable because there are times when you don't want the lines in listing and literal blocks to wrap.

There are two ways to prevent lines from wrapping so that horizontal scrolling is used instead:

- `nowrap` block option
- unset the `prewrap` document attribute (on by default)

You can use the `nowrap` option on literal or listing blocks to prevent lines from being wrapped in the HTML.

Listing block with nowrap option syntax

```asciidoc
[source%nowrap,java]
----
public class ApplicationConfigurationProvider extends HttpConfigurationProvider
{
   @Override
   public Configuration getConfiguration(ServletContext context)
   {
      return ConfigurationBuilder.begin()
               .addRule()
               .when(Direction.isInbound().and(Path.matches("/{path}")))
               .perform(Log.message(Level.INFO, "Client requested path: {path}"))
               .where("path").matches(".*");
   }
}
----
```

When the nowrap option is used, the `nowrap` class is added to the `<pre>` element. This class changes the CSS to `white-space: pre` and `word-wrap: normal`.

Result: Listing block with nowrap option applied

```java
public class ApplicationConfigurationProvider extends HttpConfigurationProvider
{
   @Override
   public Configuration getConfiguration(ServletContext context)
   {
      return ConfigurationBuilder.begin()
                .addRule()
                .when(Direction.isInbound().and(Path.matches("/{path}")))
                .perform(Log.message(Level.INFO, "Client requested path: {path}"))
                .where("path").matches(".*");
   }
}
```

To prevent lines from wrapping globally, unset the `prewrap` attribute on the document.

Disable prewrap globally (thus, enabling nowrap)

```asciidoc
:prewrap!:
```

When the prewrap attribute is unset, the `nowrap` class is added to any `<pre>` elements.

Now, you can use the line wrapping strategy that works best for you and your readers.

40.2. Summary

Pending

41. Macro and Block Passthroughs

Passthroughs are the “anything goes” mechanism in Asciidoctor. As its name implies, a passthrough passes its contents directly to the output document. The contents of a passthrough are excluded from all substitutions.

Using passthroughs to pass content without processing can couple your document to a specific output format, such as HTML. In these cases, you can use conditional preprocessor directives to declare passthrough markup for each backend you need to support.

41.1. Passthrough macros

Asciidoctor provides several ways to write a passthrough macro.

Pass macro

The `pass` macro can be applied to inline or block content. You can include a comma-seperated list of substitutions prior to the passthrough content.

Inline pass macro

```
pass:optional-substitution-attribute-1,optional-substitution-attribute-x[content passed directly to the output] followed by
```

Asciidoctor does not implement the block form of the block macro. Instead, you should use a pass block.

To exclude a phrase from substitutions and disable escaping of special characters, enclose it in the inline pass macro:

```
The markup pass:[<u>underline me</u>] renders as underlined text.
```
ASCIIDOC

The markup <u>underline me</u> renders as underlined text.

If you want to enable ad-hoc `quotes` substitution, then assign the `macros` value to `subs` and use the inline pass macro.

```
[subs="verbatim,macros"] (1)
----
I better not contain *bold* or _italic_ text.
pass:quotes[But I should contain *bold* text.] (2)
----
```

1. `macros` is assigned to `subs`, which allows any macros within the block to be processed.
2. The pass macro is assigned the `quotes` value. Text within the square brackets will be formatted.

The inline pass macro does introduce additional markup into the Java source code that makes it invalid in raw form, but the output it produces will be valid when viewed in a viewer (HTML, PDF, etc.).

```
I better not contain *bold* or _italic_ text.
But I should contain bold text.
```

The inline pass macro also accepts shorthand values for specifying substitutions.

- `c` = special characters
- `q` = quotes
- `a` = attributes
- `r` = replacements
- `m` = macros

- `p` = post replacements

For example, the quotes text substitution value is assigned in the inline passthrough macro below:

```
The markup pass:q[<u>underline *me*</u>] renders as underlined text and "`me`" is bold.
```
ASCIIDOC

The markup <u>underline **me**</u> renders as underlined text and “me” is bold.

Triple-plus passthrough

The triple-plus passthrough works the same way as the pass macro. To exclude content from substitutions, enclose it in triple pluses (+++).

```
+++content passed directly to the output+++ followed by normal content.
```

The triple-plus macro is often used to output custom HTML or XML.

```
The markup +++<u>underline me</u>+++ renders as underlined text.
```
ASCIIDOC

The markup <u>underline me</u> renders as underlined text.

41.2. Block passthroughs

The `pass` style and delimited passthrough block exclude blocks of content from all substitutions.

The pass style can be set on a paragraph or an open block.

```
[pass]
<u>underline me</u> renders as underlined text.
```
ASCIIDOC

A block passthrough is delimited by four plus signs (`++++`).

```
++++
<video poster="images/movie-reel.png">
  <source src="videos/writing-zen.webm" type="video/webm">
</video>
++++
```
ASCIIDOC

If you want substitutions to be performed on the content in a delimited passthrough block, you can add them using the `subs` attribute.

```
[subs="attributes"]
++++
{name}
image:tiger.png[]
++++
```
ASCIIDOC

42. Open Blocks

The most versatile block of all is the open block.

Open block syntax

```
--
An open block can be an anonymous container,
or it can masquerade as any other block.
--
```

Result: Open block

An open block can be an anonymous container, or it can masquerade as any other block.

An open block can act as any other block, with the exception of *pass* and *table*. Here's an example of an open block acting as a sidebar.

Open block masquerading as a sidebar syntax

```
[sidebar]
.Related information
--
This is aside text.

It is used to present information related to the main content.
--
```

Result: Open block masquerading as a sidebar

Related information

This is aside text.

It is used to present information related to the main content.

This is an open block acting as a source block.

Open block masquerading as a source block syntax

```
[source]
--
puts "I'm a source block!"
--
```

Result: Open block masquerading as a source block

```
puts "I'm a source block!"
```

Enriching Your Content

Part introduction pending

43. Equations and Formulas

If you need to get technical in your writing, Asciidoctor integrates with MathJax (http://www.mathjax.org). MathJax is the standard library for displaying Science, Technology, Engineering and Math (STEM) expressions in the browser.

Thanks to MathJax JavaScript library (http://www.mathjax.org), Asciidoctor supports both AsciiMathML (http://docs.mathjax.org/en/latest/asciimath.html) and TeX and LaTeX (http://docs.mathjax.org/en/latest/tex.html) math notation in the same document.

44. Activating stem support

To activate equation and formula support, simply set the `stem` attribute in the document's header (or by passing the attribute to the commandline or API).

Setting the stem attribute

```
= My Diabolical Mathmatical Opus
Jamie Moriarty
:stem: (1)
```

(1) The default interpreter value, `asciimath`, is assigned implicitly.

By default, Asciidoctor's stem support assumes all equations are AsciiMath if not specified explicitly. If you want to use the LaTeX interpreter by default, assign `latexmath` to the stem attribute.

Assigning an alternative interpreter to the stem attribute

```
= My Diabolical Mathmatical Opus
Jamie Moriarty
:stem: latexmath
```

You can still use both interpreters in the same document. The value of the `stem` attribute merely sets the default interpreter. To set the interpreter explicitly for a given block or inline span, just use `asciimath` or `latexmath` in place of `stem` as explained in Using multiple stem interpreters.

Stem content can be displayed inline with other content or as discrete blocks. No substitutions are applied to the content within a stem macro or block.

44.1. Inline stem content

The best way to mark up an inline formula is to use the `stem` macro.

Inline stem macro syntax

```
stem:[sqrt(4) = 2] (1) (2)

Water (stem:[H_2O]) is a critical component.
```

(1) The inline stem macro contains only one colon (`:`).

(2) Place the content you want interpreted within the square brackets (`[]`) of the macro.

Rendered inline stem content

$\sqrt{4} = 2$

Water (H_2O) is a critical component.

44.2. Block stem content

Block formulas are marked up by assigning the `stem` style to a delimited passthrough block.

Delimited stem block syntax

```
[stem] (1)
++++ (2)
sqrt(4) = 2
++++
```

(1) Assign the stem style to the passthrough block.

(2) A passthrough block is delimited by a line of four consecutive plus signs (`++++`).

The result is rendered beautifully in the browser thanks to MathJax!

Rendered delimited stem block

$\sqrt{4} = 2$

You don't need to add special delimiters around the expression as the MathJax documentation (http://meta.math.stackexchange.com/questions/5020/mathjax-basic-tutorial-and-quick-reference) suggests. Asciidoctor handles that for you automatically!

44.3. Using multiple stem interpreters

You can use multiple interpreters for stem content within the same document by using the interpreter's name instead of the default `stem` name.

For example, if you want LaTeXMath to interpret an inline equation, name the macro `latexmath`.

Inline latexmath macro syntax

```
latexmath:[C = \alpha + \beta Y^{\gamma} + \epsilon]
```

ASCIIDOC

Rendered latexmath content

$C = \alpha + \beta Y^{\gamma} + \epsilon$

The name that maps to the interpreter you want to use can also be applied to block stem content.

Delimited asciimath block syntax

```
= My Diabolical Mathmatical Opus
Jamie Moriarty
:stem: latexmath

[asciimath]
++++
sqrt(4) = 2
++++
```

ASCIIDOC

45. User Interface Macros

We are looking for feedback on these macros before setting them in stone. If you have suggestions, we want to hear from you!

You **must** set the `experimental` attribute to enable these macros.

45.1. Keyboard shortcuts

Asciidoctor recognizes a macro for creating keyboard shortcuts using the syntax kbd:[key(+key)*].

Keyboard macro syntax

```
ASCIIDOC
|===
|Shortcut |Purpose

|kbd:[F11]
|Toggle fullscreen

|kbd:[Ctrl+T]
|Open a new tab

|kbd:[Ctrl+Shift+N]
|New incognito window

|kbd:[Ctrl + +]
|Increase zoom
|===
```

Result: Keyboard macros displaying common browser keyboard shortcuts

Shortcut	Purpose
F11	Toggle fullscreen
Ctrl + T	Open a new tab
Ctrl + Shift + N	New incognito window
Ctrl + +	Increase zoom

You no longer have to struggle to explain to users what keys they are supposed to press.

45.2. Menu selections

Trying to explain to someone how to select a menu item can be a pain. With the `menu` macro, the symbols do the work.

Menu macro syntax

```
ASCIIDOC
To save the file, select menu:File[Save].

Select menu:View[Zoom > Reset] to reset the zoom level to the default setting.
```

The instructions in the example above appear below.

Result: Menu macros displaying menu selections

To save the file, select File ▸ Save.

Select View ▸ Zoom ▸ Reset to reset the zoom level to the default setting.

45.3. UI buttons

It can be equally difficult to communicate to the reader that they need to press a button. They can't tell if you are saying "OK" or they are supposed to look for a button labeled **OK**. It's all about getting the semantics right. The `btn` macro to the rescue!

Button macro syntax

```
Press the btn:[OK] button when you are finished.

Select a file in the file navigator and click btn:[Open].
```
ASCIIDOC

Result: Button macros displaying UI buttons

Press the [**OK**] button when you are finished.

Select a file in the file navigator and click [**Open**].

46. Icons

Asciidoctor provides a number of ways to display icons in a document. The default stylesheet includes admonition and callout icons for the admonition and callout blocks, but you can also specify custom icons. Additionally, Asciidoctor can "draw" icons using the Font Awesome (http://fortawesome.github.io/Font-Awesome/) font-based icon set. You can see the icon name options on the icons page (http://fortawesome.github.io/Font-Awesome//icons/).

When you aren't using font-based icons, or you are using the DocBook backend, the icon set is limited to which icons you have in your `iconsdir` directory.

46.1. Admonition icons

To use this feature, set the value of the `icons` document attribute to `font` in the document header. Asciidoctor will then emit HTML markup that selects an appropriate font character from the Font Awesome font for each admonition block.

Here's an example, starting with the AsciiDoc source:

Set icons attribute in the document header

```
= Document Title
:icons: font

NOTE: Asciidoctor supports font-based admonition icons, powered by Font Awesome!
```

ASCIIDOC

the HTML it produces:

Result: HTML output when the icons attribute is set

```
<div class="admonitionblock note">
<table>
<tr>
<td class="icon">
<i class="fa icon-note" title="Note"></i>
</td>
<td class="content">
Asciidoctor supports font-based admonition icons, powered by Font Awesome!
</td>
</tr>
</table>
</div>
```

XML

This is how the admonition looks rendered.

Result: Admonition block label is displayed as an icon when the icons attribute is set

Asciidoctor supports font-based admonition icons, powered by Font Awesome!

Asciidoctor adds a reference to the Font Awesome stylesheet and font files served from a CDN to the document header:

```
<link rel="stylesheet"
  href="http://cdnjs.cloudflare.com/ajax/libs/font-awesome/3.1.0/css/font-awesome.min.css">
```

XML

The default stylesheet (or any stylesheet produced by the Asciidoctor stylesheet factory (http://github.com/asciidoctor/asciidoctor-stylesheet-factory)) is required for this feature to work.

46.2. Inline icons

An icon can be inserted at an arbitrary place in paragraph content with an inline macro.

Here's an example that inserts the Font Awesome tags icon in front of a list of tag names.

Inline icon macro syntax

```
icon:tags[] ruby, asciidoctor
```

ASCIIDOC

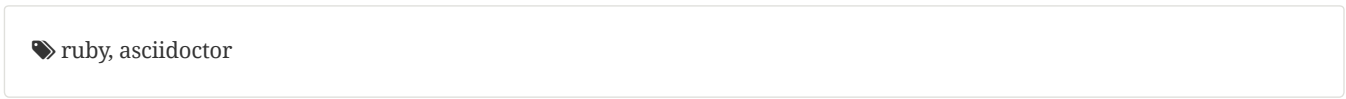
Here's how the HTML converter renders the above syntax when the `icons` attribute is assigned the `font` value.

Result: HTML output

```xml
<div class="paragraph">
<p><span class="icon"><i class="fa fa-tags"></i></span> ruby, asciidoctor</p>
</div>
```

More importantly, here's how it looks!

Result: Inline icon macro

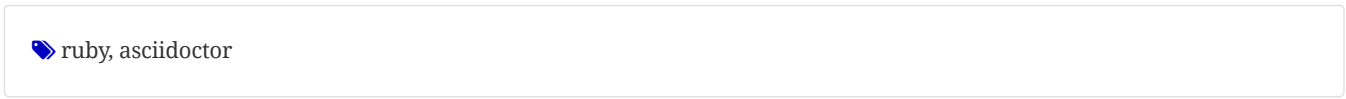


You can even give the icon color by assigning it a role.

Inline icon macro and role syntax

```asciidoc
icon:tags[role="blue"] ruby, asciidoctor
```

Result: Inline icon macro and role



If you aren't using font-based icons, Asciidoctor looks for icon images on disk, in the `iconsdir`, naturally. Here's how the HTML converter renders an icon when the `icons` attribute is not set or empty.

Result: HTML output

```xml
<div class="paragraph">
<p><span class="image"><img src="./images/icons/tags.png" alt="tags"></span> ruby, asciidoctor</p>
</div>
```

Here's how it renders in the DocBook backend, regardless of the icons attribute value.

Result: DocBook output

```xml
<inlinemediaobject>
  <imageobject>
    <imagedata fileref="./images/icons/tags.png"/>
  </imageobject>
  <textobject><phrase>tags</phrase></textobject>
</inlinemediaobject> ruby, asciidoctor
```

Relationship to the inline image macro

The inline icon macro is similar to the inline image macro with a few exceptions:

- If the icons attribute has the value font, the macro will translate to a font-based icon in the HTML converter (e.g., `<i class="icon-tags"></i>`)
- If the icons attribute does not have the value font, or the converter is DocBook, the macro will insert an image into the document that resolves to a file in the iconsdir directory (e.g., `<img src="./images/icons/tags.png">`)

The file resolution strategy when using image-based icons is the same used to locate images for the admonition icons. The file extension is set using the `icontype` attribute, which defaults to PNG (`png`).

46.2.1. Size, rotate, and flip

The icon macro has a few attributes that can be used to modify the size and orientation of the icon. At the moment, these are specific to Font Awesome and therefore only apply to HTML output when icon fonts are enabled.

`size`

First positional attribute; scales the icon; values: `1x` (default), `2x`, `3x`, `4x`, `5x`, `lg`, `fw`

`rotate`

Rotates the icon; values: `90`, `180`, `270`

`flip`
Flips the icon; values: `horizontal`, `vertical`

The first unnamed attribute is assumed to be the size. For instance, to make the icon twice the size as the default, simply add `2x` inside the square brackets.

```
icon:heart[2x]
```
ASCIIDOC

This is equivalent to:

```
icon:heart[size=2x]
```
ASCIIDOC

And this is how the  displays.

The previous example emits the following HTML:

```
<span class="icon"><i class="fa fa-heart fa-2x"></i></span>
```
XML

If you want to line up icons so that you can use them as bullets in a list, use the `fw` size as follows:

```
[%hardbreaks]
icon:bolt[fw] bolt
icon:heart[fw] heart
```

To rotate and flip the icon, specify these options using attributes:

```
icon:shield[rotate=90, flip=vertical]
```
ASCIIDOC

The  looks like this.

The previous example emits the following HTML:

```
<span class="icon"><i class="fa-shield fa-rotate-90 fa-flip-vertical"></i></span>
```
XML

46.2.2. Link and window

Like an inline image, it's possible to add additional metadata to an inline icon.

Below are the possible attributes that apply to both font-based and image-based icons:

`link`
The URI target used for the icon, which will be rendered as a link

`window`
The target window of the link (when the `link` attribute is specified) (HTML converter)

Here's an example of an icon rendered as a link:

```
icon:download[link="http://rubygems.org/downloads/asciidoctor-1.5.2.gem"]
```
ASCIIDOC

The previous example emits the following HTML:

```
<span class="icon"><a class="image" href="http://rubygems.org/downloads/asciidoctor-1.5.2.gem"><i class="fa-download"></i><
```
XML

46.2.3. Regular icon attributes

Below are the possible attributes that apply in the case that font-based icons are **not** in use:

`alt`
: The alternate text on the `<img>` tag (HTML backend) or text for `<inlinemediaobject>` (DocBook converter)

`width`
: The width applied to the image

`height`
: The height applied to the image

`title`
: The title of the image displayed when the mouse hovers over it (HTML converter)

`role`
: The role applied to the element that surrounds the icon

Currently, the inline icon macro doesn't support any options to change it's physical position (such as alignment left or right).

47. Source Code and Syntax Highlighting

Listing blocks and paragraphs are commonly used to display source code. For this reason, Asciidoctor has enabled several syntax highlighting libraries, including CodeRay, highlight.js, Prettify, and Pygments, that you can use to style source code.

To enable syntax highlighting in a document, set the `source-highlighter` attribute in the document header. If you set the attribute in the document, it *must* be defined in the document header.

```
= Document Title
:source-highlighter: <value>
```

For example:

```
= Document Title
:source-highlighter: coderay
```

See the following table for possible values of the `source-highlighter` attribute.

