AsciiDoc Writer's Guide

Table of Contents

This guide provides a gentle introduction to AsciiDoc, a *plain text* documentation **syntax** and **processor**. This introduction is intended for anyone who wants to reduce the effort required to write and publish content, whether for technical documentation, articles, web pages or good ol'-fashioned prose.

If you want to know what AsciiDoc is all about, find the answer in What is AsciiDoc?. If you're looking for a concise survey of the AsciiDoc syntax, consult the AsciiDoc Syntax Quick Reference.

In this guide, you'll learn:

- The basic structure of an AsciiDoc document
- How to create your first AsciiDoc document
- How to add other structural elements such as lists, block quotes and source code

- How to convert an AsciiDoc document to HTML, DocBook and PDF

In addition to covering the AsciiDoc basics, this guide also suggests a set of conventions to help you create more consistent documents and maximize your writing productivity.

Let's dive in to AsciiDoc!

Writing in AsciiDoc

The goal of this section is to teach you how to compose your first AsciiDoc document. Hopefully, when you look back, you'll agree it just makes sense.

Your adventure with AsciiDoc begins in your favorite text editor.

It's just text, mate.

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) and makes conversion tedious.

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

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

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

Open up your favorite text editor and get ready to write some AsciiDoc!

Content is king!

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

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

Two paragraphs in an AsciiDoc document

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

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

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

Just like that, **you're writing in AsciiDoc!** As you can see, it's just like writing an e-mail.

Save the file with a file extension of `.adoc`.

If you want to find out how to convert the document to HTML, DocBook or PDF, skip ahead to the section on Converting your document.

Wrapped text and hard line breaks

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

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

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

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

Rubies are red,
Topazes are blue.

Line breaks preserved using the hardbreaks option

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

Ruby is red.
Java is black.

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

Line breaks preserved throughout the document using the hardbreaks attribute

```
= Line Break Doc Title
:hardbreaks:

Rubies are red,
Topazes are blue.
```

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

Admonitions

There are certain statements you may want to draw attention to by taking them out of the content's flow and labeling them with a priority. These are called admonitions. It's rendered style is determined by the assigned label (i.e., value). Asciidoctor provides five admonition style labels:

- `NOTE`
- `TIP`
- `IMPORTANT`
- `CAUTION`
- `WARNING`

Caution vs. Warning

When choosing the admonition type, you may find yourself getting confused between "caution" and "warning" as these words are often used interchangeably. Here's a simple rule to help you differentiate the two:

- Use **CAUTION** to advise the reader to *act* carefully (i.e., exercise care).
- Use **WARNING** to inform the reader of danger, harm, or consequences that exist.

To find a deeper analysis, see https://www.differencebetween.com/difference-between-caution-and-vs-warning/.

When you want to call attention to a single paragraph, start the first line of the paragraph with the label you want to use. The label must be uppercase and followed by a colon (`:`).

Admonition paragraph syntax

```
WARNING: Wolpertingers are known to nest in server racks. ① ②
Enter at your own risk.
```

① The label must be uppercase and immediately followed by a colon (`:`).

② Separate the first line of the paragraph from the label by a single space.

Result: Admonition paragraph

Wolpertingers are known to nest in server racks. Enter at your own risk.

An admonition paragraph is rendered in a callout box with the admonition label—or its corresponding icon—in the gutter. Icons are enabled by setting the `icons` attribute on the document.

Admonitions can also encapsulate any block content, which we'll cover later.

All words and no emphasis makes the document monotonous. Let's give our paragraphs some ***emotion***.

Mild punctuation, strong impact

Just as we emphasize certain words and phrases when we speak, we can emphasize them in text by surrounding them with punctuation. AsciiDoc refers to this markup as *quoted text*.

Quoted text

For instance, in an e-mail, you might "speak" a word louder by enclosing it in asterisks.

```
I can't believe it, we *won*!
```

As you would expect, the asterisks make the text **won** bold. You can almost sense the emotion. This is one example of quoted (i.e., formatted) text.

The term "quote" is used liberally here to apply to any symbols that surround text in order to apply emphasis or special meaning.