Possible values for the `source-highlighter` attribute

Library Name	Attribute Value	Supported Environments
CodeRay (http://coderay.rubychan.de)	`coderay`	Asciidoctor, AsciidoctorJ
highlight.js (https://highlightjs.org)	`highlightjs`	Asciidoctor, AsciidoctorJ, Asciidoctor.js
Prettify (https://code.google.com/p/google-code-prettify)	`prettify`	Asciidoctor, AsciidoctorJ, Asciidoctor.js
Pygments (http://pygments.org)	`pygments`	Asciidoctor

To apply highlighting to source code, you must assign the `source` name and a source language to a paragraph or delimited block. You can also set the source language globally using the `source-language` attribute.

Code block with title and syntax highlighting

```
[[app-listing]] (1)
[source,ruby] (2) (3)
.app.rb (4)
---- (5)
require 'sinatra'

get '/hi' do
  "Hello World!"
end
----
```

1. An optional ID can be added to the block. See Defining an Anchor.
2. Assign the block name `source` to the first position in the attribute list.
3. Assign a source language to the second position.
4. An optional title can be added to the block.
5. The source block name is typically assigned to listing and literal blocks.

Result: Source block with title and syntax highlighting

app.rb

```ruby
require 'sinatra'

get '/hi' do
  "Hello World!"
end
```

Source paragraph

```
[source,xml] (1)
<meta name="viewport"
  content="width=device-width, initial-scale=1.0">

This is normal content. (2)
```

(1) Place the attribute list directly on the paragraph

(2) Once an empty line is encountered the syntax highlighting is unset.

Result: Source paragraph

```xml
<meta name="viewport"
  content="width=device-width, initial-scale=1.0">
```

This is normal content.

If the majority of your source blocks use the same source language, you can set the `source-language` attribute in the document header and assign a language to it.

Source language attribute

```
:source-highlighter: pygments
:source-language: java

[source]
public void setAttributes(Attributes attributes) {
                this.options.put(ATTRIBUTES, attributes.map());
        }

Don't worry, if you need to use another source language, you can assign the language directly on the block.

[source,ruby]
require 'sinatra'
```

Additionally, you can use an include directive to insert source code into an AsciiDoc document directly from a file.

Code block inserted from another file

```
[source,ruby]
----
include::app.rb[]
----
```

Syntax highlighting is not disabled if an explicit `subs` attribute is used on a source listing as long as `specialcharacters` is in the subs list.

47.1. Pygments

Pygments (http://pygments.org) is a popular syntax highlighter that supports a broad range of programming, template and markup languages (http://pygments.org/languages/).

In order to use Pygments with Asciidoctor, you need Python 2 (https://www.python.org) and the pygments.rb (https://rubygems.org/gems/pygments.rb) gem (which bundles Pygments (http://pygments.org)).

You *do not* need to install Pygments itself. It comes bundled with the pygments.rb gem.

You must have Python 2 install. Python 3 is not sufficient to use Pygments with the pygments.rb gem. Check that you have a `python2` (Linux, OSX) or `py -2` (Windows) binary on your PATH.

Installing Python and the pygments.rb gem via the CLI (cross platform)

```
$ "`\which apt-get || \which yum || \which brew`" install python (1)
$ gem install pygments.rb (2)
```

CONSOLE

(1) Install Python using your package manager

(2) Install the pygments.rb gem

Once you've installed these libraries, assign `pygments` to the `source-highlighter` attribute in your document's header.

```
:source-highlighter: pygments
```

ASCIIDOC

You can further customize the source block output with additional Pygments attributes.

pygments-style	Sets the name of the color theme Pygments uses. Default: `pastie`.
pygments-css	Controls what method is used for applying CSS to the tokens. Can be `class` or `style`. Default: `class`.
pygments-linenums-mode	Controls how line numbers are laid out. Can be `table` or `inline`. Default: `table`.

Customizing a source block with Pygments attributes

```
:source-highlighter: pygments
:pygments-style: manni
:pygments-linenums-mode: inline

[source,ruby,linenums]
----
ORDERED_LIST_KEYWORDS = {
  'loweralpha' => 'a',
  'lowerroman' => 'i',
  'upperalpha' => 'A',
  'upperroman' => 'I'
   #'lowergreek' => 'a'
   #'arabic'     => '1'
   #'decimal'    => '1'
}
----
```

ASCIIDOC

Result: Source block using inline line numbers and the manni theme

The Manni Theme

```
1 ORDERED_LIST_KEYWORDS = {
2   'loweralpha' => 'a',
3   'lowerroman' => 'i',
4   'upperalpha' => 'A',
5   'upperroman' => 'I'
6    #'lowergreek' => 'a'
7    #'arabic'     => '1'
8    #'decimal'    => '1'
9 }
```

To list the available Pygments styles, run the following command in a terminal:

```
$ $(dirname $(gem which pygments.rb))/../vendor/pygments-main/pygmentize -L styles
```

The pygments.rb gem uses the version of Pygments that it bundles. This command ensures that you are invoking the `pygmentize` command used by that gem.

If you already have Pygments installed on your system, and you're invoking Asciidoctor via the API, you can instruct Asciidoctor to use your own installation of Pygments by adding the following line before invoking Asciidoctor the first time:

```ruby
Pygments.start '/path/to/pygments'
```

See the Pygments stylesheet section to learn about the `pygments-css` attribute.

47.2. CodeRay

CodeRay (http://coderay.rubychan.de/) is an encoding-aware, syntax highlighter that supports the languages listed below.

C	C++	Clojure
CSS	Delphi	diff
ERB	Go	Groovy
HAML	HTML	Java
JavaScript	JSON	Lua
PHP	Python	Ruby
Sass	SQL	Taskpaper
XML	YAML	

In order to use CodeRay with Asciidoctor, you need the coderay RubyGem (https://rubygems.org/gems/coderay).

Installing the CodeRay RubyGem via the CLI

```
$ gem install coderay
```

Once you've installed the RubyGem, assign `coderay` to the `source-highlighter` attribute in your document's header.

```
:source-highlighter: coderay
```

You can further customize the source block output with additional CodeRay attributes.

coderay-css	Controls what method is used for applying CSS to the tokens. Can be `class` or `style`. Default: `class`.
coderay-linenums-mode	Controls how line numbers are laid out. Can be `table` or `inline`. Default: `table`.

Customizing a source block with CodeRay line numbers

```
:source-highlighter: coderay
:coderay-linenums-mode: inline

[source,ruby,linenums]
----
ORDERED_LIST_KEYWORDS = {
  'loweralpha' => 'a',
  'lowerroman' => 'i',
  'upperalpha' => 'A',
  'upperroman' => 'I'
   #'lowergreek' => 'a'
   #'arabic'     => '1'
   #'decimal'    => '1'
}
----
```
ASCIIDOC

See the CodeRay stylesheet section to learn about the `coderay-css` attribute.

48. Callouts

Pending

48.1. Copy and paste friendly callouts

In versions prior to Asciidoctor 0.1.4, when a reader visiting an HTML page generated by Asciidoctor selected source code from a listing that contained callouts and copied it, the callout numbers would get caught up in the copy. If the reader pasted that code and tried to run it, likely the extra characters from the callouts caused compile or runtime errors.

Asciidoctor uses CSS to prevent callouts from being selected.

On the other side of the coin, you don't want the callout annotations or CSS messing up your raw source code either. You can tuck your callouts neatly inside line comments. Asciidoctor will recognize the line comments characters in front of a callout number, optionally offset by a space, and remove them when rendering the document.

Here are the line comments that are supported:

```
ASCIIDOC
----
line of code  // <1>
line of code  # <2>
line of code  ;; <3>
----
<1> A callout behind a line comment for C-style languages.
<2> A callout behind a line comment for Ruby, Python, Perl, etc.
<3> A callout behind a line comment for Clojure.
```

Here's how it looks when rendered:

```
line of code  (1)
line of code  (2)
line of code  (3)
```

(1) A callout behind a line comment for C-style languages.

(2) A callout behind a line comment for Ruby, Python, Perl, etc.

(3) A callout behind a line comment for Clojure.

Callouts must be placed at the end of the line.

48.1.1. XML callouts

XML doesn't have line comments, so our "tuck the callout behind a line comment" trick doesn't work here. To use callouts in XML, you must place the callout's angled brackets around the the XML comment and callout number.

Here's how it appears in a listing:

```
ASCIIDOC
[source,xml]
----
<section>
  <title>Section Title</title> <!--1-->
</section>
----
<1> The section title is required.
```

Here's how it looks when rendered:

```
XML
<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. Now both you and the reader can copy and paste XML source code containing callouts without worrying about errors.

48.2. Callout icons

The font icons setting also enables callout icons drawn using CSS.

```
= Document Title
:icons: font (1)

NOTE: Asciidoctor now supports font-based admonition icons, powered by Font Awesome! (2)
```

1 Activates font-based icons in the HTML5 backend

2 Admonition block that uses a font-based icon

49. Include Directive

The include directive allows you to insert content from a file into a parent file, effectively merging the content.

Include directives are handled by the preprocessor, so they are oblivious to the structure of the document. They are primarily just a file merger, with one exception. The include directives can resolve document-level attributes.

Only content in AsciiDoc files are preprocessed when included. Asciidoctor recognizes files with the extensions `.asciidoc`, `.adoc`, `.ad`, `.asc`, or `.txt` as AsciiDoc files. The content in all other files are included as is.

The include directive is *disabled* when Asciidoctor is run in secure mode, which is the default safe mode when using Asciidoctor via the API. In secure mode, include directives become *links* in the output document. To learn more about secure mode, refer to the section Running Asciidoctor Securely.

Files included in the master (top-level) document are resolved relative to the base directory, which defaults to the directory of the master document unless otherwise specified. Files included inside a file which itself has been included are resolved relative to the current document.

Document parts

```
= Reference Documentation
Lead Developer

This is documentation for project X.

include::basics.adoc[]

include::installation.adoc[]

include::example.adoc[]
```

ASCIIDOC

Includes and whitespace

Asciidoctor does not insert blank lines between adjacent include statements. We recommend always inserting a blank line around each include directive to avoid unexpected results (e.g., a section title getting appended to the end of a previous paragraph). Also note that the indentation of the included content *is not* altered unless you set the indent attribute.

The `tag` and `line` attributes on the include directive can be used to retrieve specific chunks of content.

Asciidoctor supports targets that include spaces and the `{sp}` attribute references.

If you need to prevent the include directive from being processed, perhaps to show an include directive example in your document, escape it by prefixing the line with a backslash.

Escaping an include directive

```
\include::fragment.adoc[]
```

ASCIIDOC

The include directive will be shown without the preceding backslash in the output document.

Unprocessed include directive

```
include::fragment.adoc[]
```

ASCIIDOC

49.1. Selecting parts of a document to include

The include directive supports extracting portions of content from within a document. The content is specified either by user defined tags or a range of line numbers.

When including multiple line ranges or multiple tags, the individual values can be separated either by a comma or a semi-colon. If commas are used, then the entire attribute value must be enclosed in quotes. Using the semi-colon as the data separator alleviates this requirement.

49.1.1. By tagged regions

Tags are useful when you want to display specific regions of content from an include file instead of all of its content. You can select tagged regions of content with the `tag` macro and the `tags` attribute. The example below shows how you tag a region of content inside a file containing multiple code examples.

Tagged code snippets in a file named core.rb

```ruby
# tag::timings[] (1) (2)
if timings
  timings.record :read
  timings.start :parse
end
# end::timings[] (3) (4)
# tag::parse[] (5)
doc = (options[:parse] == false ? (Document.new lines, options) :
    (Document.new lines,options).parse)
timings.record :parse if timings
doc
# end::parse[] (6)
```

1. To indicate the start of a tagged region, insert a comment line in the code.
2. Assign a unique name to the tag macro. In this example the tag is called *timings*.
3. Insert another comment line where you want the tagged region to end.
4. Assign the name of the region you want to terminate to the `end` macro.
5. This is the start a tagged snippet named *parse*.
6. This is the end of the tagged snippet named *parse*.

The `tag::[]` and `end::[]` directives should be placed after a line comment as defined by the language of the source file. The directives must also appear at the *end* of the line. In the previous example, we choose to prefix the lines with a hash (`#`) because that's the start of a line comment in Ruby.

In the next example, the tagged region named *parse* is called by the `include` directive.

Calling the parse code snippet from a document

```asciidoc
[source,ruby]
----
include::core.rb[tags=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.

Calling the timings and the parse code snippets from a document

```asciidoc
[source,ruby]
----
include::core.rb[tags=timings;parse]
----
```

It is also possible to have fine-grained tagged regions inside larger tagged regions.

For example, if your include file has the following content:

```asciidoc
// tag::snippets[]
// tag::snippet-a[]
snippet a
// end::snippet-a[]

// tag::snippet-b[]
snippet b
// end::snippet-b[]
// end::snippets[]
```

And you include this file using the following include directive:

```
include::file-with-snippets.adoc[tag=snippets]
```
ASCIIDOC

The following lines will be selected and displayed:

```
snippet a

snippet b
```
ASCIIDOC

Notice that none of the lines with the tag directives are displayed.

for XML files you can use the `<!-- tag::name[] -->` and `<!-- end::name[] -->` delimiters

Alternately, you can select content by line number.

49.1.2. By line ranges

To include content by a line or line range, assign a numerical value to the `lines` attribute.

```
include::filename.txt[lines=5..10]
```
ASCIIDOC

Multiple ranges can be specified as well. Using a comma as separator.

```
include::filename.txt[lines="1..10,15..20"]
```
ASCIIDOC

Other example using semi-colons (no need to quote the `lines` attribute value).

```
include::filename.txt[lines=7;14..25;28..43]
```
ASCIIDOC

You can refer to the last line of the document in the range using the value -1.

```
include::filename.txt[lines=12..-1]
```
ASCIIDOC

49.2. Relative leveloffset

The `leveloffset` attribute is used to shift the level of sections (http://asciidoc.org/userguide.html#X90) when combining documents. It works great for a single include level, but it quickly breaks down when you get into multiple levels of nesting.

The problem is that the level offset value is assumed to be absolute. Asciidoctor now supports *relative* level offset values using a leading + or - operator.

```
:leveloffset: +1
include::chapter-01.adoc[]
:leveloffset: -1
```

Alternatively, you can specify the `leveloffset` attribute directly on the include directive so you don't have to worry about restoring the old value.

```
include::chapter-01.adoc[leveloffset=+1]
```

49.3. Normalizing block indentation

Source code snippets from external files are often padded with a leading block indent. This leading block indent is relevant in its original context. However, once inside the documentation, this leading block indent is no longer needed.

The attribute `indent` allows the leading block indent to be stripped and, optionally, a new block indent to be set for blocks with verbatim content (listing, literal, source, verse, etc.).

- When `indent` is 0, the leading block indent is stripped (tabs are also replaced with 4 spaces).
- When `indent` is > 0, the leading block indent is first stripped (tabs are also replaced with 4 spaces), then a block is indented by the number of columns equal to this value.

For example, this AsciiDoc source:

```
[source,ruby,indent=0]
----
    def names
      @name.split ' '
    end
----
```

ASCIIDOC

Produces:

```
def names
  @name.split ' '
end
```

This AsciiDoc source:

```
[indent=2]
----
    def names
      @name.split ' '
    end
----
```

Produces:

```
  def names
    @name.split ' '
  end
```

49.4. Include files relative to a common source directory

Asciidoctor expands attributes in the target of the include directive. That means you only have to type the unique part of the path when linking to a source file.

Example:

```
:sourcedir: src/main/java

[source,java]
----
include::{sourcedir}/org/asciidoctor/Asciidoctor.java[]
----
```

ASCIIDOC

The target of the include resolves to:

```
src/main/java/org/asciidoctor/Asciidoctor.java
```

49.5. Include content from a URI

The include directive can insert content directly from a URI.

Here's an example that demonstrates how to include AsciiDoc content:

```
:asciidoctor-source: https://raw.githubusercontent.com/asciidoctor/asciidoctor/master

include::{asciidoctor-source}/README.adoc[]
```

ASCIIDOC

Since this is a potentially dangerous feature, 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 Asciidoctor to read content from a URI.

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:

- install the open-uri-cached gem
- set the `cache-uri` attribute

When these two conditions are satisfied, Asciidoctor will cache content read from a URI according the to HTTP caching recommendations (http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html).

50. Docinfo file

You can add custom content to the header or footer of a document with document information (`docinfo`) files. The docinfo file's content and file extension depends on whether the source document's output is HTML or DocBook.

50.1. Create a docinfo file

When the source document will be rendered to HTML, the docinfo file can contain the elements allowed in an HTML `<HEAD>` section, including:

- `<base>`
- `<link>`
- `<meta>`
- `<noscript>`
- `<script>`
- `<style>`
- `<title>`

As seen in the example below, docinfo files are useful for storing metadata, stylesheet, and script information. Note that you do not need to include the `<head>` tag, nor is the `<title>` tag required.

A docinfo file for HTML output

```HTML
<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.

When DocBook 5.0 is the rendered output, the docinfo file may include any of the <info> element's 48 children (http://www.docbook.org/tdg5/en/html/info.html), such as:

- `<address>`
- `<copyright>`
- `<edition>`
- `<keywordset>`
- `<publisher>`
- `<subtitle>`

The following example shows some of the content a docinfo file for DocBook might contain.

A docinfo file for DocBook 5.0 output

```xml
<author>
  <personname>
    <firstname>Karma</firstname>
    <surname>Chameleon</surname>
  </personname>
  <affiliation>
    <jobtitle>Word SWAT Team Leader</jobtitle>
  </affiliation>
</author>

<keywordset>
  <keyword>open source</keyword>
  <keyword>documentation</keyword>
  <keyword>adventure</keyword>
</keywordset>

<printhistory>
  <para>April, 2019. Twenty-sixth printing.
  </para>
</printhistory>
```

Docinfo files for DocBook output must be saved with the `.xml` file extension. To see another example of a docinfo file for DocBook, checkout the RichFaces Developer Guide docinfo file (https://github.com/richfaces/richfaces-docs/blob/master/Developer_Guide/src/main/docbook/en-US/Developer_Guide-docinfo.xml).

50.2. Docinfo attributes and file names

To add content from a docinfo file to the header of a rendered document:

1. The docinfo file must be named correctly
2. The appropriate docinfo attribute must be set in the source document

How you name your docinfo file and the docinfo attribute you set in your source document depends on whether you want to add content to:

a. a specific document

b. all the documents in a directory

c. a and b

The attributes and the corresponding file names are listed in the table below.

Docinfo attributes and file names

Attribute	Purpose	docinfo file name for HTML	docinfo file name for DocBook
docinfo	Add content to a specific document	`<docname>-docinfo.html`	`<docname>-docinfo.xml`
docinfo1	Add content to all documents in the same directory	`docinfo.html`	`docinfo.xml`
docinfo2	Add content from either docinfo file	`<docname>.html`, `<docname>-docinfo.html`	`<docname>.xml`, `<docname>-docinfo.xml`

Let's apply the information above to an example exercise.

You have two AsciiDoc documents, adventure.adoc and insurance.adoc, saved in the same folder. You want to add the same content to the headers of both documents when they're rendered to HTML.

1. Create a docinfo file using <head> elements.
2. Save the docinfo file as docinfo.html.
3. Set the `docinfo1` attribute in adventure.adoc and insurance.adoc.

You also want to include some additional content to the header of adventure.adoc.

1. Create **another** docinfo file using <head> elements.
2. Save the docinfo file as adventure-docinfo.html.
3. Set the `docinfo` attribute in adventure.adoc.
4. Alternatively, you could remove the docinfo1 attribute and set docinfo2 instead.

50.3. Footer docinfo files

Footer docinfo files are differentiated from header docinfo files by adding `-footer+` to the file name. In the HTML output, the footer content is inserted inside the footer div (i.e., `<div id="footer">`). In the DocBook output, the footer content is inserted immediately before the ending `</article>` or `</book>` element.

docinfo

If you want to add content to the footer of a specific document, put the content in the file `<docname>-docinfo-footer.html` (for HTML output) or `<docname>-docinfo-footer.xml` (for DocBook output), where `<docname>` is the name of the document without the AsciiDoc extension. Then, set the attribute docinfo in the source document.

docinfo1

If you want to add content to the footer of all documents in the same directory, put the content in the file docinfo-footer.html or docinfo-footer.xml. Then, set the attribute docinfo1 in the source document to enable the feature.

docinfo2

If you want the document to look for either docinfo file, set the attribute docinfo2 in the source document.

The "Last updated" line in the footer can be disabled by unassigning the attribute `last-update-label`.

50.4. Attribute substitution

Docinfo files can include attribute references.

For example, if you created the following docinfo file:

Docinfo file containing a revnumber attribute reference

```xml
<edition>{revnumber}</edition>
```

And this source document:

Source document including a revision number

```
= Document Title
Author Name
v1.0, 2013-06-01
:doctype: book
:backend: docbook
:docinfo:
```

Then the rendered DocBook output would be:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<book xmlns="http://docbook.org/ns/docbook"
    xmlns:xlink="http://www.w3.org/1999/xlink" version="5.0" lang="en">
  <info>
    <title>Document Title</title>
    <date>2013-06-01</date>
    <author>
      <personname>
        <firstname>Author</firstname>
        <surname>Name</surname>
      </personname>
    </author>
    <authorinitials>AN</authorinitials>
    <revhistory>
      <revision>
        <revnumber>1.0</revnumber> (1)
        <date>2013-06-01</date>
        <authorinitials>AN</authorinitials>
      </revision>
    </revhistory>
  </info>
</book>
```

(1) The revnumber attribute reference was replaced by the source document's revision number in the rendered output.

51. Counter Attributes

```
[caption=""]
.Parts{counter2:index:0}
|===
|Part Id |Description

|PX-{counter:index}
|Description of PX-{index}

|PX-{counter:index}
|Description of PX-{index}
|===
```

Parts

Part Id	Description
PX-1	Description of PX-1
PX-2	Description of PX-2

Structuring, Navigating, and Referencing Your Content

Asciidoctor provides macros or attributes for the specialized frontmatter and backmatter sections commonly found in journal articles, academic papers, and books. In the following sections, you'll learn the dependency, structure, and output rules for the special section macros and attributes and how to use them.

52. Title Page

Section pending

53. Colophon

Book colophons list information such as the ISBN, publishing house, edition and copyright dates, legal notices and disclaimers, cover art, design, and book layout credits, and any significant production notes. In most mass market books, the colophon is on the verso (back side) of the title page, but it can also be located at the end of the book.

```
[colophon]
= Colophon

The Asciidoctor Press, Ceres and Denver

(C) 2013 by The Asciidoctor Press

Published in the Milky Way Galaxy

This book is designed by Dagger Flush, Denver, Colorado.
The types are handset Chinchilla and Dust, designed by Leeloo.
Leeloo designed the typefaces to soften the bluntness of documentation.
Created in Asciidoctor and Fedora 19.
The printing and binding is by Ceres Lithographing, Inc., Ceres, Milky Way.
```

54. Table of Contents

A table of contents (TOC) is a list of section and nested subsection titles that is generated when Asciidoctor processes a document. The TOC is inserted directly below the document's title and author, titled *Table of Contents*, and lists the section 1 and section 2 level titles by default.

When the `docbook` converter is used, a TOC is inserted by default. To include a TOC when using the `html` or a custom backend, you must set the `toc` attribute. The toc attribute can be specified via the command line (`-a toc`) or in the document's header (`:toc:`)

TOC enabled via the CLI

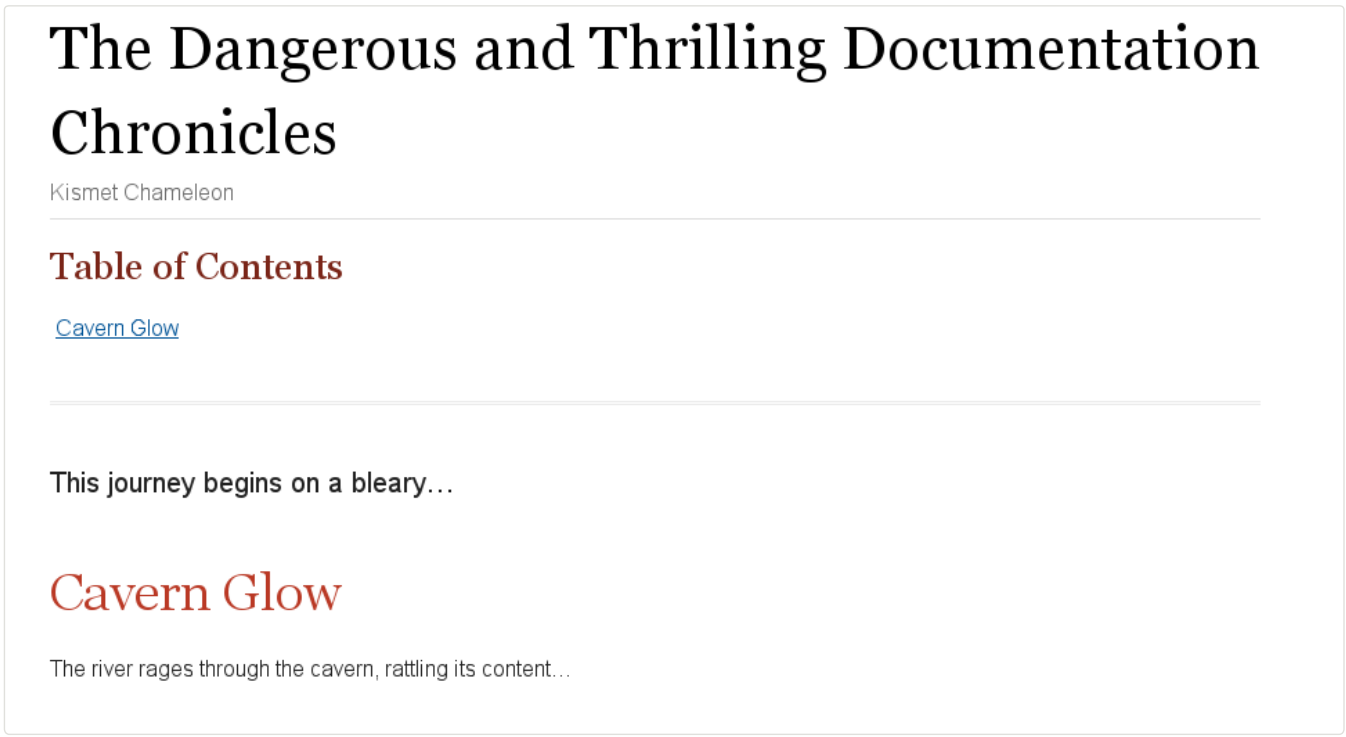
```
$ asciidoctor -a toc adventure.adoc
```

TOC enabled via the document's header

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

This journey begins on a bleary...

== Cavern Glow

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

When the `toc` attribute is set, a TOC is inserted directly below the title and author in the rendered document.

Result: Setting the toc attribute



Asciidoctor provides additional attributes to customize the placement, layout and title of the table of contents.

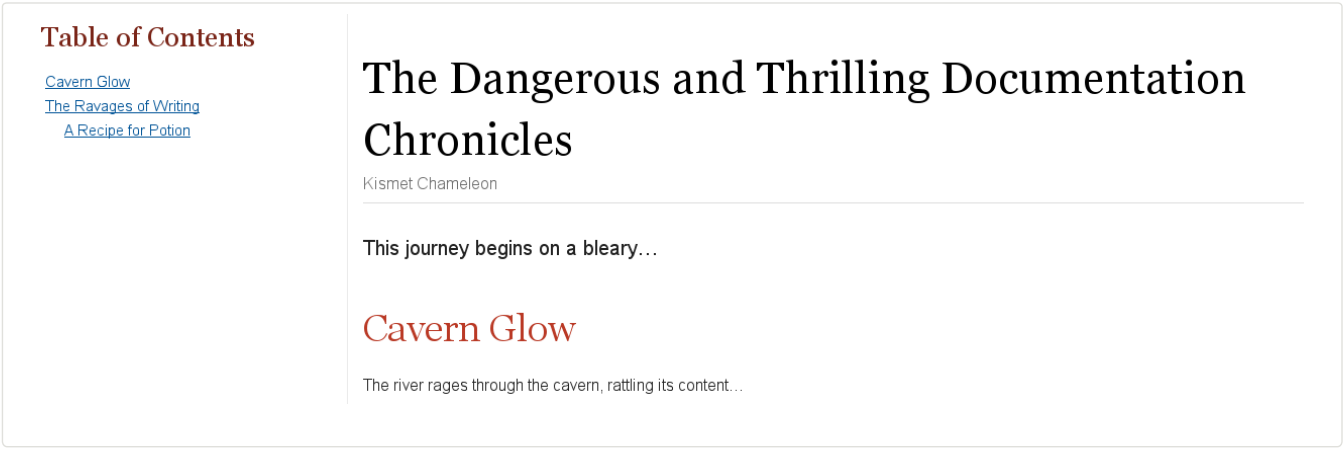
54.1. Left or right column layout

The TOC can be positioned in a column to the left or right of the document's content by assigning the `left` or `right` value to the toc attribute. The column is scrollable and fixed.

Assigning the left value to toc

```
= The Dangerous and Thrilling Documentation Chronicles
Kismet Chameleon
:toc: left

This journey begins on a bleary...

== Cavern Glow

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

== The Ravages of Writing

Her finger socks had been vaporized by crystalline nuggets of...

=== A Recipe for Potion

Two fresh Burdockian leaves, harvested by the light of the teal moons...

==== Searching for Burdockian

Crawling through the twisted understory...
```

Result: Assigning the left value to toc



54.2. Manual placement

The `macro` value on the toc attribute allows you to place the TOC anywhere in your document. To control where the TOC is rendered in your document, set the toc attribute to `macro`. Then, within the body of your document, use the `toc::[]` block macro to indicate the TOC's location.

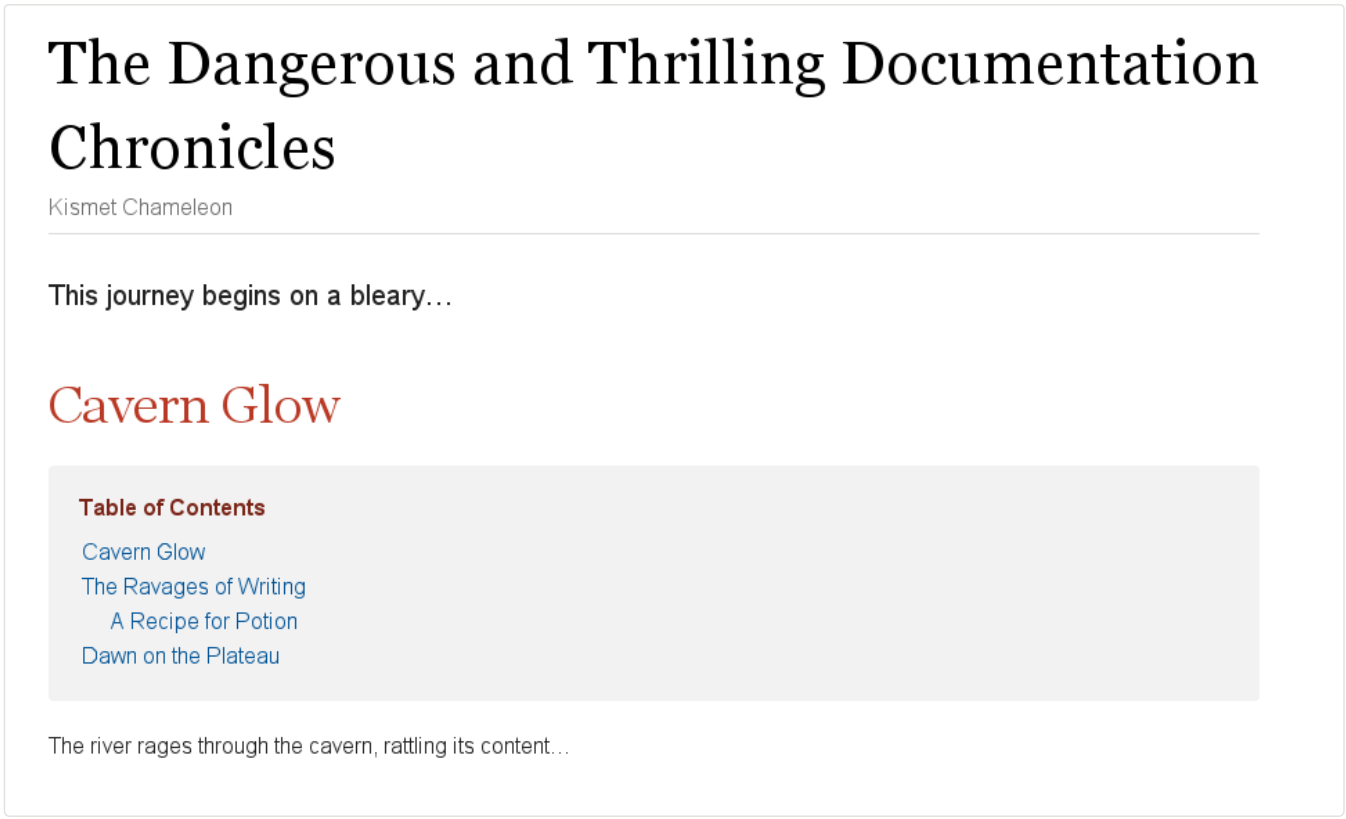
Setting toc value to macro and using the toc::[] macro

```
= The Dangerous and Thrilling Documentation Chronicles
Kismet Chameleon
:toc: macro (1)

This journey begins on a bleary...

== Cavern Glow

toc::[] (2)

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

(1) The toc attribute must be set to macro in order to enable the use of the toc::[] macro.

(2) In this example, the toc::[] macro is placed below the first section's title, indicating that this is the location where the TOC will be displayed once the document is rendered.

Result: Setting toc value to macro and using the toc::[] macro



Asciidoctor also has a `toc-placement` attribute, that when set with a value of `preamble` places the TOC below the preamble. You do not need to include the toc::[] block macro in the document when using the preamble value.

If you set toc-placement but don't assign it a value, the TOC will be placed in the document's header by default.

The `toc-placement` and `toc-position` attributes are mutually exclusive.

54.3. Title

The `toc-title` attribute allows you to change the title of the TOC.

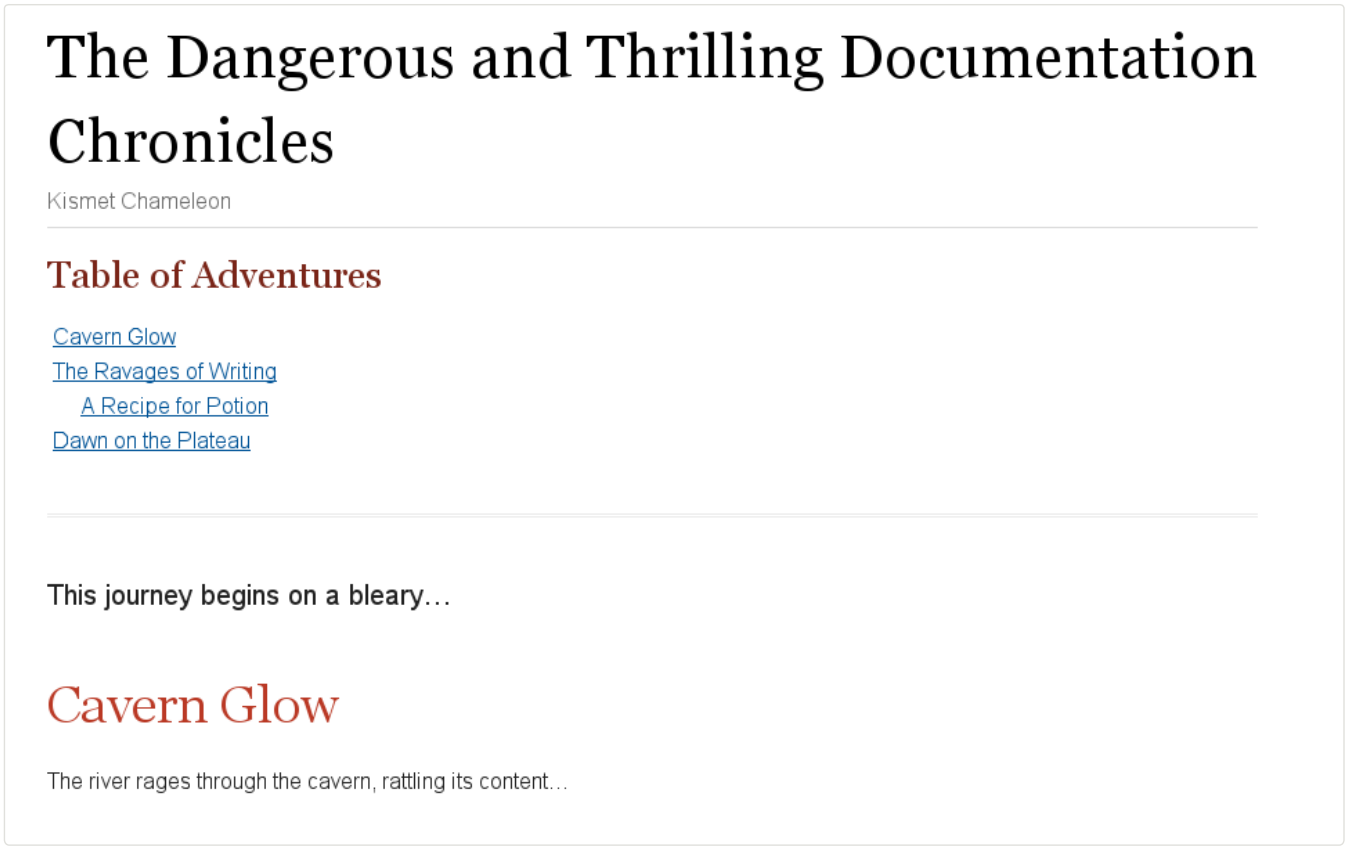
Defining a new TOC title

```
= The Dangerous and Thrilling Documentation Chronicles
Kismet Chameleon
:toc: (1)
:toc-title: Table of Adventures (2)

This journey begins on a bleary...