Here are the forms of quoted text that AsciiDoc recognizes:

Bold, italic, and monospace formatting syntax

```
bold *constrained* & **un**constrained

italic _constrained_ & __un__constrained

bold italic *_constrained_* & **__un__**constrained

monospace `constrained` & ``un``constrained

monospace bold `*constrained*` & ``**un**``constrained

monospace italic `_constrained_` & ``__un__``constrained

monospace bold italic `*_constrained_*` & ``**__un__**``constrained
```

When you want to quote text (e.g., place emphasis) somewhere other than at the boundaries of a word, you need to double up the punctuation.

Result: Bold, italic, and monospace text

bold **constrained** & **un**constrained

italic *constrained* & *un*constrained

bold italic ***constrained*** & ***un***constrained

monospace `constrained` & `un`constrained

monospace bold **`constrained`** & **`un`**constrained

monospace italic *`constrained`* & *`un`*constrained

monospace bold italic ***`constrained`*** & ***`un`***constrained

Any quoted text can be prefixed with an attribute list. The first positional attribute is treated as a role. The role can be used to apply custom styling to the text. For instance:

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

When converting to HTML, the word “asciidoc” is wrapped in `<span>` tags and the role is used as the element’s CSS class:

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

You can apply styles to the text using CSS.

You may not always want these substitutions to take place. In those cases, you’ll need to use markup

to escape the text.

Preventing substitution

If you are getting quoted text behavior where you don't want it, you can use a backslash or a passthrough macro to prevent it.

To prevent punctuation from being interpreted as formatting markup, precede it with a backslash (\). If the formatting punctuation begins with two characters (e.g., __), you need to precede it with two backslashes (\\). This is also how you can prevent character and attribute references from substitution. When your document is processed, the backslash is removed so it doesn't display in your output.

```
\*Stars* will appear as *Stars*, not as bold text.

\&sect; will appear as an entity, not the &sect; symbol.

\\__func__ will appear as __func__, not as emphasized text.

\{two-semicolons} will appear {two-semicolons}, not resolved as ;;.
```

Asciidoctor supports several forms of the passthrough macro.

inline pass macro

An inline macro named `pass` that can be used to passthrough content. Supports an optional set of substitutions.

```
pass:[content like #{variable} passed directly to the output] followed by normal
content.

content with only select substitutions applied: pass:c,a[__<{email}>__]
```

single and double plus

A special syntax for preventing text from being formatted. Only escapes special characters for compliance with the output format and doesn't support explicit substitutions.

triple plus

A special syntax for designating passthrough content. Does not apply any substitutions (equivalent to the inline pass macro) and doesn't support explicit substitutions.

double dollar (deprecated)

A deprecated special syntax for designating passthrough content. Like the triple plus, does not apply any substitutions and doesn't support explicit substitutions.

Asciidoctor does not implement the block pass macro. Instead, you should use a pass block.

Inline pass macro and explicit substitutions

To exclude a phrase from substitutions and disable escaping of special characters, enclose it in the inline pass macro. For example, here's one way to format text as underline when generating HTML from AsciiDoc:

```
The text pass:[<u>underline me</u>] is underlined.
```

The text <u>underline me</u> is underlined.

If you want to enable ad-hoc `quotes` substitution, then assign the `macros` value to `subs` and use the inline pass macro.

```
[subs=+macros] ①
----
I better not contain *bold* or _italic_ text.
pass:quotes[But I should contain *bold* text.] ②
----
```

① `macros` is assigned to `subs`, which allows any macros within the block to be processed.

② The pass macro is assigned the `quotes` value. Text within the square brackets will be formatted.

The inline pass macro does introduce additional markup into the source code that could make it invalid in raw form. However, the output it produces will be valid when viewed in a viewer (HTML, PDF, etc.).

```
I better not contain *bold* or _italic_ text.
But I should contain 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 text pass:q[<u>underline *me*</u>] is underlined and the word "`me`" is bold.
```

The text <u>underline me</u> is underlined and the word “me” is bold.

Triple plus passthrough

The triple-plus passthrough works much the same way as the pass macro. To exclude content from substitutions, enclose it in triple pluses (+++).

```
+++content passed directly to the output+++ followed by normal content.
```

The triple-plus macro is often used to output custom HTML or XML.

```
The text +++<u>underline me</u>+++ is underlined.
```

The text <u>underline me</u> is underlined.

Single plus enclosure

To exclude a phrase from substitutions, enclose it in plus signs (+).

```
This +*literal*+ will appear as *literal*.
```

Replacements

AsciiDoc also recognizes textual representations of symbols, arrows and dashes.

Textual symbol replacements

Name	Syntax	Unicode Replacement	Rendered	Notes
Copyright	(C)	&#169;	©	
Registered	(R)	&#174;	®	
Trademark	(TM)	&#8482;	™	

Name	Syntax	Unicode Replacement	Rendered	Notes
Em dash	`--`	`&#8212;`	—	Only replaced if between two word characters, between a word character and a line boundary, or flanked by spaces. When flanked by space characters (e.g., `a -- b`), the normal spaces are replaced by thin spaces ( ).
Ellipsis	`...`	`&#8230;`	…	
Single right arrow	`->`	`&#8594;`	→	
Double right arrow	`=>`	`&#8658;`	⇒	
Single left arrow	`<-`	`&#8592;`	←	
Double left arrow	`<=`	`&#8656;`	⇐	
Typographic apostrophe	`Sam's`	`Sam&#8217;s`	Sam’s	The typewriter apostrophe is replaced with the typographic (aka curly) apostrophe.

This mild punctuation does not take away from the readability of the text. In fact, you could argue that it makes the text easier to read. What’s important is that these are conventions with which you are likely already familiar.

Punctuation is used in AsciiDoc to create another very common type of element in documents, *lists!*

Lists, lists, lists

There are three types of lists supported in AsciiDoc:

1. Unordered
2. Ordered
3. Description

Unordered and ordered lists are structurally very similar. They consist of items that are prefixed by different types of markers (i.e., bullet). In contrast, description lists—also called variable, labeled, or term-definition lists—are collections of terms that each have their own supporting content. Unlike unordered and ordered lists, description lists are rarely nested, though they often contain the former.

Let’s explore each type of list, then mix them together. We’ll also look at how to put complex content inside a list item.

Lists of things

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
```

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

Rendered unordered list

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

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

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

Rendered unordered list with a title

Kizmet's Favorite Authors

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

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

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

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

Separating Lists

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

```
* Apples
* Oranges

//-

* Walnuts
* Almonds
```

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

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

Rendered nested, unordered list

Possible DefOps manual locations

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

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

Since Asciidoctor 1.5.8, you can nest unordered lists to any depth. Keep in mind, however, that some interfaces will begin flatting lists after a certain depth. GitHub starts flattening list after 10 levels of nesting.

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

- level 1
  - level 2
    - level 3
      - level 4
        - level 5
- 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're shooting for plain text markup that is readable *as is*.

Ordering the things

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

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

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

```
. Protons
. Electrons
. Neutrons
```

1. Protons
2. Electrons
3. Neutrons

If you decide to use number for your ordered list, you have to keep them sequential. This differs

from other lightweight markup languages. It's one way to adjust the numbering offset of a list. For instance, you can type:

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

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

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

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

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

Parts of an atom

3. Protons
2. Electrons
1. Neutrons

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

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

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

Parts of an atom

1. Protons
2. Electrons
3. Neutrons

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

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

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

A nested ordered list

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

The following table shows the numbering scheme used by default for each nesting level.

Ordered list numbering scheme by level

Level	Numbering Scheme	Examples	CSS class (HTML converter)
1	Arabic	1. 2. 3.	arabic
2	Lower Alpha	a. b. c.	loweralpha
3	Lower Roman	i. ii. iii.	lowerroman
4	Upper Alpha	A. B. C.	upperalpha
5	Upper Roman	I. II. III.	upperroman

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
```

Description lists

A description list (often abbreviate as dlist) is useful when you need to include a description or supporting text for one or more terms. Each item in a description list consists of:

- one or more terms
- a separator following each term (typically a double colon, `::`)
- at least one space or endline
- the supporting content (either text, attached blocks, or both)

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

```
CPU:: The brain of the computer.
Hard drive:: Permanent storage for operating system and/or user files.
RAM:: Temporarily stores information the CPU uses during operation.
Keyboard:: Used to enter text or control items on the screen.
Mouse:: Used to point to and select items on your computer screen.
Monitor:: Displays information in visual form using text and graphics.
```

By default, the content of each item is displayed below the description when rendered. Here's a preview of how this list is rendered:

A basic description list

CPU
: The brain of the computer.

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

RAM
: Temporarily stores information the CPU uses during operation.