== Cavern Glow

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

(1) The toc attribute must be set in order to use toc-title.

(2) toc-title is set and assigned the value `Table of Adventures` in the document's header.

Result: Defining a new TOC title



54.4. Levels

By default, the TOC will display level 1 and level 2 section titles. You can set a different level with the `toclevels` attribute.

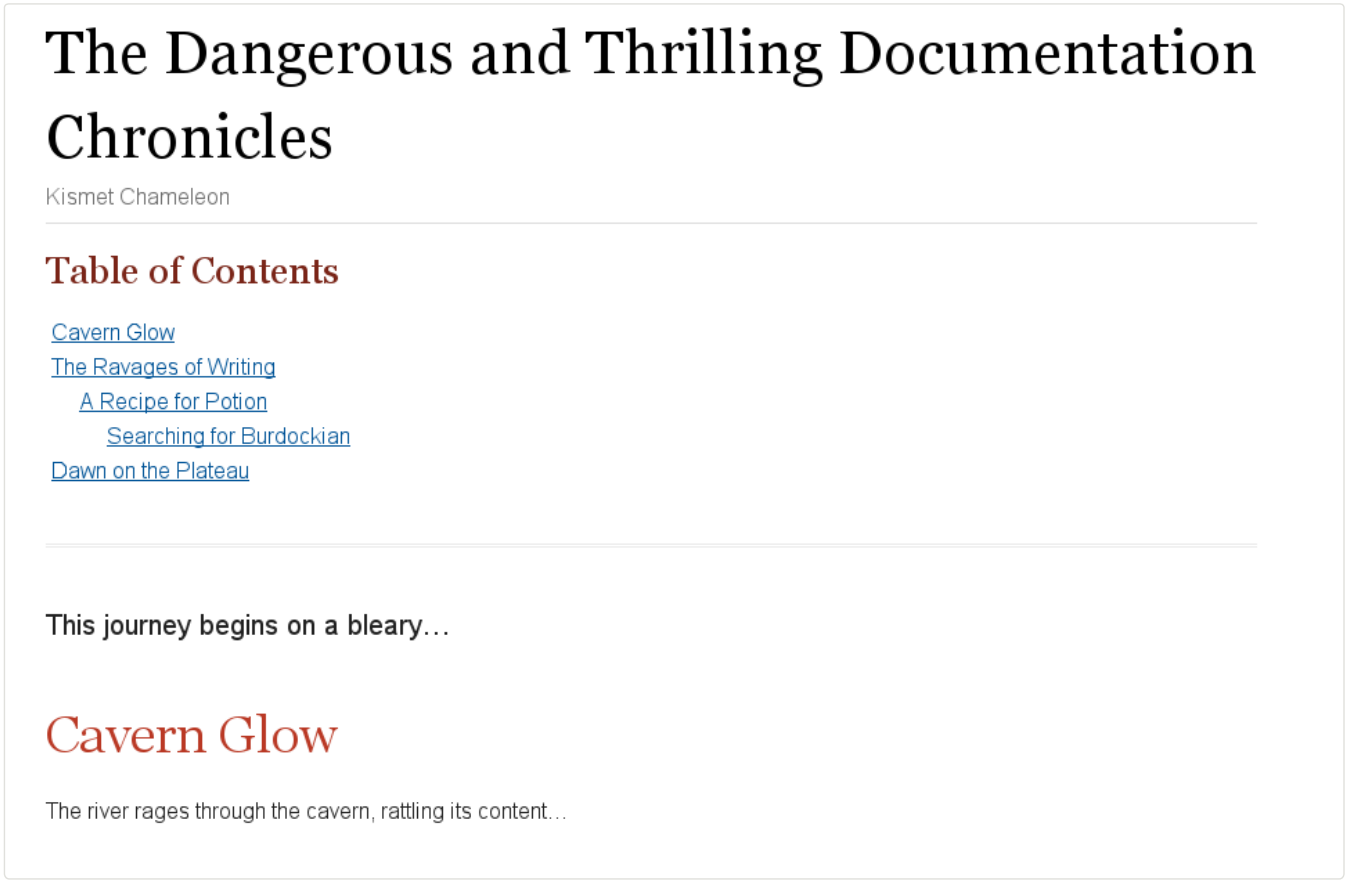
Defining a new toclevels value

```
= The Dangerous and Thrilling Documentation Chronicles
Kismet Chameleon
:toc: (1)
:toclevels: 4 (2)

This journey begins on a bleary...

== Cavern Glow

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

== The Ravages of Writing

Her finger socks had been vaporized by crystalline nuggets of...

=== A Recipe for Potion

Two fresh Burdockian leaves, harvested by the light of the teal moons...

==== Searching for Burdockian

Crawling through the twisted understory...
```

(1) The toc attribute must be set in order to use toclevels.

(2) toclevels is set and assigned the value 4 in the document's header. The TOC will list the titles of the section 1, 2, 3, and 4 levels when the document is rendered.

Result: Defining a new toclevels value



54.5. Using a TOC with embeddable HTML

When AsciiDoc is converted to embeddable HTML (i.e., the `header_footer` option is `false`), there are only two valid values for the `toc` attribute:

- 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 sidebar TOC (left or right) isn't available in this mode. That's because the embeddable HTML doesn't have the outer framing necessary to support a sidebar TOC.

The macro (aka manual) placement requires the use of the `toc::[]` macro somewhere in the document. The preamble placement only works if your document has a preamble (otherwise, there is no slot for it).

54.6. Table of Contents summary

Table of contents attributes and values

Attribute	Values	Example Syntax	Comments	Backends
`toc`	auto, left, right, macro, preamble	`:toc:`	Defaults to `auto` if value is blank. When `header_footer` is disabled, only `macro` and `preamble` are recognized. Use `macro` in order to use the `toc::[]` block macro in the document body. DocBook backend sets `toc` by default.	html, docbook, pdf

`toclevels`	1–5	`:toclevels: 4`	Default value is `2`.	html, pdf
`toc-title`	<user defined>	`:toc-title: Rocking the Table`	Default title is *Table of Contents*.	html, pdf
`toc-placement`	auto, preamble, macro	`:toc-placement: preamble`	*Deprecated: Use* `toc` *attribute instead.* Default value is `auto` when `toc` is specified.	html
`toc-position`	left, right	`:toc-position: right`	*Deprecated: Use* `toc` *attribute instead.*	html

55. Abstract

An abstract is a concise overview of an article. Typically found in the frontmatter of academic, research, and analytical papers, an abstract can be classified as structured or unstructured and complete or limited. Abstracts with subheadings are structured abstracts, whereas abstracts without subheadings are unstructured. A complete (informative) abstract states the key topics and findings of the paper while a limited (descriptive) abstract briefly describes the structure of the article. An abstract title is not necessary.

```
[abstract]
== Documentation Abstract

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

56. Preface

An author explains how the idea behind their book manifested and developed in the preface. The preface can include acknowledgments and be signed by the author with her name, writing date and location.

The preface can contain subsections.

```
[preface]
== Documentation Preface

The basis for this documentation germinated when I awoke one morning and was confronted by the dark and stormy eyes of the
She had conquered the mountain of government reports that, over the course of six months, had eroded into several minor foo

=== Acknowledgments

I would like to thank the Big Bang and String Theory.
```

If the document is a multi-part book, the preface title must be level 0 and its subsections must start at level 2.

57. Dedication

The dedication page is where the author expresses gratitute toward a person.

```
[dedication]
= Dedication

For S.S.T.--

thank you for the plague of archetypes.
```

58. Book Parts and Part Introductions

You can group the sections of your document into parts. Parts can only be used when the `doctype` is `book`. Part titles are specified with a single equals sign (`=`), the equivalent to a document title, i.e. section level 0. After the part title, you can include an optional introduction, which is designated by the `partintro` attribute.

```
[partintro]
.Optional part introduction title
--
Optional part introduction goes here.
--
```

A part can also include its own preface, bibliography, glossary and index.

```
= Title of Part 1

[partintro]
--
This is the introduction to the first part of our mud-encrusted journey.
--

== Chapter 1

There was mud...

== Chapter 2

Great gobs of mud...

[glossary]
== Part 1 Glossary

[glossary]
mud:: wet, cold dirt

= Part 2

[preface]
== Part 2 Preface

This part was written because...

== Chapter 1

The mud had turned to cement...
```

59. Appendix

You can define that some sections are appendix sections by adding the `[appendix]` marker before the section header.

```
[appendix]
== Copyright and License
```

Sections marked as appendix have different header, built as follow:

- A fixed prefix (“Appendix” is the default value)
- Letters are used to number the section (A, B, C⋯)
- A colon
- The section title

The prefix can be modified with the `appendix-caption` attribute. It can be unset with:

ASCIIDOC

```
: appendix-caption!:
```

Or set to a different value with

ASCIIDOC

```
: appendix-caption: App
```

Appendix sections can be combined with subsections. Example:

ASCIIDOC

```
:appendix-caption: App
:numbered:

== Title

=== Subsection

[appendix]
== First Appendix

=== First subsection

=== Second subsection

[appendix]
== Second Appendix
```

The sections will be numbered as follows:

ASCIIDOC

```
1. Title
1.1. Subsection

App A: First Appendix
A.1. First subsection
A.2. Second subsection

App B: Second Appendix
```

60. Glossary

You can include a glossary of definitions by including the `[glossary]` marker before the section header and before the first definition.

```
[glossary]
== Glossary

[glossary]
mud:: wet, cold dirt
rain::
        water falling from the sky
```

61. Bibliography

References

```
_The Pragmatic Programmer_ <<prag>> should be required reading for
all developers.

[bibliography]
- [[[prag]]] Andy Hunt & Dave Thomas. The Pragmatic Programmer:
  From Journeyman to Master. Addison-Wesley. 1999.
- [[[seam]]] Dan Allen. Seam in Action. Manning Publications.
  2008.
```

The Pragmatic Programmer [prag] should be required reading for all developers.

- [prag] Andy Hunt & Dave Thomas. The Pragmatic Programmer: From Journeyman to Master. Addison-Wesley. 1999.
- [seam] Dan Allen. Seam in Action. Manning Publications. 2008.

62. Index

You specify index terms using one of the following syntaxes

```
=== Sir Lancelot

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.

indexterm:[Lancelot] (3)
```

1. The double parenthesis inline macro adds a primary index term, and includes the term in the generated output.
2. The triple parenthesis inline macro allows for an optional second and third index term.
3. The full macro allow for optional second and third index terms.

Asciidoctor does not currently output the index when using the HTML5 backend (the same is true for AsciiDoc Python). To output an index currently, you have to produce HTML5 (by way of DocBook) using the DocBook toolchain (`a2x` or `fopub`). We're tracking support for native index generation when using the HTML5 backend in issue #450 (https://github.com/asciidoctor/asciidoctor/issues/450).

63. Footnotes

A footnote is created with the footnote macro (`footnote:[]`). If you plan to reference a footnote more than once, use the ID footnote macro (`footnoteref:[]`).

Footnote and footnoteref syntax

```
The hail-and-rainbow protocol can be initiated at five levels: double, tertiary, supernumerary, supermassive, and apocalypt
A bold statement.footnoteref:[disclaimer,Opinions are my own.] (3) (4)

Another outrageous statement.footnoteref:[disclaimer] (5)
```

(1) Insert the footnote macro directly after any punctuation. Note that the footnote macro only uses one colon.

(2) Insert the footnote's content within the square brackets of the macro. The text may span several lines.

(3) The first time you enter a footnote you want to reuse, give it a unique ID in the first position.

(4) Separate the ID from the footnote text with a comma. Do not enter a space between the comma and the footnote text.

(5) The next time you reference the footnote you only need to insert the ID in the square brackets.

When rendered, the footnotes will be numbered consecutively throughout the article.

Rendered footnotes

The hail-and-rainbow protocol can be initiated at five levels: double, tertiary, supernumerary, supermassive, and apocalyptic party.[1] A bold statement.[2]

Another outrageous statement.[2]

1 The double hail-and-rainbow level makes my toes tingle.
2 Opinions are my own.

Processing Your Content

While the AsciiDoc syntax is designed to be readable in raw form, the intended audience for that format are writers and editors. Readers aren't going to appreciate the raw text nearly as much. Aesthetics matter. You'll want to apply nice typography with font sizes that adhere to the golden ratio, colors, icons and images to give it the respect it deserves. That's where the Asciidoctor themes and backends come into play.

64. Selecting an Output Format

The Asciidoctor processor parses an AsciiDoc document and converts it to a variety of formats, including HTML, DocBook and PDF. The processor produces the output format by using a backend that you specify with the `-b` (`--backend`) command line option. If no backend is indicated, the processor uses the default backend (`html5`).

Asciidoctor has four native backends.

`html5`
: HTML 5 markup styled with CSS3. Asciidoctor's default backend.

`xhtml5`
: XHTML 5 markup.

`docbook`
: DocBook XML 5.0 markup.

`docbook45`
: DocBook XML 4.5 markup.

It also has several backends that can be plugged in.

`manpage`
: Manual pages for Unix and Unix-like operating systems.

`pdf`
: Portable Document Format (PDF).

`deckjs`
: Deck.js slideshow.

65. HTML

Asciidoctor's default output format is HTML.

`html5`
: HTML 5 markup styled with CSS3.

65.1. Using the command line

In this section, we'll create a sample document, then process and render it with Asciidoctor's `html5` converter.

1. Create an AsciiDoc file like the one below
2. Save the file as `mysample.adoc`

```
= My First Experience with the Dangers of Documentation

In my world, we don't have to worry about mutant, script-injecting warlocks.
No.
We have something far worse.
We're plagued by Wolpertingers.

== Origins

You may not be familiar with these http://en.wikipedia.org/wiki/Wolpertinger[ravenous beasts], but, trust me, they'll eat y
```

To convert mysample.adoc to HTML from the command line:

1. Open a console
2. Switch to the directory that contains the mysample.adoc document
3. Call the Asciidoctor processor with the `asciidoctor` command, followed by the name of the document you want to render

```
$ asciidoctor mysample.adoc
```

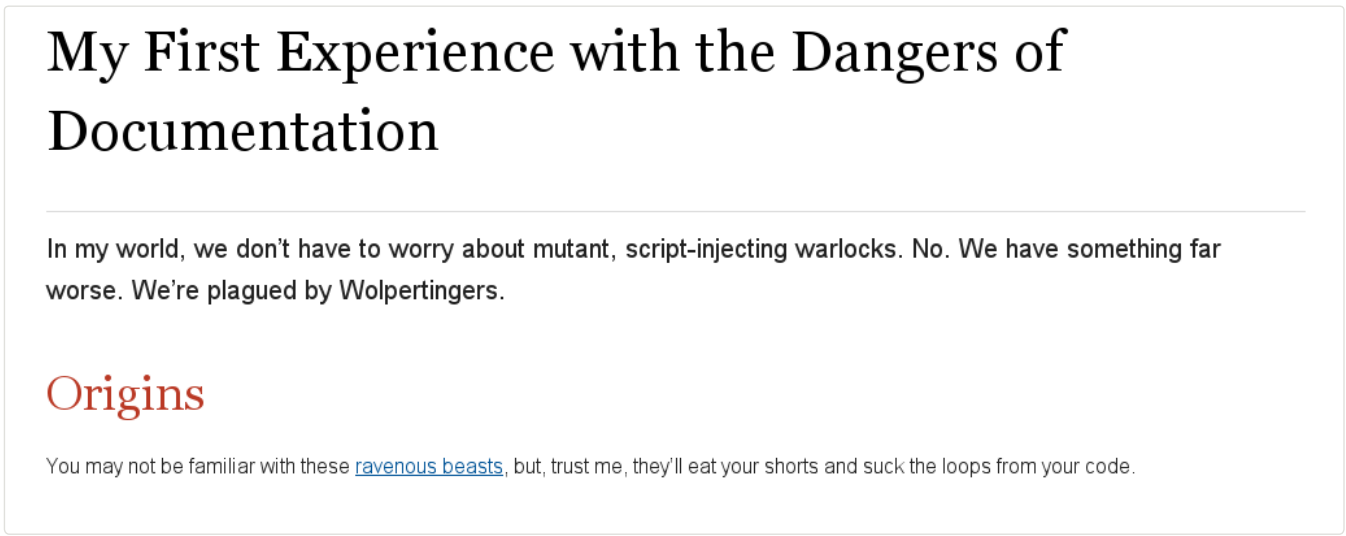
Remember, Asciidoctor's default converter is html5, so it isn't necessary to specify it with the `-b` command.

You won't see any messages printed to the console. If you type `ls` or view the directory in a file manager, there is a new file named `mysample.html`.

```
$ ls
mysample.adoc  mysample.html
```

Asciidoctor derives the name of the output document from the name of the input document.

Open `mysample.html` in your web browser. Your document should look like the image below.



The document's text, titles, and link is styled by the default Asciidoctor stylesheet, which is embedded in the HTML output. As a result, you could save mysample.html to any computer and it will look the same.

65.2. Using the Ruby API

Asciidoctor also includes a Ruby API that lets you generate an HTML document directly from a Ruby application.

```ruby
require 'asciidoctor'

Asciidoctor.convert_file('mysample.adoc', :in_place => true)
```

Alternatively, you can capture the HTML output in a variable instead of writing it to a file.