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

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

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

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

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

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

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

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

Dairy

- Milk
- Eggs

Bakery

- Bread

Produce

- Bananas

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

```
Dairy::

  * Milk
  * Eggs

Bakery::

  * Bread

Produce::

  * Bananas
```

Hybrid lists

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

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

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

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

Here's how the list is rendered:

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.

Complex list content

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

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

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

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

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

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

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

List continuation

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

Here's an example:

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

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

  The header is optional.

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

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

```
* The header in AsciiDoc must start with a document title.
+
----
= Document Title
----
+
Keep in mind that the header is optional.

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

Here's how the source is rendered:

A list with complex content

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

  ```
  = Document Title
  ```

  Keep in mind that the header is optional.

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

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

If you're attaching more than one block to a list item, you're strongly encouraged to wrap the content inside an open block. That way, you only need a single list continuation line to attach the open block to the list item. Within the open block, you write like you normally would, no longer

having to worry about adding list continuations between the blocks to keep them attached to the list item.

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

```
* The header in AsciiDoc must start with a document title.
+
--
Here's an example of a document title:

----
= Document Title
----

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

Here's how that content is rendered:

A list with complex content wrapped in an open block

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

  Here's an example of a document title:

  ```
  = Document Title
  ```

  

  The header is optional.

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

```
* list item
+
--
include::shared-content.adoc[]
--
```

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

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

Dropping the principal text

If the principal text of a list item is blank, the node for the principal text is dropped. This is how you

can get the first block (such as a listing block) to line up with the list marker. You can make the principal text blank by using the `{blank}` attribute reference.

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

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

Here's how the source is rendered:

A list with complex content

1.
```
print("one")
```
2.
```
print("one")
```

Attaching to an ancestor list

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

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

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

+
paragraph attached to parent list item
```

Here's how the source is rendered:

A block attached to the parent list item

- parent list item
  - child list item

  paragraph attached to parent list item

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

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


+
paragraph attached to grandparent list item
```

Here's how the source is rendered:

A block attached to the grandparent list item

- grandparent list item
  - parent list item
    - child list item

  paragraph attached to grandparent list item

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

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

Here's how the source is rendered:

A nested block enclosed in an open block

- grandparent list item
  - parent list item
    - child list item

  paragraph attached to grandparent list item

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

Links and images

AsciiDoc makes it easy to include links, images and other types of media in a document.

External links

There's nothing you have to do to make a link to a URL. Just include the URL in the document and AsciiDoc will turn it into a link.

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

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

You can think of these like implicit macro names. Since the URL in the example below begins with a protocol (in this case *https* followed by a colon), Asciidoctor will automatically turn it into a hyperlink when it is processed.

```
The homepage for the Asciidoctor Project is https://asciidoctor.org. ①
```

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

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

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

```
:hide-uri-scheme:

https://asciidoctor.org
```

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

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

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

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

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

When a URL does not start with one of the common schemes, or the URL is not surrounded by word boundaries, you must use the `link` macro. The `link` macro is a stronger version of a URI macro, which you can think of like an unconstrained macro. The URL is preceded by `link:` and followed by square brackets. The square brackets may include optional link text. The URL is used for the text of the link if link text is not specified. Prior to 1.5.7, if the `linkattrs` document attribute is set, the text in square brackets is parsed as attributes, which allows a window name or role to be specified. Since 1.5.7, attributes are parsed automatically if an equal sign is found after a comma (e.g., `[link text,window=_blank]`).

Anatomy of a link macro

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

Let's consider a case where we need to use the link macro (instead of just a URI macro) to expand a link when it's not adjacent to a word boundary (i.e., unconstrained).

```
search/link:https://ecosia.org[Ecosia]
```

search/Ecosia

If we didn't use the `link:` prefix in this case, the URL macro would not be detected by the parser.

Target window and role attributes for links

Prior to 1.5.7, Asciidoctor *does not* parse attributes in the link macro by default. If you want attributes in the link macro to be parsed, you must set the `linkattrs` document attribute in the header. Since 1.5.7, this parsing is automatic (and the attribute is not required) if an equal sign is found after a comma. When attribute parsing is enabled, you can then specify the name of the

target window using the `window` attribute.

```
= Asciidoctor Document Title

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

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

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

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

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

When attribute parsing is enabled, you can add a role (i.e., CSS class) to the link.