```ruby
html = Asciidoctor.convert_file('mysample.adoc', :header_footer => true)
puts html
```

65.3. Styling the HTML with CSS

Asciidoctor uses CSS for HTML document styling and JavaScript for generating document attributes such as a table of contents and footnotes. It comes bundled with a fresh, modern stylesheet, named `asciidoctor.css`. When you generate a document with the `html5` backend, the asciidoctor.css stylesheet is embedded into the HTML output by default.

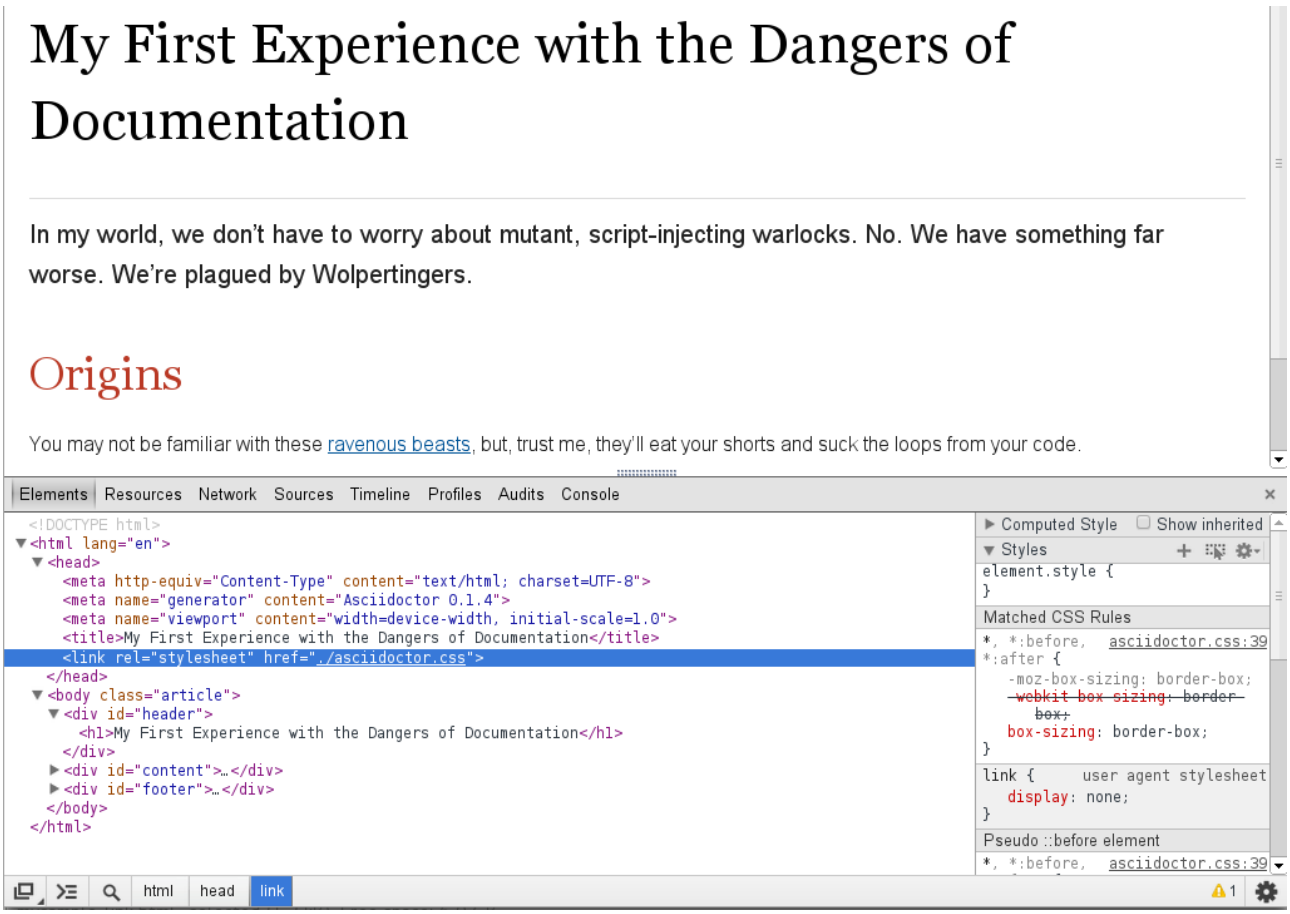
However, you can link to the stylesheet instead of embedding it. To have your document link to the default stylesheet, set the `linkcss` attribute in the document's header.

```
= My First Experience with the Dangers of Documentation
:linkcss:

In my world, we don't have to worry about mutant, script-injecting warlocks.
No.
We have something far worse.
We're plagued by Wolpertingers.

== Origins

You may not be familiar with these http://en.wikipedia.org/wiki/Wolpertinger[ravenous beasts], but, trust me, they'll eat y
```

You can also set linkcss with the CLI.

```
$ asciidoctor -a linkcss mysample.adoc
```

Now, when you view the directory, you should see the file asciidoctor.css in addition to `mysample.adoc` and `mysample.html`. The linkcss attribute automatically copies asciidoctor.css to the output directory. Additionally, you can inspect mysample.html in your browser and see `<link rel="stylesheet" href="./asciidoctor.css">` inside the `<head>` tags.



If you don't want any styles applied to the HTML output of your document, unset the `stylesheet` attribute.

```
$ asciidoctor -a stylesheet! mysample.adoc
```

One of Asciidoctor's strengths is the ease in which you can swap the default stylesheet for an alternative Asciidoctor theme or use your own custom stylesheet.

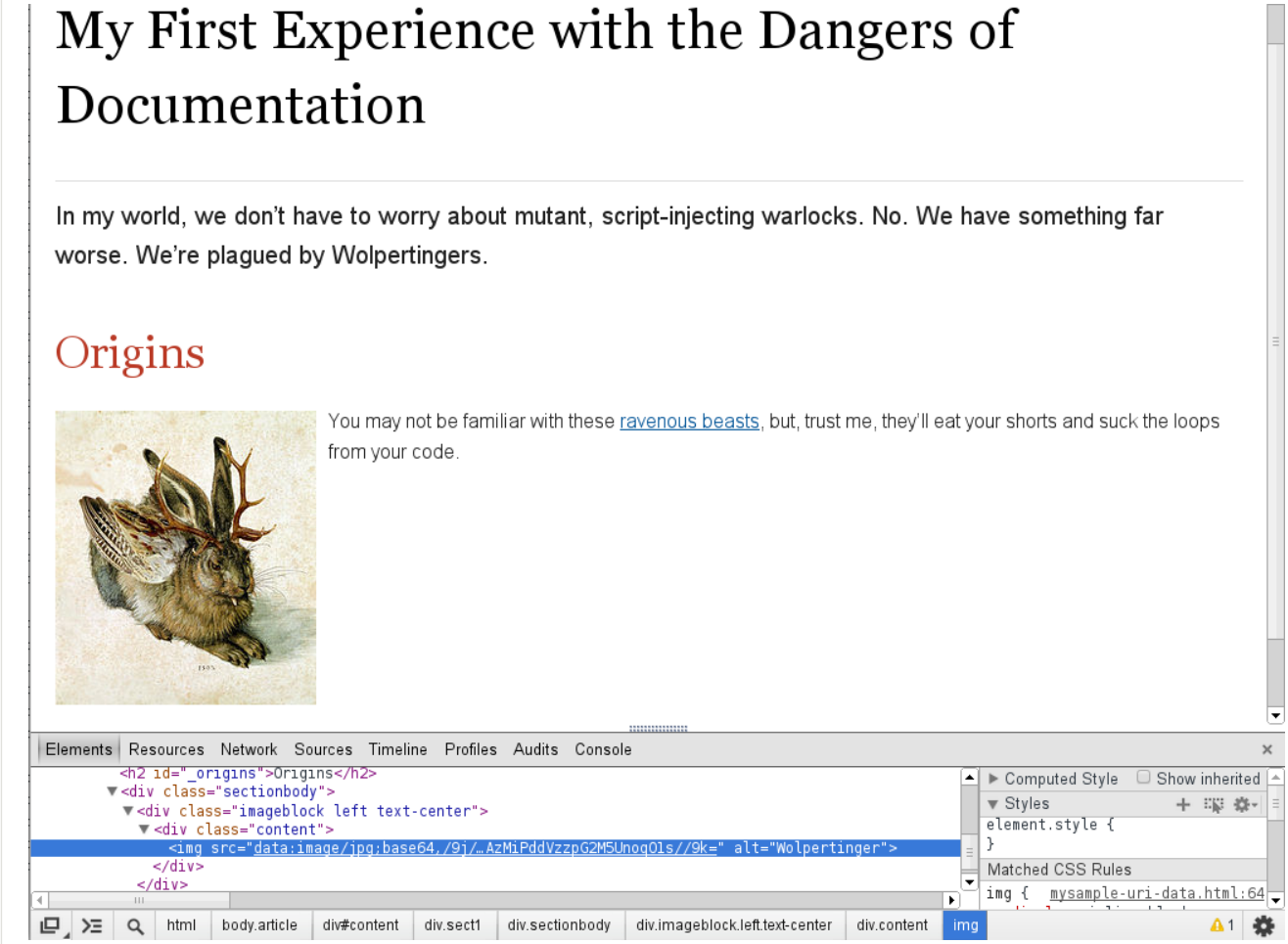
65.4. Managing images

Images are not embedded in the HTML output by default. If you have image references in your document, you'll have to save the image files in the same directory as your rendered document. Or, by passing the `data-uri` attribute to the processor, you can embed the images into the document.

To embed images into the HTML output, set data-uri on the command line or in the document's header.

```
$ asciidoctor -a data-uri mysample.adoc
```

```
= My First Experience with the Dangers of Documentation
:imagesdir: myimages
:data-uri:

In my world, we don't have to worry about mutant, script-injecting warlocks.
No.
We have something far worse.
We're plagued by Wolpertingers.

== Origins

[.left.text-center]
image::wolpertinger.jpg[Wolpertinger]

You may not be familiar with these http://en.wikipedia.org/wiki/Wolpertinger[ravenous beasts], but, trust me, they'll eat y
```



65.5. CodeRay and Pygments stylesheets

Asciidoctor will also embed the theme stylesheet for the CodeRay or Pygments syntax highlighter.

CodeRay

If the `source-highlighter` attribute is `coderay` and the `coderay-css` attribute is `class`, the CodeRay stylesheet is:

- *embedded* by default
- *copied* to the file *asciidoctor-coderay.css* inside the `stylesdir` folder within the output directory if `linkcss` is set

Pygments

If the `source-highlighter` attribute is `pygments` and the `pygments-css` attribute is `class`, the Pygments stylesheet is:

- *embedded* by default
- *copied* to the file *asciidoctor-pygments.css* inside the `stylesdir` folder within the output directory if `linkcss` is set

66. XHTML

To convert AsciiDoc to XHTML, set the backend to `xhtml5`.

Set the backend to xhtml5 in the CLI

```
asciidoctor -b xhtml5 document.adoc
```
CONSOLE

You can also set the backend using the `backend` attribute in the document header and assigning it the `xhtml5` value.

To emit XHTML when using the deck.js templates, set both the backend and the `htmlsyntax` attribute:

Using htmlsyntax with an alternate converter

```
asciidoctor -T /path/to/asciidoctor-backends -b deckjs -a htmlsyntax=html deck.adoc
```
CONSOLE

Black Box Decoded: xhtml and htmlsyntax

The XHTML converter is built on the HTML 5 converter, but when you use the XHTML converter the `htmlsyntax` attribute is implicitly assigned the value `xml`. The HTML 5 converter implicitly assigns `html` to the htmlsyntax attribute

For all other backends, htmlsyntax is not set explicitly. If you want to set the `:format` option to `:html5` when using Slim or Haml, you need to set the htmlsyntax attribute to xml explicitly.

67. DocBook

Asciidoctor can produce DocBook 5.0 and 4.5 output. Since the AsciiDoc syntax was designed with DocBook output in mind, the conversion is very good. There's a corresponding DocBook element for each markup in the AsciiDoc syntax.

To convert the `mysample.adoc` document to DocBook 5.0 format, call the processor with the backend flag set to `docbook`.

```
$ asciidoctor -b docbook mysample.adoc
```

A new XML document, named `mysample.xml`, will now be present in the current directory.

```
$ ls
mysample.adoc  mysample.html  mysample.xml
```

Here's a snippet of the XML generated by the DocBook converter.

XML generated from AsciiDoc

```xml
<?xml version="1.0" encoding="UTF-8"?>
<article xmlns="http://docbook.org/ns/docbook"
    xmlns:xlink="http://www.w3.org/1999/xlink" version="5.0" xml:lang="en">
  <info>
    <title>Hello, AsciiDoc!</title>
    <date>2013-09-03</date>
    <author>
      <personname>
        <firstname>Doc</firstname>
        <surname>Writer</surname>
      </personname>
      <email>doc@example.com</email>
    </author>
    <authorinitials>DW</authorinitials>
  </info>
  <simpara>
    An introduction to <link xlink:href="http://asciidoc.org">AsciiDoc</link>.
  </simpara>
  <section xml:id="_first_section">
    <title>First Section</title>
    <itemizedlist>
      <listitem>
        <simpara>item 1</simpara>
      </listitem>
      <listitem>
        <simpara>item 2</simpara>
      </listitem>
    </itemizedlist>
  </section>
</article>
```

The `xmlns` attribute is now added to the root DocBook element by default. Set the attribute `noxmlns` to disable this feature.

If you're on Linux, you can view the DocBook file with Yelp (https://wiki.gnome.org/action/show/Apps/Yelp).

```
$ yelp mysample.xml
```

And of course, if you're using the Asciidoctor Ruby API, you can generate a DocBook document directly from your application.

Generate DocBook output from the API

```ruby
Asciidoctor.convert_file('mysample.adoc', :in_place => true,
    :backend => 'docbook')
```

By default, the docbook converter produces DocBook 5.0 output that is compliant to the DocBook 5.0 specification.

A summary of the differences are as follows:

- XSD declarations are used on the document root instead of a DTD

- `<info>` elements for document info instead of `<articleinfo>` and `<bookinfo>`
- elements that hold the author's name are wrapped in a `<personname>` element
- the id for an element is defined using an `xml:id` attribute
- `<link>` is used for links instead of `<ulink>`
- the URL for a link is defined using the `xlink:href` attribute

Refer to What's new in DocBook v5.0? (http://www.docbook.org/tdg5/en/html/ch01.html#introduction-whats-new) for more details about how DocBook 5.0 differs from DocBook 4.5.

If you need to output DocBook 4.5, set the backend to `docbook45`.

```
$ asciidoctor -b docbook45 sample.adoc
```

68. Man pages

One of the most interesting uses of AsciiDoc is for creating man pages (short for manual pages) for Unix and Unix-like operating systems. A man page conforms to a special document structure. That structure is recognized by Asciidoctor, when the `doctype` attribute is set to `manpage`. Additionally, Asciidoctor produces the same output when rendering a man page to HTML using the `html5` backend.

When reading an AsciiDoc document using the manpage doctype, Asciidoctor parses the following man page metadata:

- `mantitle`
- `manvolnum`
- `manname`
- `manpurpose`

The mantitle and manvolum are captured from the document title. The manname and manpurpose are taken from the first section of the document, which must be a level 1 section and have content in the format `<manname> - <manpurpose>`.

Here's an example of a man page written in AsciiDoc:

```
= asciidoctor(1)
:doctype: manpage

== NAME
asciidoctor - converts AsciiDoc source files...

== SYNOPSIS
*asciidoctor* ['OPTION']... 'FILE'...
```
ASCIIDOC

From this document, Asciidoctor extracts the following man page-related attributes:

`mantitle`	asciidoctor
`manvolnum`	1
`manname`	asciidoctor
`manpurpose`	converts AsciiDoc source files…

Refer to the AsciiDoc source of the Asciidoctor man page (https://raw.githubusercontent.com/asciidoctor/asciidoctor/master/man/asciidoctor.adoc) to see a complete example. The man pages for git are also produced from AsciiDoc documents, so you can use those as another example to follow.

The Asciidoctor Backends repository hosts an early draft of a manpage backend (https://github.com/asciidoctor/asciidoctor-backends/tree/master/erb/manpage), which is now used to generate the man page for Asciidoctor.

69. PDFs

To convert your document to PDF, you'll need to use the new asciidoctor-fopub tool (https://github.com/asciidoctor/asciidoctor-fopub). Directions for using the tool are documented in the asciidoctor-fopub README (https://github.com/asciidoctor/asciidoctor-fopub/blob/master/README.adoc).

70. Preview Your Content

Section pending

70.1. Guard/Live viewer

Section pending

71. Process multiple source files from the CLI

You can pass multiple source files (or a file name pattern) to the Asciidoctor CLI and it will process all the files in turn.

Let's assume there are two AsciiDoc files in your directory, a.adoc and b.adoc. When you enter the following command in your terminal:

```
$ asciidoctor a.adoc b.adoc
```

Asciidoctor will process both files, transforming a.adoc to a.html and b.adoc to b.html.

To save you some typing, you can use the glob operator (`*`) to match both files using a single argument:

```
$ asciidoctor *.adoc
```

The shell will expand the previous command to the one you typed earlier:

```
$ asciidoctor a.adoc b.adoc
```

You can also render all the AsciiDoc files in immediate subfolders using the double glob operator (`**`) in place of the directory name:

```
$ asciidoctor **/*.adoc
```

To match all files in the current directory and immediate subfolders, use both glob patterns:

```
$ asciidoctor *.adoc **/*.adoc
```

If the file name argument is quoted, the shell will not expand it:

```
$ asciidoctor '*.adoc'
```

This time, the text `*.adoc` gets passed directly to Asciidoctor instead of being expanded to a.adoc and b.adoc. In this case, Asciidoctor handles the glob matching internally in the same way the shell does (when the file name is not in quotes)--with one exception. Asciidoctor can match files in the current directory and subfolders at any depth using a single glob pattern:

```
$ asciidoctor '**/*.adoc'
```

TIP

If you process multiple nested AsciiDoc files at once and are also applying a custom stylesheet to them, you'll need to manage the stylesheet's location.

72. Specifying an output file

The `asciidoctor` command writes to the specified output file if the input is `stdin`.

For example, the following command writes to `output.html` instead of to the console:

```
$ echo "content" | asciidoctor -o output.html
```

73. Running Asciidoctor Securely

Asciidoctor provides security levels that control the read and write access of attributes, the `include` directive, macros, and scripts while a document is processing. Each level includes the restrictions enabled in the prior security level.

`UNSAFE`
: A safe mode level that disables any security features enforced by Asciidoctor. Ruby is still subject to its own restrictions.

  This is the default safe mode for the CLI. Its integer value is `0`.

`SAFE`
: This safe mode level prevents access to files which reside outside of the parent directory of the source file. It disables all macros, except the `include` directive. The paths to `include` files must be within the parent directory. It allows assets to be embedded in the document.

  Its integer value is `1`.

`SERVER`
: A safe mode level that disallows the document from setting attributes that would affect the rendering of the document. This level trims `docfile` to its relative path and prevents the document from:

  - setting `source-highlighter`, `doctype`, `docinfo` and `backend`
  - seeing `docdir`

  It allows `icons` and `linkcss`.

  Its integer value is `10`.

`SECURE`
: A safe mode level that disallows the document from attempting to read files from the file system and including their contents into the document. Additionally, it:

  - disables icons
  - disables the `include` directive
  - data can not be retrieved from URIs
  - prevents access to stylesheets and JavaScripts
  - sets the backend to `html5`
  - disables `docinfo` files
  - disables `data-uri`
  - disables `docdir` and `docfile`
  - disables source highlighting

  Asciidoctor extensions may still embed content into the document depending whether they honor the safe mode setting.

  This is the default safe mode for the API. Its integer value is `20`.

You can set Asciidoctor's safe mode level using the CLI or API.

73.1. Set the safe mode in the CLI

When Asciidoctor is invoked via the CLI, the safe mode is set to `UNSAFE` by default. You can change the safe level by executing one of the following commands in the CLI.

`-S`, `--safe-mode=SAFE_MODE`
: Sets the safe mode level of the document according to the assigned level (`UNSAFE`, `SAFE`, `SERVER`, `SECURE`).

`--safe`, `asciidoctor-safe`

Sets the safe mode level to `SAFE`. Provided for compatibility with the python AsciiDoc `safe` command.

73.2. Set the safe mode in the API

The default safe level in the API is `SECURE`.

In the API, you can set the safe mode using a string, symbol or integer value. The value must be set in the document constructor using the `:safe` option.

```
result = Asciidoctor.convert_file('master.adoc', :safe => 'server')
```

or

```
result = Asciidoctor.convert_file('master.adoc', :safe => :server)
```

or

```
result = Asciidoctor.convert_file('master.adoc', :safe => 10)
```

73.3. Set attributes based on the safe mode

You can enable or disable content within your document based on safe modes using the `safe-mode` attribute.

The safe mode can be defined by its integer value or name. For example, to assign the `SAFE` level to the attribute, you could append `10` or `safe` to the attribute.

```
safe-mode-10
```

or

```
safe-mode-safe
```

The attributes in the next example enable the author to define replacement text for features that are disabled in high security environments.

For example:

```
ifdef::safe-mode-secure[]
Links to chapters.
endif::safe-mode-secure[]
```
ASCIIDOC

Customizing Your Output

Asciidoctor provides a default stylesheet and built-in converters so you can quickly process and render your document, but it also lets you use custom stylesheets and converters. The Asciidoctor project includes alternative stylesheet themes from the stylesheet factory and specialized backends, such as the Deck.js backend. You can also create your own themes and backends.

74. Custom Themes

Section introduction pending

74.1. Creating a theme

You can create your own themes to apply to your documents.

Themes go in the *sass/* folder. To create a new theme, let's call it `hipster`, start by creating two new files:

sass/hipster.scss

- Imports the theme settings, which includes default variables and resets
- Imports the AsciiDoc components
- Defines any explicit customization

sass/settings/_hipster.scss

- Sets variables that customize Foundation 4 and the AsciiDoc CSS components

Here's a minimal version of *sass/hipster.scss*:

```scss
@import "settings/hipster";
@import "components/asciidoc";
@import "components/awesome-icons";
```

You don't have to include the underscore prefix when importing files.

The `awesome-icons` component is only applicable to HTML generated by Asciidoctor > 0.1.2 with the `icons` attribute set to `font`.

You can add any explicit customization below the import lines.

The variables you can set in *sass/settings/_hipster.scss* are a combination of the Foundation 4 built-in global settings (http://github.com/asciidoctor/asciidoctor-stylesheet-factory/blob/master/sass/settings/_settings.scss.dist) and global settings and imports for the AsciiDoc components (http://github.com/asciidoctor/asciidoctor-stylesheet-factory/blob/master/sass/settings/_defaults.scss).

Once you've created your custom theme, it's time to apply it to your document.

74.2. Applying a theme

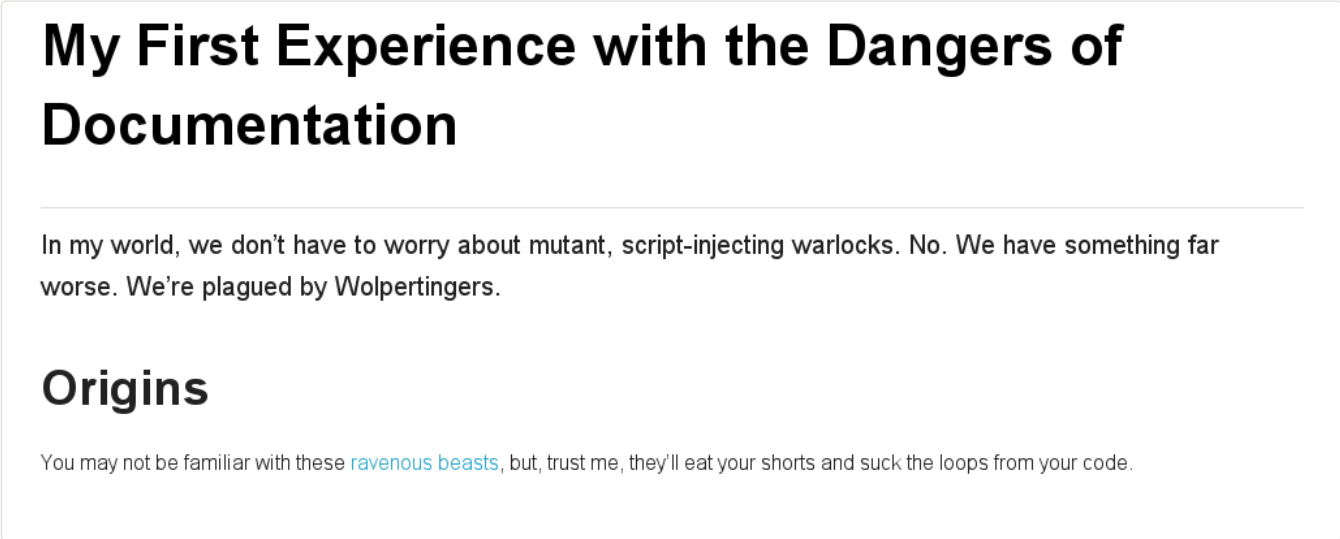
A custom stylesheet can be stored in the same directory as your document or in a separate directory. And, like the default stylesheet, you can link your document to your custom stylesheet or embed it. If the stylesheet is in the same directory as your document, you can apply it when rendering your document to HTML from the CLI.

```
$ asciidoctor -a stylesheet=mystyles.css mysample.adoc
```

1. Save your custom stylesheet in the same directory as `mysample.adoc`
2. Call the `asciidoctor` processor
3. Set `-a` (`--attribute`) and `stylesheet`
4. Assign the stylesheet file's name to the `stylesheet` attribute
5. Enter your document file's name.

Alternatively, let's set the `stylesheet` attribute in the header of `mysample.adoc`.

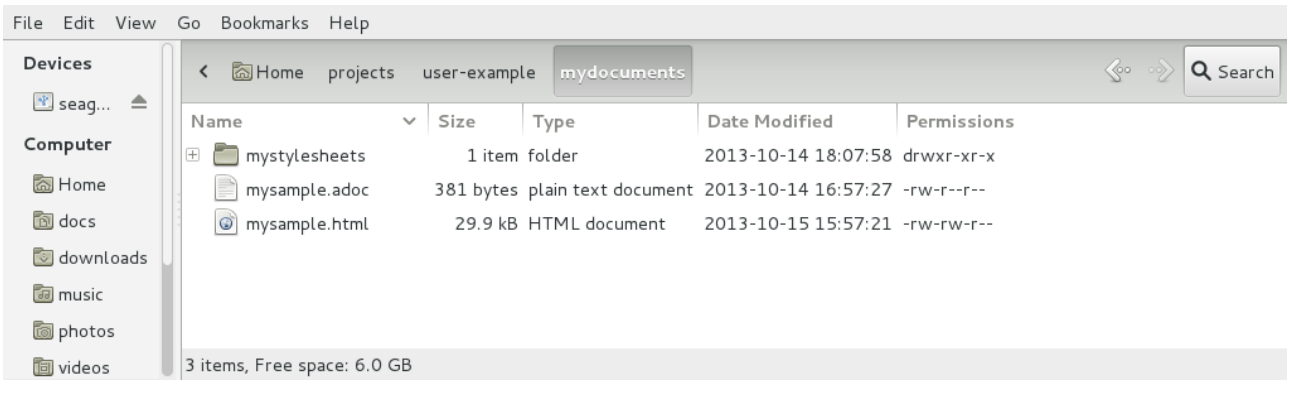
```
= My First Experience with the Dangers of Documentation
:stylesheet: mystyles.css

In my world, we don't have to worry about mutant, script-injecting warlocks.
No.
We have something far worse.
We're plagued by Wolpertingers.

== Origins

You may not be familiar with these http://en.wikipedia.org/wiki/Wolpertinger[ravenous beasts], but, trust me, they'll eat y
```



When your document and the stylesheet are stored in different directories, you need to tell Asciidoctor where to look for the stylesheet in relation to your document. Asciidoctor uses the relative or absolute path you assign to the `stylesdir` attribute to find the stylesheet. Let's move `mystyles.css` into `mydocuments/mystylesheets/`. Our AsciiDoc document, `mysample.adoc`, is saved in the `mydocuments/` directory.

```
= My First Experience with the Dangers of Documentation
:stylesdir: mystylesheets/
:stylesheet: mystyles.css

In my world, we don't have to worry about mutant, script-injecting warlocks.
No.
We have something far worse.
We're plagued by Wolpertingers.

== Origins

You may not be familiar with these http://en.wikipedia.org/wiki/Wolpertinger[ravenous beasts], but, trust me, they'll eat y
```

After processing `mysample.adoc`, its HTML output (`mysample.html`), which includes the embedded `mystyles.css` stylesheet, is created in the `mydocuments/` directory.



You can also set `stylesdir` in the CLI.

```
$ asciidoctor -a stylesdir=mystylesheets/ -a stylesheet=mystyles.css mysample.adoc
```

If you don't want to embed the `mystyles.css` stylesheet into your HTML output, make sure to set `linkcss`.

```
= My First Experience with the Dangers of Documentation
:stylesdir: mystylesheets/
:stylesheet: mystyles.css
:linkcss:

In my world, we don't have to worry about mutant, script-injecting warlocks.
No.
We have something far worse.
We're plagued by Wolpertingers.

== Origins

You may not be familiar with these http://en.wikipedia.org/wiki/Wolpertinger[ravenous beasts], but, trust me, they'll eat y
```

After your document is rendered, notice that a copy of `mystyles.css` was not created in the `mydocuments/` directory. Unlike when you link to the default Asciidoctor stylesheet, any custom stylesheets you link to are not copied to the directory where your output is saved.

Stylesheets and processing multiple nested documents

When you are running Asciidoctor once across a nested set of documents, it's currently not possible to specify a single relative path for the `stylesdir` attribute that would work for all of the documents. This is because the relative depth of the stylesheet's location differs for the documents in the subdirectories. One way to solve this problem is to maintain the path to `stylesdir` in each document.

Let's say you have three AsciiDoc documents saved in the following directory structure:

```
/mydocuments
  a.adoc
  b.adoc
  /mynesteddocuments
    c.adoc
  /mystylesheets
```

For `a.adoc` and `b.adoc`, set `stylesdir` to:

```
/:stylesdir: mystylesheets
```

For `c.adoc`, set `stylesdir` to:

```
/:stylesdir: ./mystylesheets
```

If you're serving your documents from a webserver, you can solve this problem by providing an absolute path to the stylesheet.

75. Stylesheet Factory

The Asciidoctor stylesheet factory (http://github.com/asciidoctor/asciidoctor-stylesheet-factory) is where themes are developed for styling your documents. Specifically, these stylesheets can be used to quickly customize the look and feel of HTML 5 documents generated by Asciidoctor.

To view the Asciidoctor themes in action, visit the theme showcase (http://themes.asciidoctor.org/preview/).

The Asciidoctor stylesheet factory is only compatible with Asciidoctor 0.1.2 and greater.

75.1. Setting up the factory

The stylesheets in the Asciidoctor stylesheet factory are built using Compass (http://compass-style.org), a CSS authoring framework that uses Sass (http://sass-lang.com) to generate CSS files. The styles and components are generated by Foundation 4 (http://foundation.zurb.com), an awesome and flexible CSS component framework that ensures your stylesheet is cross-browser and mobile friendly.

75.1.1. Install the gems

In order to build the stylesheets, you must download the Asciidoctor stylesheet factory repository and install the Compass and Foundation gems.

1. Download or clone the Asciidoctor stylesheet factory repository (http://github.com/asciidoctor/asciidoctor-stylesheet-factory).

It does not matter where you save the project on your system.

2. Make sure you have Ruby and RubyGems (http://www.ruby-lang.org/en/downloads) installed, and, ideally, Bundler (http://rubygems.org/gems/bundler).
3. Run Bundler to install the Compass and Foundation gems.

```
$ bundle install
```

The previous `bundle` command is equivalent to the following two commands:

```
$ gem install compass --version 0.12.2
$ gem install zurb-foundation --version 4.3.2
```

You don't need to execute these `gem install` commands if you use Bundler.

Once you have the gems installed, you can build the stylesheets.

75.1.2. Build the stylesheets

To build the stylesheets:

1. Navigate to the Asciidoctor stylesheet factory directory on your system.
2. Run Compass's `compile` command.

```
$ compass compile
```

The stylesheets are compiled from the Sass source files in the *sass/* folder and written to the *stylesheets/* folder. You can reference the stylesheets in *stylesheets/* from your HTML file.

75.2. Applying a stylesheet

Let's practice applying a stylesheet to a simple AsciiDoc file.

1. Create an AsciiDoc file like the one below.
2. Save the file as *sample.adoc*.

```
= Introduction to AsciiDoc
Doc Writer <doc@example.com>

A preface about http://asciidoc.org[AsciiDoc].

== First Section

* item 1
* item 2

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

Next, you'll use the Asciidoctor processor to generate HTML and apply a stylesheet to it from the *stylesheets/* directory.

75.3. Generate an HTML document

Now it's time to pick the stylesheet you want to apply to your content when it is rendered. In your file browser, navigate to the *stylesheets/* directory. Or, using a console, change to the *stylesheets/* directory and list the available stylesheets using the `ls` command.

```
$ ls
```

Console output of `ls` command

```
asciidoctor.css  foundation-lime.css    iconic.css       riak.css
colony.css       foundation-potion.css  maker.css        rubygems.css
foundation.css   golo.css               readthedocs.css
```

Let's apply the *colony.css* stylesheet to the sample document.

1. Navigate to the directory where you saved *sample.adoc*.
2. Call the `asciidoctor` processor.
3. Specify the stylesheet you want applied with the `stylesheet` attribute.
4. Tell the processor where the specified stylesheet is located with the `stylesdir` attribute.

   ```
   $ asciidoctor -a stylesheet=colony.css -a stylesdir=../stylesheets sample.adoc
   ```

Open a browser, navigate to *sample.html* and checkout the result! If you inspect the *sample.html* document, you should see that the stylesheet is embedded in the rendered document.

Stylesheet images

The Golo, Maker, and Riak themes include image assets. To apply these themes to your document correctly, the applicable images must be copied into the same directory as the generated output.

For example, to apply the *maker.css* stylesheet to *sample.adoc*:

1. Copy *body-bh.png* from the *images/maker/* directory into the output directory.
2. Call the `stylesheet` and `styledir` attributes.

   ```
   $ asciidoctor -a stylesheet=maker.css -a stylesdir=../stylesheets sample.adoc
   ```

Navigate to *sample.html* in your browser. The *body-bh.png* image should add a graph paper-like background to your generated output.

75.4. External preview

You may want to preview sample HTML files on another computer or device. To do that, you need to serve them through a web server. You can quickly serve HTML files in the root directory of the project using the following command:

Using Python

```
$ python -m SimpleHTTPServer 4242
```

or

Using Ruby >= 1.9.3

```
$ ruby -run -e httpd . -p 4242
```

76. Slideshows

The conversion to HTML-based slides is handled by a custom backend. The backends usually requires that you adhere to a set of conventions to define a slide and its content. But there's nothing in that content that restricts the document from being converted to a regular HTML page or PDF.

The first backend created was the one for deck.js. Asciidoctor has backends for generating dzslides and reveal.js presentations as well. Backends for other presentation frameworks are in the works.

76.1. Deck.js

The deck.js backend (http://github.com/asciidoctor/asciidoctor-backends) for Asciidoctor is a collection of Haml templates that transform an AsciiDoc document to HTML 5-based slides animated by deck.js (http://imakewebthings.com/deck.js).

76.1.1. Gem requirements

The Asciidoctor deck.js backend requires the following gems:

- asciidoctor
- tilt
- haml

To determine what gems are installed on your system, open a terminal window and type:

```
$ gem list --local
```

A list of installed gems will be returned.

If you're missing the `asciidoctor`, `tilt`, or `haml` gems, install them from the command line with `gem install`.

```
$ gem install asciidoctor tilt haml
```

76.1.2. Backend and deck.js installation

1. Download or `git clone` the Asciidoctor backends (http://github.com/asciidoctor/asciidoctor-backends) repository.

   ```
   $ git clone git://github.com/asciidoctor/asciidoctor-backends.git
   ```

2. Create a directory named `deck.js` inside your working directory (i.e. where your AsciiDoc document resides).
3. Download (https://github.com/imakewebthings/deck.js/archive/latest.zip) and extract the deck.js archive or `git clone` the deck.js (https://github.com/imakewebthings/deck.js) repository into the `deck.js` directory you created in step 2.

   ```
   $ git clone git://github.com/imakewebthings/deck.js.git
   ```

4. If you plan to split your slides, download the `deck.split.js` extension (https://github.com/houqp/deck.split.js) and copy it into the `deck.js/extensions` directory.
5. If you want to use the fullscreen photo feature, create an `images` directory in your working directory.

76.1.3. Deckjs backend attributes

There are a number of document attributes specific to the `deckjs` backend.

`deckjs` *backend document attributes*

Attribute	Description	Example
:backend: deckjs	Activates the deck.js backend to render the document as a deck.js presentation	`:backend: deckjs`
:deckjs_theme: <theme>		`:deckjs_theme: web-2.0`

	Sets the deck.js theme to neon, swiss or web-2.0	
:deckjs_transition: <type>	Sets the transition style to horizontal-slide, vertical-slide or fade	`:deckjs_transition: fade`
:customjs: <javascript location>	Sets a custom javascript file; can be used as a deck.js custom configuration	`:customjs: <javascript location>`
:customcss: <css location>	Sets a custom css file	`:customcss: <css location>`
:menu:	Toggle to and from a grid layout overview of all the slides by pressing the `m` key	`:menu:`
:navigation:	Renders clickable previous and next navigation icons on the slides	`:navigation:`
:status:	Renders the current slide number and total number of slides	`:status:`
:split:	Registers the split module for use in the document	`:split:`

You can also specify a custom stylesheet using the `stylesheet` attribute, which customizes AsciiDoc elements like section, paragraph, images, etc.

The attributes described in the table above are set in the header of your document.

Header with `deckjs` backend attributes

```
= Presentation Title
Presenter Name
:backend: deckjs
:deckjs_theme: web-2.0
:deckjs_transition: horizontal-slide
:navigation:
```

76.1.4. Slide syntax examples

Structuring a slideshow and writing the slide content uses the same syntax as a typical AsciiDoc document, with a few added features.