```
Chat with other Asciidoctor users on the https://discuss.asciidoctor.org/[*mailing
list*^,role=green].
```

Chat with other Asciidoctor users on the **mailing list**.

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

Links to relative files

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

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

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

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

For links to relative AsciiDoc documents cross references should be used.

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*).

In Asciidoctor, the inline xref macro is used to create cross references (also called in-text or page citations) to content elements (sections, blocks, or phrases) that have an ID (regardless of whether that ID is explicit or auto-generated).

You create a cross reference by enclosing the ID of the target block or section (or the path of another document with an optional anchor) in double angled brackets.

Cross reference using the ID of the target section

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

Rendered cross reference using the ID of the target section

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

You can also link to a block or section using the title by referencing its title, referred to as a *natural cross reference*. The title must contain at least one space character or contain at least one uppercase letter. (If you are using Ruby < 2.4, that uppercase letter is restricted to the basic Latin charset).

Cross reference using a section's title

```
Refer to <<Internal Cross References>>.
```

Rendered cross reference using a section's title

Refer to Internal Cross References.

Converters usually use the reftext of the target as the default text of the link. When the document is parsed, attribute references in the reftext are substituted immediately. When the reftext is displayed, additional reftext substitutions are applied to the text (specialchars, quotes, and replacements).

You can override the reftext of the target by specifying alternative text at the location of the cross reference. After the ID, add a comma and then enter the custom text you want the cross reference to display.

Cross reference with custom xreflabel text

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

Rendered cross reference using custom xreflabel text

Learn how to use attributes within the link macro.

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

Inline xref macro

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

Cross references can also be used to create a link to a file relative to the current document. For links to another AsciiDoc document, this is the preferred way.

The trailing hash (#) means that you refer to the top of the document.

Cross reference to the top of a relative AsciiDoc document

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

Converted HTML for cross reference to relative AsciiDoc document

```
Refer to <a href="document-b.html">Document B</a> for more information.
```

To link directly to a section in the document, append the section's ID after the hash (#).

Cross reference to a specific section of a relative AsciiDoc document

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

Converted HTML for cross reference to section of a relative AsciiDoc document

```
Refer to <a href="document-b.html#section-b">Section B</a> for more information.
```

In both cases, this syntax will also work if you are inside the document you are referring to. This is useful if you are sharing the same link across multiple documents.

In the link that is created from the inter-document cross reference, the source file extension is replaced with the value of the `outfilesuffix` attribute. To customize the file extension used in the target of the link, simply change the value of this attribute.

Image references are similar to links since they are also references to URLs or files. The difference, of course, is that they display the image in the document.

Images

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

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

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

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

You can also give the image an id, a title (i.e., caption), set its dimensions (i.e., width and height) and make it a link:

```
[#img-sunset]
.A mountain sunset
[link=https://www.flickr.com/photos/javh/5448336655]
image::sunset.jpg[Sunset,300,200]
```

The title of a block image is displayed underneath the image when rendered. Here's a preview:

A hyperlinked image with caption

A mountain sunset

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.

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

```
Press the image:save.png[Save, title="Save"] button.
```

For inline images, the optional title is displayed as a tooltip.

You can also include other types of media, such as audio and video. Consult the block audio and video macros section of the AsciiDoc User Guide for details.

If paragraphs and lists are the meat of the document, then titles and sections are its bones. Let's explore how to give structure to our document.

Titles, titles, titles

AsciiDoc supports three types of titles:

1. Document title
2. Section title
3. Block title

All titles are optional in AsciiDoc. This section will define each title type and explain how and when to use them.

Document title

Just as every e-mail has a subject, every document (typically) has a title. The title goes at the top of an AsciiDoc document.

A document title is an *optional* feature of an AsciiDoc document.

To create a document title, begin the first line of the document with one equal sign followed by at least one space (`= `), then the text of the title. This syntax is the simplest (and thus recommended) way to declare a document title.

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

```
= Lightweight Markup Languages

According to Wikipedia...
```

The document title is part of the document header. So what else can go in the header? Good question.

The document header

Notice the blank line between the title line and the first line of content in the previous example. This blank line separates the document header from the document body (in this case a paragraph). The document title is part of the document header. In all, the document header contains the title, author, revision information and document-wide attributes.

If the title line is not offset by a blank line, it gets interpreted as a section title, which we'll discuss later.

Your document now has a title, but what about an author? Just as every e-mail has a sender, every document must surely have an author. Let's see how to add additional information to the header, including an author.

There are two optional lines of text you can add immediately below the document title for defining common document attributes:

Line 1

Author name and an optional e-mail address

Line 2

An optional revision, a date and an optional remark

Let's add these lines to our document:

```
= Lightweight Markup Languages
Doc Writer <doc.writer@asciidoc.org>
v1.0, 2012-01-01

According to Wikipedia...
```

The header now contains a document title, an author, a revision number, and a date. This information will typically be displayed as a formatted header at the top of the output document.

The header, including the document title, is *not required*. If absent, the AsciiDoc processor will happily convert whatever content is present. The header is only used when generating a full document. It's excluded from the output of an embedded document.

The document header can also be used to define attributes.

Document attributes

Attributes are one of the features that sets AsciiDoc apart from other lightweight markup languages. You can use attributes to toggle features or to store reusable or replacement content.

Most often, attributes are defined in the document header. There are scenarios where they can be defined inline, but we'll focus on the more common usage.

An attribute entry consists of a name surrounded by colons at the beginning of the line followed by at least one space, then the content. The content is optional.

Here's an example of an attribute that holds the version of an application:

```
= User Guide
Doc Writer <doc.writer@asciidoc.org>
2012-01-01
:appversion: 1.0.0
```

There should be no blank lines between the first attribute entry and the rest of the header.

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

```
The current version of the application is {appversion}.
```

Attributes are also commonly used to store URLs, which can get quite lengthy. Here's an example:

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

Here's the attribute in use:

```
Information about the Asciidoctor package for Fedora can found at {fedpkg}.
```

Document attributes can also be used to toggle settings or 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 define the `toc` attribute:

```
:toc:
```

To undefine an attribute, place a `!` at the end of the name:

```
:linkcss!:
```

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
```

Attribute values can also be set and overridden when invoking the AsciiDoc processor. We'll explore that feature later.

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.

As your document grows, you'll want to break the content into sections, like in this guide. That's accomplished using section titles.

Section titles

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

A section title uses the same syntax as a document title, except the line may begin with two to six equal signs instead of just a single equal sign. The number of equal signs represents the nesting level (using a 0-based index).

Here are all the section levels permitted in an AsciiDoc document (for an article doctype, the default), shown below the document title:

```
= Document Title (Level 0)

== Level 1 Section

=== Level 2 Section

==== Level 3 Section

===== Level 4 Section

====== Level 5 Section

== Another Level 1 Section
```

When the document is converted to HTML 5 (using the built-in `html5` backend), each section title becomes a heading element where the heading level matches the number of equal signs. For example, a level 1 section (2 equal signs) maps to an `<h2>` element.

Section levels cannot be chosen arbitrarily. There are two rules you must follow:

1. A document can only have multiple level 0 sections if the `doctype` is set to `book`.[1]
2. Section levels cannot be skipped when nesting sections

For example, the following syntax is illegal:

```
= Document Title

= Illegal Level 0 Section (violates rule #1)

== First Section

==== Illegal Nested Section (violates rule #2)
```

Content above the first section (after the document title) is part of the preamble. Once the first section is reached, content is associated with the section that precedes it:

```
== First Section

Content of first section

=== Nested Section

Content of nested section

== Second Section

Content of second section
```

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

To have the processor auto-number the sections, define the `sectnums` attribute in the document header:

```
:sectnums:
```

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
```

Preamble

Content between the document title and the first section is called the preamble. If a document title is not present, this content is not wrapped in a preamble section.

```
= Document Title

preamble

another preamble paragraph

== First Section
```

When using the default Asciidoctor stylesheet, this preamble is rendered in the style of a lead (i.e., larger font).

You can also assign titles to individual elements.

Block titles

You can assign a title to any paragraph, list or delimited block element. The title is used as the element's caption. 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 with no spaces in between.

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

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

Speaking of block titles, let's dig into blocks and discover which types of blocks AsciiDoc supports.

Building blocks in AsciiDoc

AsciiDoc provides a nice 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.

Delimited blocks

You've already seen many examples of the listing block (i.e., code block), which is surrounded by lines with four or more hyphens.