Let's see some examples of the `deckjs` backend features:

Slide titles and background images

```
= Title of Presentation (1)
Presenter Name
:backend: deckjs
:deckjs_transition: fade
:navigation:

== Title of Slide One (2)

This is the first slide after the title slide.

[canvas-image="images/example.jpg"] (3)
== Slide Two's Title will not be displayed (4)

[role="canvas-caption", position="center-up"] (5)
This text is displayed on top of the example.jpg image.
```

1. The presentation title and author's name will be displayed on the title slide.
2. Each new slide is designated by a level 1 section title (`==`).
3. The `canvas-image` attribute embeds a fullscreen image as a slide background. Position the attribute above the title of the slide you want the image applied to.
4. When the `canvas-image` attribute is applied to a slide, that slide's title will not be displayed.
5. `canvas-caption` applies a colored box around the caption text. `position` specifies the location of the caption block (bottom-left, top-left, bottom-right, top-right, center-up, center-down)

Stepped paragraphs, lists, and blocks

```
== Stepped paragraphs

[options="step"]
This paragraph is displayed first.

[options="step"]
Then this paragraph is displayed when the _Next_ arrow is clicked.

== Stepped list items

[options="step"]
* A bullet is displayed each time the _Next_ arrow is clicked.
* B
* C

== Stepped blocks

[options="step"]
```

Block one

```
[options="step"]
```

Block two

```
```

The `step` option reveals each paragraph, bullet, etc. separately each time you click the *Next* arrow.

The original AsciiDoc `deckjs` backend for the AsciiDoc processor used the option `incremental` instead of `step`. We've changed it to `step` in order to save you some typing.

Split

```
= Presentation Title
Presenter Name
:backend: deckjs
:split: (1)

== This Slide is Split

This Slide will act like

<<< (2)

three individual slides with the same title

<<<

once the document is rendered.
```

(1) To create multiple, consecutive slides with the same title, set the `split` attribute in the document header.

(2) Then, within a slide, insert `<<<` to specify the slide breaks.

76.1.5. Rendering

To render your presentation as HTML5, execute the command:

```
$ asciidoctor -T ../asciidoctor-backends/haml presentation.adoc
```

1. The command `-T` (`--template-dir`) tells the Asciidoctor processor to override the built-in backends.
2. Directly after `-T` is the path to where you saved or cloned the Asciidoctor backends repository containing the `deckjs` backend (step 1 under the installation section).

77. Custom Backends

Section pending

77.1. Creating a backend

Section pending

77.1.1. Storing multiple templates

Custom templates can be stored in multiple directories. That means you can build on an existing backend by copying only the templates you want to modify. Then, just pass both the directory holding the original templates and the directory containing your customized templates when you invoke Asciidoctor.

In the CLI, multiple template directories are specified by using the `-T` option multiple times.

```
$ asciidoctor -T /path/to/original/templates -T /path/to/modified/templates sample.adoc
```

In the API, multiple template directories are specified by passing an array to the `template_dirs` option:

RUBY

```ruby
Asciidoctor.convert_file 'sample.adoc', :safe => :safe, :in_place => true,
    :template_dirs => ['/path/to/original/templates', '/path/to/modified/templates']
```

Publishing Your Content

NOTE: Section pending

78. Repositories

Section pending

79. Static website generators

Section pending

79.1. Front matter added for static site generators

Many static site generators (i.e., Jekyll, Middleman, Awestruct) rely on "front matter" added to the top of the document to determine how to render the content. Front matter typically starts on the first line of a file and is bounded by block delimiters (e.g., `---`).

Here's an example of a document that contains front matter:

```
-(1)
layout: default (2)
-(1)
= Document Title

content
```

ASCIIDOC

(1) Front matter delimiters

(2) Front matter data

The static site generator removes these lines before passing the document to the AsciiDoc processor to be rendered. Outside of the tool, however, these extra lines can throw off the processor.

If the `skip-front-matter` attribute is set, Asciidoctor 0.1.4 will recognize the front matter and consume it before parsing the document. Asciidoctor stores the content it removes in the `front-matter` attribute to make it available for integrations. Asciidoctor also removes front matter when reading include files.

Awestruct can read front matter directly from AsciiDoc attributes defined in the document header, thus eliminating the need for this feature.

79.1.1. Configuring attributes for Awestruct

Awestruct defines a set of default attributes that it passes to the API in its */default-site.yml* file. One of the attributes in that configuration is `imagesdir`. The value there is set to */images*. That means the value in your document is skipped due to the precedence rules.

Fortunately, there is one additional place you can override the attribute. This gives you the opportunity to set your own default and to flip the precedence order so that the document wins out. If an attribute value that is passed to the API ends with an `@` symbol, it makes that assignment have a lower precedence than an assignment in the document.

You can define attributes you want to pass to the API in the *_config/site.yml* file. Here's an example entry for Asciidoctor:

```yaml
asciidoctor:
  :safe: safe
  :attributes:
    imagesdir: /assets/images@
    icons: font
    ...
```

YAML

The second-level keys (safe and attributes, in this case) must have colons on both sides of the key name. The rest of the keys only have a colon after the key.

With this configuration added, you should observe that the `imagesdir` attribute in your document is now respected.

Using Asciidoctor's API

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

Asciidoctor also has a Java API that mirrors the Ruby API. The Java API calls through to the Ruby API using an embedded JRuby runtime. See the Asciidoctor Java integration project (http://asciidoctor.org/docs/install-and-use-asciidoctor-java-integration) for more information.

80. Load and Render a File Using the API

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

```ruby
require 'asciidoctor'
```

With that in place, you can start processing AsciiDoc documents. To parse a file into an `Asciidoctor::Document` object:

```ruby
doc = Asciidoctor.load_file 'sample.adoc'
```

You can get information about the document:

```ruby
puts doc.doctitle
puts doc.attributes
```

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

```ruby
Asciidoctor.convert_file 'sample.adoc', :in_place => true
```

The command will output to the file `sample.html` in the same directory.

You can render the file to DocBook 5.0 by setting the `:backend` option to *`docbook`*:

```ruby
Asciidoctor.convert_file 'sample.adoc', :in_place => true, :backend => 'docbook'
```

The command will output to the file `sample.xml` in the same directory. If you're on Linux, you can view the file using Yelp (https://wiki.gnome.org/action/show/Apps/Yelp).

80.1. Render strings

To render an AsciiDoc-formatted string:

```ruby
puts Asciidoctor.convert '*This* is Asciidoctor.'
```

When rendering a string, the header and footer are excluded by default to make Asciidoctor consistent with other lightweight markup engines like Markdown. If you want the header and footer, just enable it using the `:header_footer` option:

```ruby
puts Asciidoctor.convert '*This* is Asciidoctor.', :header_footer => true
```

Now you'll get a full HTML 5 file. If you only want the inline markup to be processed, set the `:doctype` option to *`inline`*:

```ruby
puts Asciidoctor.convert '*This* is Asciidoctor.', :doctype => 'inline'
```

As before, you can also produce DocBook 5.0:

```ruby
puts Asciidoctor.convert '*This* is Asciidoctor.', :header_footer => true,
  :backend => 'docbook'
```

When rendering a string the TOC is only included by default when using the `:header_footer` option as shown above. However, you can force it to be included without the header and footer by setting the `toc` attribute with a value of `macro` and using the `toc::[]` macro in the string itself.

If you don't like the output you see, you can change it. Any of it!

81. Provide custom templates

Asciidoctor allows you to override the converter methods used to render almost any individual AsciiDoc element. If you provide a directory of Tilt (https://github.com/rtomayko/tilt)-compatible templates, named in such a way that Asciidoctor can figure out which template goes with which element, Asciidoctor will use the templates in this directory instead of its built-in templates for any elements for which it finds a matching template. It will fallback to its default templates for everything else.

```ruby
puts Asciidoctor.convert '*This* is Asciidoctor.', :header_footer => true,
  :template_dir => 'templates'
```

The Document and Section templates should begin with `document.` and `section.`, respectively. The file extension is used by Tilt to determine which view framework it will use to use to render the template. For instance, if you want to write the template in ERB, you'd name these two templates `document.html.erb` and `section.html.erb`. To use Haml, you'd name them `document.html.haml` and `section.html.haml`.

Templates for block elements, like a Paragraph or Sidebar, would begin with `block_<style>.`. For instance, to override the default Paragraph template with an ERB template, put a file named `block_paragraph.html.erb` in the template directory you pass to the `Document` constructor using the `:template_dir` option.

For more usage examples, see the (massive) test suite (https://github.com/asciidoctor/asciidoctor/tree/master/test).

Extensions

Extensions have proven to be central to the success of AsciiDoc and the Python implementation. Despite the success they've enjoyed, there's still *plenty* of room for improvement.

Extensions in AsciiDoc (technically called filters) have a number of problems:

- they are challenging to write because they work at such a low-level (read as: nasty regular expressions)
- they are fragile since they rely on system commands to do anything significant
- they are hard to distribute due to the lack of integration with a formal distribution system

The goal for Asciidoctor has always been to allow extensions to be written using the full power of a programming language (whether it be Ruby, Java, Groovy or JavaScript), similar to what we've done with the backend (rendering) mechanism. That way, you don't have to shave yaks to get the functionality you want, and you can distribute the extension using defacto-standard packaging mechanisms (like RubyGems or JARs).

The extension API is available as a technology preview in Asciidoctor 0.1.4.

Technology Preview

The extension API should be considered **experimental** and **subject to change**. This technology preview is intended for developers interested in playing around with it and helping to mature the design.

If you need the capabilities that extensions provide now, don't be afraid to jump on board. Just keep in mind that you may need to rework parts of your extensions when you upgrade Asciidoctor.

82. Extension Points

Here are the extension points that are available in Asciidoctor 0.1.4.

Preprocessor

Processes the raw source lines before they are passed to the parser.

Treeprocessor

Processes the Asciidoctor::Document (AST) once parsing is complete.

Postprocessor

Processes the output after the document has been rendered, but before it's written to disk.

Block processor

Processes a block of content marked with a custom block style (i.e., `[custom]`). (similar to an AsciiDoc filter)

Block macro processor

Registers a custom block macro and processes it (e.g., `gist::12345[]`).

Inline macro processor

Registers a custom inline macro and processes it (e.g., `btn:[Save]`).

Include processor

Processes the `include::<filename>[]` directive.

These extensions are registered per document using a callback that feels sort of like a DSL:

```ruby
Asciidoctor::Extensions.register do |document|
  preprocessor FrontMatterPreprocessor
  treeprocessor ShellSessionTreeprocessor
  postprocessor CopyrightFooterPostprocessor
  block ShoutBlock
  block_macro GistBlockMacro if document.basebackend? 'html'
  inline_macro ManInlineMacro
  include_processor UriIncludeProcessor
end
```

Extension classes must be defined outside of the register block. Once an extension class is registered, it is frozen, preventing further modification. If you define an extension class inside the register block, it will result in an error on subsequent invocations.

You can register more than one processor of each type, though you can only have one processor per custom block or macro. Each registered class is instantiated when the Asciidoctor::Document is created.

There is currently no extension point for processing a built-in block, such as a normal paragraph. Look for that feature in a future Asciidoctor release.

83. Example Extensions

Below are several examples of extensions and how they are registered.

83.1. Preprocessor example

Purpose

Skim off front matter from the top of the document that gets used by site generators like Jekyll and Awestruct.

sample-with-front-matter.adoc

```asciidoc
---
tags: [announcement, website]
---
= Document Title

content

[subs="attributes,specialcharacters"]
.Captured front matter
....
---
{front-matter}
---
....
```

FrontMatterPreprocessor

```ruby
require 'asciidoctor'
require 'asciidoctor/extensions'

class FrontMatterPreprocessor < Asciidoctor::Extensions::Preprocessor
  def process document, reader
    lines = reader.lines # get raw lines
    return reader if lines.empty?
    front_matter = []
    if lines.first.chomp == '---'
      original_lines = lines.dup
      lines.shift
      while !lines.empty? && lines.first.chomp != '---'
        front_matter << lines.shift
      end

      if (first = lines.first).nil? || first.chomp != '---'
        lines = original_lines
      else
        lines.shift
        document.attributes['front-matter'] = front_matter.join.chomp
        # advance the reader by the number of lines taken
        (front_matter.length + 2).times { reader.advance }
      end
    end
    reader
  end
end
```

Usage

```ruby
Asciidoctor::Extensions.register do
  preprocessor FrontMatterPreprocessor
end

Asciidoctor.convert_file 'sample-with-front-matter.adoc', :safe => :safe
```

83.2. Treeprocessor example

Purpose

Detect literal blocks that contain shell commands, strip the prompt character and style the command using CSS in such a way that the prompt character cannot be selected (as seen on help.github.com).

sample-with-shell-session.adoc

```
$ echo "Hello, World!"
> Hello, World!

$ gem install asciidoctor
```
ASCIIDOC

ShellSessionTreeprocessor

```ruby
class ShellSessionTreeprocessor < Asciidoctor::Extensions::Treeprocessor
  def process document
    return unless document.blocks?
    process_blocks document
    nil
  end

  def process_blocks node
    node.blocks.each_with_index do |block, i|
      if block.context == :literal &&
          (((first_line = block.lines.first).start_with? '$ ') ||
            (first_line.start_with? '> '))
        node.blocks[i] = convert_to_terminal_listing block
      else
        process_blocks block if block.blocks?
      end
    end
  end

  def convert_to_terminal_listing block
    attrs = block.attributes
    attrs['role'] = 'terminal'
    prompt_attr = (attrs.has_key? 'prompt') ?
        %( data-prompt="#{block.sub_specialchars attrs['prompt']}") : nil
    lines = block.lines.map do |line|
      line = block.sub_specialchars line.chomp
      if line.start_with? '$ '
        %(<span class="command"#{prompt_attr}>#{line[2..-1]}</span>)
      elsif line.start_with? '&gt; '
        %(<span class="output">#{line[5..-1]}</span>)
      else
        line
      end
    end
    create_listing_block block.document, lines * EOL, attrs, subs: nil
  end
end
```
RUBY

Usage

```ruby
Asciidoctor::Extensions.register do |document|
  treeprocessor ShellSessionTreeprocessor
end

Asciidoctor.convert_file 'sample-with-shell-session.adoc', :safe => :safe
```
RUBY

83.3. Postprocessor example

Purpose

Insert copyright text in the footer.

CopyrightFooterPostprocessor

```ruby
class CopyrightFooterPostprocessor < Asciidoctor::Extensions::Postprocessor
  def process document, output
    content = (document.attr 'copyright') || 'Copyright Acme, Inc.'
    if document.basebackend? 'html'
      replacement = %(<div id="footer-text">\\1<br>\n#{content}\n</div>)
      output = output.sub(/<div id="footer-text">(.*?)<\/div>/m, replacement)
    elsif document.basebackend? 'docbook'
      replacement = %(<simpara>#{content}</simpara>\n\\1)
      output = output.sub(/(<\/(?:article|book)>)/, replacement)
    end
    output
  end
end
```
RUBY

Usage

```
Asciidoctor::Extensions.register do
  postprocessor CopyrightFooterPostprocessor
end

Asciidoctor.convert_file 'sample-with-copyright-footer.adoc', :safe => :safe
```
ASCIIDOC

83.4. Block processor example

Purpose

Register a custom block style named `shout` that uppercases all the words and converts periods to exclamation points.

sample-with-shout-block.adoc

```
[shout]
The time is now. Get a move on.
```
ASCIIDOC

ShoutBlock

```
require 'asciidoctor'
require 'asciidoctor/extensions'

class ShoutBlock < Asciidoctor::Extensions::BlockProcessor
  PeriodRx = /\.(?= |$)/

  use_dsl

  named :shout
  on_context :paragraph
  name_positional_attributes 'vol'
  parse_content_as :simple

  def process parent, reader, attrs
    volume = ((attrs.delete 'vol') || 1).to_i
    create_paragraph parent, (reader.lines.map {|l| l.upcase.gsub PeriodRx, '!' * volume }), attrs
  end
end
```
RUBY

Usage

```
Asciidoctor::Extensions.register do
  block ShoutBlock
end

Asciidoctor.convert_file 'sample-with-shout-block.adoc', :safe => :safe
```
RUBY

83.5. Block macro processor example

Purpose

Create a block macro named `gist` for embedding a gist.

sample-with-gist-macro.adoc

```
.My Gist
gist::123456[]
```
ASCIIDOC

GistBlockMacro

```ruby
require 'asciidoctor'
require 'asciidoctor/extensions'

class GistBlockMacro < Asciidoctor::Extensions::BlockMacroProcessor
  use_dsl

  named :gist

  def process parent, target, attrs
    title_html = (attrs.has_key? 'title') ?
        %(<div class="title">#{attrs['title']}</div>\n) : nil

    html = %(<div class="openblock gist">
#{title_html}<div class="content">
<script src="https://gist.github.com/#{target}.js"></script>
</div>
</div>)

    create_pass_block parent, html, attrs, subs: nil
  end
end
```

Usage

```ruby
Asciidoctor::Extensions.register do
  block_macro GistBlockMacro if document.basebackend? 'html'
end

Asciidoctor.convert_file 'sample-with-gist.adoc', :safe => :safe
```

83.6. Inline macro processor example

Purpose

Create an inline macro named `man` that links to a manpage.

sample-with-man-link.adoc

```asciidoc
See man:gittutorial[7] to get started.
```

ManpageInlineMacro

```ruby
require 'asciidoctor'
require 'asciidoctor/extensions'

class ManInlineMacro < Asciidoctor::Extensions::InlineMacroProcessor
  use_dsl

  named :man
  name_positional_attributes 'volnum'

  def process parent, target, attrs
    text = manname = target
    suffix = ''
    target = %(#{manname}.html)
    suffix = if (volnum = attrs['volnum'])
      "(#{volnum})"
    else
      nil
    end
    parent.document.register :links, target
    %(#{(create_anchor parent, text, type: :link, target: target).convert}#{suffix})
  end
end
```

Usage

```ruby
Asciidoctor::Extensions.register do
  inline_macro ManInlineMacro
end

Asciidoctor.convert_file 'sample-with-man-link.adoc', :safe => :safe
```

83.7. Include processor example

Purpose

Include a file from a URI.

Asciidoctor supports including content from a URI out of the box if you set the `allow-uri-read` attribute (not available if the safe mode is `secure`).

sample-with-uri-include.adoc

```asciidoc
:source-highlighter: coderay

.Gemfile
[source,ruby]
----
include::https://raw.githubusercontent.com/asciidoctor/asciidoctor/master/Gemfile[]
----
```

UriIncludeProcessor

```ruby
require 'asciidoctor'
require 'asciidoctor/extensions'
require 'open-uri'

class UriIncludeProcessor < Asciidoctor::Extensions::IncludeProcessor
  def handles? target
    (target.start_with? 'http://') or (target.start_with? 'https://')
  end

  def process doc, reader, target, attributes
    content = (open target).readlines
    reader.push_include content, target, target, 1, attributes
    reader
  end
end
```

Usage

```ruby
Asciidoctor::Extensions.register do
  include_processor UriIncludeProcessor
end

Asciidoctor.convert_file 'sample-with-uri-include.adoc', :safe => :safe
```

You can see plenty more extension examples in the extensions lab (https://github.com/asciidoctor/asciidoctor-extensions-lab).