```
----
This is an example of a _listing block_.
The content inside is displayed as <pre> text.
----
```

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

Here's how the block above appears when converted to HTML and viewed in a browser:

```
This is an example of a _listing block_.
The content inside is displayed as <pre> text.
```

Here's the HTML source that is generated:

```
<div class="listingblock">
  <div class="content monospaced">
    <pre>This is an example of a _listing block_.
The content inside is displayed as &lt;pre&gt; text.</pre>
  </div>
</div>
```

You should notice a few things about how the content is processed:

- the HTML tag `<pre>` is escaped
- the endlines are preserved
- the phrase "listing block" is not italicized, despite having underscores around it.

Each type of block is processed according to its purpose. Literal blocks don't receive the full set of substitutions normally applied to a paragraph. Since a listing block is typically used for source code, substitutions are not desirable.

The following table identifies the delimited blocks that AsciiDoc provides by default, their purpose and what substitutions are performed on its content.

Name (Style)	Line delimiter	Purpose	Substitutions
comment	////	Private notes that are not displayed in the output	none
example	====	Designates example content or defines an admonition block	normal

Name (Style)	Line delimiter	Purpose	Substitutions
literal	`....`	Output text to be displayed exactly as entered	verbatim
listing, source	`----`	Source code or keyboard input to be displayed as entered	verbatim
open	`--`	Anonymous block that can act as any other block (except *pass* or *table*)	varies
pass	`++++`	Raw text to be passed through unprocessed	none
quote, verse	`____`	A quotation or verse with optional attribution	normal
sidebar	`****`	Aside text displayed outside the flow of the document	normal
table	`|===`	Used to display tabular content or advanced layouts	varies

AsciiDoc allows delimited lines to be longer than 4 characters. **Don't do it.** Maintaining long delimiter lines is a *colossal* waste of time, not to mention arbitrary and error prone. Use the minimum line length required to create a delimited block and *move on* to drafting the content. The reader will never see the long delimiters anyway since they are not carried over to the output.

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

Group / Substitution	Normal	Verbatim	None
Special chars	Yes	Yes	No
Callouts	No	Yes	No
Quotes	Yes	No	No
Attributes	Yes	No	No
Replacements	Yes	No	No
Macros	Yes	No	No
Post replacements	Yes	No	No

In order to apply normal substitutions to an attribute value, surround it with single quotes. There

are two exceptions to this behavior: At the moment normal substitutions are not applied to the `options` and `title` attribute values.

You can control how blocks are displayed using block metadata.

Block metadata

Metadata can be assigned to any block. There are several types of metadata:

- Title
- Id (i.e., anchor)
- Style (first unnamed block attribute)
- Named block attributes

Here's an example of a quote block that includes all types of metadata:

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

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

Here's the metadata extracted from this block:

Title

Gettysburg Address

Id

gettysburg

Style

quote

Named block attributes

attribution

Abraham Lincoln

citetitle

Address delivered at the dedication of the Cemetery at Gettysburg

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, such as the style, controls how the block is converted.

Masquerading blocks

Some blocks can masquerade as other blocks, a feature which is controlled by the block style. The block style is the first positional attribute in the block attribute list.

Admonition blocks

For instance, an example block can act as an admonition block:

```
[NOTE]
====
This is an example of an admonition block.

Unlike an admonition paragraph, it may contain any AsciiDoc content.
The style can be any one of the admonition labels:

* NOTE
* TIP
* WARNING
* CAUTION
* IMPORTANT
====
```

Listing and source code blocks

At the start of this tutorial, remember how painful we said it is to insert source code into a document using a traditional word processor. They just aren't designed for that use case. **AsciiDoc is!**

In fact, inserting source code in an AsciiDoc is incredibly easy. Just shove the raw code into a listing block.

```
----
require 'asciidoctor'

puts Asciidoctor.convert_file 'mysample.adoc', to_file: false
----
```

To enable syntax highlighting in the output, set the style on the block to `source` and specify the source language in the second attribute position.

```
[source,ruby]
----
require 'asciidoctor'

puts Asciidoctor.convert_file 'mysample.adoc', to_file: false
----
```

You can even use source code that's in a separate file. Just use the AsciiDoc include directive:

```
[source,ruby]
----
include::example.rb[]
----
```

To really show how well-suited AsciiDoc is for technical documentation, it also supports callouts in source code. Code callouts are used to explain lines of source code. The explanations are specified below the listing and keyed by number. Here's an example:

```
[source,ruby]
----
require 'asciidoctor'  # <1>

Asciidoctor.convert_file 'mysample.adoc'  # <2>
----
<1> Imports the library
<2> Reads, parses, and converts the file
```

Here's how the callouts appear when rendered:

Source code with callouts

```
require 'asciidoctor'  ①

puts Asciidoctor.convert_file 'mysample.adoc'  ②
```

① Imports the library

② Reads, parses, and converts the file

Open blocks

The most versatile block of all is the open 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:

```
[sidebar]
.Related information
--
This is aside text.