Build Integrations and Implementations

NOTE: Section pending

84. Java

Section pending

85. Gradle

Section pending

86. Maven

Section pending

87. JavaDoc

Section pending

88. JavaScript

Section pending

89. Yard

Section pending

90. Rdoc

Section pending

Conversions and Migrations

NOTE: Section pending

91. Convert Markdown to Asciidoctor

Section pending

92. Migrate from AsciiDoc to Asciidoctor

Section pending

Resources

NOTE: Section pending

93. Copyright and License

94. Authors

Asciidoctor was written by Dan Allen (https://github.com/mojavelinux), Ryan Waldron (https://github.com/erebor), Jason Porter (https://github.com/lightguard), Nick Hengeveld (http://github.com/nickh) and other contributors (https://github.com/asciidoctor/asciidoctor/graphs/contributors).

The initial code from which Asciidoctor emerged was written by Nick Hengeveld (http://github.com/nickh) to process the git man pages for the Git project site (https://github.com/github/gitscm-next). Refer to the commit history of asciidoc.rb (https://github.com/github/gitscm-next/commits/master/lib/asciidoc.rb) to view the initial contributions.

AsciiDoc was written by Stuart Rackham and has received contributions from many other individuals.

95. Troubleshooting

1. *Part way through the document, the blocks stop rendering correctly. What went wrong?*

   When content is not rendered as you expect in the later part of a document, it's usually due to a delimited block missing its closing delimiter line. The most devious culprit is the open block. An open block doesn't have any special styling, but its contents have the same restrictions as other delimited blocks, i.e. it can not contain section titles.

   To solve this problem, first look for missing delimiter lines. Syntax highlighting in your text editor can help with this. Also look at the rendered output to see if the block styles are extending past where you intended. When working with open blocks, you may need to add custom styles (such as a red border) to the class selector `.openblock` so that you can see its boundaries.

2. *Why don't URLs containing underscores (`_`) or carets (`{caret}`) work after they're rendered?*

This problem occurs because the markup parser interprets parts of the URL (i.e., the link target) as valid text formatting markup. Most lightweight markup languages have this issue because they don't use a grammar-based parser. Asciidoctor plans to handle URLs more carefully in the future (see issue #281 (https://github.com/asciidoctor/asciidoctor/issues/281)), which may be solved by moving to a grammar-based parser (see issue #61 (https://github.com/asciidoctor/asciidoctor/issues/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: http://www.asciidoctor.org/now_this__link_works.html

This URL has repeating underscores {link-with-underscores}.
```
ASCIIDOC

Asciidoctor won't break links with underscores when they are assigned to an attribute because inline formatting markup is substituted before attributes. The URL remains hidden while the rest of the document is being formatted (strong, emphasis, monospace, etc).

Solution B

Another way to solve formatting glitches is to explicitly specify the formatting you want to have applied to a span of text. This can be done by using the inline pass macro. If you want to display a URL, and have it preserved, put it inside the pass macro and enable the macros substitution, which is what substitutes links.

```
This URL has repeating underscores pass:macros[http://www.asciidoctor.org/now_this__link_works.html].
```
ASCIIDOC

The pass macro removes the URL from the document, applies the `macros` substitution to the URL, and then restores the processed URL to its original location once the substitutions are complete on the whole document.

Alternatively, you can use `++` around the URL only. However, when you use this approach, Asciidoctor won't recognize it as a URL anymore, so you have to use the explicit `link` prefix.

```
This URL has repeating underscores link:++http://www.asciidoctor.org/now_this__link_works.html++[].
```
ASCIIDOC

For more information, see issue #625 (https://github.com/asciidoctor/asciidoctor/issues/625).

96. Additional Help

Section pending

97. Glossary

admonition paragraph

a callout paragraph that has a label or icon indicating its priority

admonition block

a callout block containing complex content that has a label or icon indicating its priority

backend

a set of templates for converting AsciiDoc source to different output format

cross reference

a link from one location in the document to another location marked by an anchor

list continuation

a plus sign (`+`) on a line by itself that connects adjacent lines of text to a list item

quoted text

text which is enclosed in special punctuation to give it emphasis or special meaning

Section expansion pending

Appendix A: Comparison of Asciidoctor and AsciiDoc Features

The table below lists the AsciiDoc and Asciidoctor attributes and values. Checkmarks (✔) indicate whether the AsciiDoc or Asciidoctor processor can process that feature. Features marked with the file icon (📄) can only be processed by the AsciiDoc processor with the inclusion of the Asciidoctor compatibility file.

The compatibility file makes the AsciiDoc processor conform with Asciidoctor's fixes and customizations. Place the file in the same directory as your AsciiDoc document and the AsciiDoc processor (`asciidoc`) will automatically use it.

Name	AsciiDoc	Asciidoctor	Notes
90 (icon)	✖	✔	
180 (icon)	✖	✔	
270 (icon)	✖	✔	
2x (icon)	✖	✔	
4x (icon)	✖	✔	
`a`	✖	✔	Can only be assigned to the inline pass macro.
`abstract`	✔	✔	
`admonition`	✔	✔	
`align` (image)	✔	✔	
`align` (table)	✔	✖	
`all` (table)	✔	✔	Value for frame
`all` (table)	✔	✔	Value for grid
`allow-uri-read`	✖	✔	
`alt` (icon)	📄	✔	
`alt` (image, link)	✔	✔	
`appendix`	✔	✔	
`aqua`	✔	✔	
`arabic`	✔	✔	
`article`	✔	✔	
`ascii-ids`	✔	✖	See Using UTF-8 titles (in dblatex) with a2x (http://aerostitch.github.io/misc/asciidoc/asciidoc-title_uft8.html)
`asciidoctor`	N/A	✔	
`asciidoctor-safe`	N/A	✔	
`asciidoctor-version`	N/A	✔	
`asciimath`	✔	✔	
`attribute-missing`	✖	✔	
`attribute-undefined`	✖	✔	
`attributes`	✔	✔	Value for `subs`

`attribution`	✔	✔	
`audio::`	✔	✔	
`author`	✔	✔	
`authorgroup`	✘	✔	
`authorinitials`	✔	✔	
`autoplay`	✔	✔	
`autowidth` (table)	✔	✔	
`backend`, `-b`	✔	✔	
`-background` (quoted text)	✔	✔	
`--base-dir`, `-B`	✔	✔	
`bibliography`	✔	✔	
`big`	✔	✔	
`black`	✔	✔	
`_blank`, `^` (link)	✘	✔	
`blue`	✔	✔	
`book`	✔	✔	
`bottom` (table)	✔	✔	Value for valign
`breakable` (table, example, block image)	✔	✔	
`brvbar`	📄	✔	
`btn`	📄	✔	
`c`	✘	✔	Can only be assigned to the inline pass macro
`cache-uri`	✘	✔	
`callouts` (subs)	✔	✔	
`caption`	✔	✔	
`CAUTION`	✔	✔	
`caution-caption`	✘	✔	
`center` (image)	✔	✔	
`center` (table)	✔	✘	Value for align
`center` (table)	✔	✘	Value for halign
`[*]`, `[X]` (checked)	✘	✔	
`circle`	✘	✔	Requires default Asciidoctor stylesheet or custom stylesheet to render properly.
`citetitle`	✔	✔	

`coderay`	✘	✔	
`coderay-css`	✘	✔	
`coderay-linenums-mode`	✘	✔	
`colophon`	✔	✔	
`cols`	✔	✔	
`cols` (table)	✔	✔	Value for grid
`comment`, `\\`, `\\\\`	✔	✔	
`compact` (list)	✔	✘	
`conf-file`, `-f`	✔	✘	
`controls`	✔	✔	
`copycss`	✘	✔	
`csv`	✔	✔	
`,` (csv shorthand)	🗎	✔	
`data-item-complete`	✘	✔	
`data-uri`	✔	✔	
`decimal`	✔	✔	
`deckjs`	TBD	✔	
`dedication`	✔	✔	
`description`	✔	✔	
`disc`	✘	✔	Requires default Asciidoctor stylesheet or custom stylesheet to render properly.
`discrete`	✘	✔	
`docbook`, `docbook45`	✔	✔	
`docbook5`	✘	✔	
`docdate`	✔	✔	
`docdatetime`	✔	✔	
`docdir`	✔	✔	
`docfile`	✔	✔	
`docinfo`	✔	✔	
`doctest`	✔	✘	
`doctime`	✔	✔	
`doctitle`	✔	✔	
`doctype`, `-d`	✔	✔	
`drop`	✘	✔	

`drop-line`	✖	✔	
`dsv`	✔	✔	
`:` (dsv shorthand)	🗎	✔	
`dump-conf`, `-c`	✔	✖	
`email`	✔	✔	
`embedded`	TBD	✔	
`encoding`	✔	✔	
`endif`	✔	✔	
`example`, `====`	✔	✔	
`example-caption`	✔	✔	
`experimental`	🗎	✔	
`external` (role, link)	TBD	✔	
`` (fenced code block)	🗎	✔	AsciiDoc can not render source-highlighting to fenced code blocks
`figure-caption`	✔	✔	
`filter`	✔	✖	
`filter` (table)	✔	✖	
`firstname`	✔	✔	
`flip` (icon)	✖	✔	
`float` (section title)	✔	✔	
`float` (image)	🗎	✔	
`float` (table)	✔	✖	
`font`	🗎	✔	
`format` (data)	✔	✔	
`frame`	✔	✔	
`footer` (table)	✔	✔	
`fuschia`	✔	✔	
`glossary`	✔	✔	
`graphviz`	✔	✖	
`gray`	✔	✔	
`green`	✔	✔	
`grid`	✔	✔	
`halign` (table)	✔	TBD	
`hardbreaks`	✖	✔	
`header` (implicit, table)	✖	✔	

`header` (table)	✔	✔	
`height` (icon)	✖	✔	
`height` (image, video)	✔	✔	
`highlightjs`	✔	✔	
`horizontal` (icon)	✖	✔	
`horizontal` (list)	✔	✔	
`html`, `html5`	✔	✔	
`icon`	✔	✔	
`icons`	✔	✔	
`iconsdir`	✔	✔	
`icontype`	✖	✔	
`id`	✔	✔	
`#` (id shorthand)	✖	✔	
`idprefix`	✔	✔	
`idseparator`	✖	✔	
`ifdef`	✔	✔	
`ifeval`	✔	✔	Asciidoctor constrains it to strictly comparing the values of attributes.
`imagesdir`	✔	✔	
`IMPORTANT`	✔	✔	
`important-caption`	✖	✔	
`include`	✔	✔	
`incremental`	✔	✔	
`indent` (include)	✖	✔	
`index`	✔	✔	
`inline` (doctype)	✖	✔	
`interactive`	✖	✔	
`kbd:`	🗎	✔	
`keywords`	✔	✔	
`lang`	✔	✔	
`large` (icon)	✖	✔	
`lastname`	✔	✔	
`latex`	✔	✖	
`latexmath`	✖	✔	

`lead`	✖	✔	Requires default Asciidoctor stylesheet or custom stylesheet to render properly.
`left` (image)	✔	✔	Value for align, float, role
`left` (table)	✔	✖	Value for align, halign
`left` (ToC)	✖	✔	
`level`	✔	✔	
`leveloffset`	✔	✔	
`lime`	✔	✔	
`lines` (include)	✖	✔	
`link`	✔	✔	
`link` (icon)	TBD	✔	
`link` (image)	✔	✔	
`linkattrs`	📄	✔	
`linkcss`	✔	✔	
`listing`, `----`	✔	✔	
`listing-caption`	✔	✔	
`literal`, `....`	✔	✔	
`line-through`	✔	✔	
`localdate`	✔	✔	
`localdatetime`	✔	✔	
`localtime`	✔	✔	
`loop`	✔	✔	
`loweralpha`	✔	✔	
`lowergreek`	✖	✔	
`lowerroman`	✔	✔	
`m`	✖	✔	Can only be assigned to the inline pass macro.
`macros`	✔	✔	
`manpage`	✔	✔	
`maroon`	✔	✔	
`max-width` (document)	✔	✔	
`menu`	📄	✔	
`middle` (table)	✔	TBD	Value for valign
`music`	✔	✖	
`navy`	✔	✔	

`no-bullet`	✘	✔	Requires default Asciidoctor stylesheet or custom stylesheet to render properly.
`no-conf`, `-e`	✔	✘	
`nocontrols`	✔	✔	
`no-header-footer`, `-s`	✔	✔	
`no-highlight`	✔	✔	
`none` (subs)	✔	✔	
`none` (table)	✔	✔	Value for frame, grid
`normal`	✔	✔	
`NOTE`	✔	✔	
`note-caption`	✘	✔	
`notitle`	✔	✔	
`noxmlns`	✔	✔	
`numbered`	✔	✔	
`olive`	✔	✔	
`open`, `--`	✔	✔	
`options`	✔	✔	
`opts` (options alias)	TBD	✔	
`%` (options shorthand)	TBD	✔	
`out-file`, `-o`	✔	TBD	
`overline`	✔	✔	
`p`	✘	✔	Can only be assigned to the inline pass macro.
`partintro`	✔	✔	
`++++`	✔	✔	
`pass` (open block, paragraph)	✔	✔	
`pdf`	✔	Pending	
`pgwide`	✔	✘	
`plaintext`	✔	✘	
`post_replacements`	✘	✔	Replaces AsciiDoc.py's `replacements2`.
`postsubs`	✔	✘	This attribute is not necessary in Asciidoctor.
`poster`	✔	✔	
`preamble` (ToC)		✔	
`preface`	✔	✔	

`presubs`	✔	✘	This attribute is not necessary in Asciidoctor.
`prettify`	✘	✔	
`properties`		✔	Where did I get this attr/value from?????
`psv`	✔	✔	
`purple`	✔	✔	
`pygments`	✔	✔	
`pygments-css`	✘	✔	
`pygments-linenums-mode`	✘	✔	
`pygments-style`	✘	✔	
`q`	✘	✔	Can only be assigned to the inline pass macro.
`quanda`	✔	✔	
`quote`, `__`	✔	✔	
`quote` (air quotes)	🗎	✔	
`quote` (Markdown-style)	🗎	✔	
`quote` (quoted paragraph)	🗎	✔	
`quotes` (substitution)	✔	✔	
`r`	✘	✔	Can only be assigned to the inline pass macro.
`red`	✔	✔	
`reftext`	✔	✔	
`related`, `rel`	✘	✔	
`replacements`	✔	✔	
`replacements2`	✔	✘	In Asciidoctor, use `post_replacements`.
`revdate`	✔	✔	
`revnumber`	✔	✔	
`revremark`	✔	✔	
`right` (image)	✔	✔	Value for align, float, role
`right` (table)	✔	✘	Value for align
`right` (table)	✔	✘	Value for halign
`right` (ToC)	✘	✔	
`role`	✔	✔	
`.` (role shorthand)	TBD	✔	
`rotate` (icon)	✘	✔	
`rows` (table)	✔	✔	Value for grid

`--safe`	✔	✔	
`SAFE, 1`	TBD	✔	
`--safe-mode , -S`	TBD	✔	
`safe-mode-<integer or name>`	✖	✔	
`scaled` (image)	✔	✖	
`scaledwidth` (image)	✔	✖	
`scriptsdir`	✔	✔	
`sectanchors`	✖	✔	
`sectids`	✔	✔	
`sectlinks`		✔	// Where did I get this attribute from?
`sectnum, section-numbers, n`	✔	✔	
`sectnumlevels`	TBD	✔	
`SECURE, 20`	TBD	✔	
`separator`	✔	TBD	
`SERVER, 10`	TBD	✔	
`sgml`	✔	✖	
`showcomments`	✔	✖	
`showtitle`	TBD	✔	
`sidebar, ****`	✔	✔	
`sides` (table)	✔	✔	Value for frame
`silver`	✔	✔	
`size` (icon)	✖	✔	
`skip`	✖	✔	
`small`	✔	✔	
`source, ----`	TBD	✔	
`source-highlighter`	✔	✔	
`specialchars, specialcharacters`	✔	✔	
`specialwords`	✔	✖	
`square`	✖	✔	Requires default Asciidoctor stylesheet or custom stylesheet to render properly.
`start`	✔	✔	
`stem`	TBD	✔	

`step`	TBD	✔	
`strong` (labeled list)	✔	✔	
`stylesdir`	✔	✔	
`stylesheet`	✔	✔	
`subs`	✔	✔	
`synopsis`	✔	✔	
`table-caption`	✔	✔	
`tabsize`	✔	✖	
`teal`	✔	✔	
`template`	✔	✔	
`template-dirs`	✔	✔	
`template-engine`	✔	✔	
`theme`	✔	✖	
`thumb`, `th`	✖	✔	
`TIP`	✔	✔	
`tip-caption`	✖	✔	
`title` (icon)	✖	✔	
`title` (image)	✔	✔	
`toc`	✔	✔	
`toc2`	✔	✔	
`toclevels`	✔	✔	
`toc-placement`	✔	✔	
`toc-postion`	✖	✔	
`toc-title`	✔	✔	
`top` (table)	✔	✖	Value for valign
`topbot` (table)	✔	✔	Value for frame
`unbreakable`	✔	✔	
`underline`	✔	✔	
`unfloat` (image)	✔	✔	
`upperalpha`	✔	✔	
`upperroman`	✔	✔	
`[ ]` (unchecked)	✖	✔	
`UNSAFE`, `0`	TBD	✔	

`valign` (table)	✔	✖	
`vbar`	📄	✔	
`verbatim`	✔	✔	Composite value for `subs`
`verse`, `__`	✔	✔	
`vertical` (icon)	✖	✔	
`video::`	✔	✔	
`WARNING`	✔	✔	
`warning-caption`	✖	✔	
`width` (icon)	✖	✔	
`width` (image, video, table)	✔	✔	
`window` (icon)	TBD	✔	
`window` (link)	TBD	✔	
`white`	✔	✔	
`xhtml11`	✔	✔	
`xmlns`	✔	✔	
`yellow`	✔	✔	

Appendix B: Built-in attributes for character replacements

Built-in entity attributes

Attribute reference	Replacement	Rendered
{blank}, {empty}	*nothing*	
{sp}	*single space*	
{nbsp}		
{zwsp}	&#8203;	
{wj}	&#8288;	
{apos}	&#39;	'
{quot}	&#34;	"
{lsquo}	&#8216;	‘
{rsquo}	&#8217;	’
{ldquo}	&#8220;	“
{rdquo}	&#8221;	”
{deg}	&#176;	°
{plus}	&#43;	+
{brvbar}	&#166;	¦

Built-in literal attributes

Attribute reference	Replacement	Rendered
{lt}	<	<
{gt}	>	>
{amp}	&	&
{startsb}	[	[
{endsb}	]	]
{vbar}	|	|
{caret}	^	^
{asterisk}	*	*
{tilde}	~	~
{backslash}	\	\
{backtick}	`	`
{two-colons}	::	::
{two-semicolons}	;;	;;
{cpp}	C++	C++

Last updated 2015-10-09 06:21:33 UTC