It is used to present information related to the main content.
--
```

Passthrough blocks

The “anything goes” mechanism in AsciiDoc is the passthrough block. As the name implies, this block passes the content of the block directly through to the output document. When you’ve encountered a complex requirement that you cannot meet using the AsciiDoc syntax, a passthrough block can come in very handy.

For example, let’s say you want to embed a GitHub gist into your document. You can define the following passthrough block:

```
++++
<script src="https://gist.github.com/piscisaureus/3342247.js"></script>
++++
```

Using a passthrough block couples your content to a specific output format, such as HTML. If you’re going to use a passthrough block, we recommend using conditional preprocessor directives to associate the format-specific content with each backend you intend to support.

Delimiters optional

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

This format is often used for single-line listings:

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

or single-line quotes:

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

While most blocks are linear, tables give you the ability to layout content horizontally as well.

A new perspective on 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. I recommend that you use tables sparingly because they interrupt the conversation with your readers. When they are the most suitable way to present the information, know that you've got a powerful tool in your hands.

You can think of a table as a delimited block that contains one or more bulleted lists. The list marker is a vertical bar (`|`). Each list represents one row in the table and must share the same number of items (taking into account any column or row spans).

Here's a simple example of a table with two columns and three rows:

```
[cols=2*]
|===
|Firefox
|Web Browser

|Ruby
|Programming Language

|TorqueBox
|Application Server
|===
```

The first non-blank line inside the block delimiter (`|===`) determines the number of columns. Since we are putting each column title on a separate line, we have to use the `cols` block attribute to explicitly state that this table has two columns. The `*` is the repeat operator. It means to repeat the column specification for the remainder of columns. In this case, it means to repeat no special formatting (since none is present) across 2 columns.

We can make the first row of the table the header by setting the `header` option on the table.

```
[cols=2*,options=header]
|===
|Name
|Group

|Firefox
|Web Browser

|Ruby
|Programming Language

...
|===
```

You can also define the `header` option using the following shorthand:

```
[%header,cols=2*]
```

Alternatively, we could define the header row on a single line offset from the body rows by a blank line so neither the `cols` or the `options` attributes are required.

```
|===
|Name |Group

|Firefox
|Web Browser

...
|===
```

The content of each item (i.e., cell) can span multiple lines, as is the case with other lists in AsciiDoc. Unlike other lists, the content of each cell may contain blank lines without the need for a list continuation to hold them together. A new cell begins when another non-escaped vertical bar (`|`) is encountered.

```
|===
|Name |Group |Description

|Firefox
|Web Browser
|Mozilla Firefox is an open-source web browser.
It's designed for standards compliance,
performance, portability.

|Ruby
|Programming Language
|A programmer's best friend.

...
|===
```

You can set the relative widths of each column using *column specifiers*—a comma-separated list of relative values defined in the `cols` block attribute. The number of entries in the list determines the number of columns.

```
[cols="2,3,5"]
|===
|Name |Group |Description

|Firefox
|Web Browser
|Mozilla Firefox is an open-source web browser.
It's designed for standards compliance,
performance and portability.

|Ruby
|Programming Language
|A programmer's best friend.

...
|===
```

If you want to include blocks or lists inside the contents of a column, you can put an `a` (for AsciiDoc) at the end of the column's relative value.

```
[cols="2,3,5a"]
|===
|Name |Group |Description

|Firefox
|Web Browser
|Mozilla Firefox is an open-source web browser.
It's designed for:

* standards compliance,
* performance and
* portability.

|Ruby
|Programming Language
|A programmer's best friend.

...
|===
```

Alternatively, you can apply the AsciiDoc style to an individual cell by prefixing the vertical bar with an `a`:

```
a|Mozilla Firefox is an open-source web browser.
It's designed for:

* standards compliance,
* performance and
* portability.
```

There's a whole collection of column and cell specifiers you can use to format the contents of the table, including styling and alignment. Consult the Tables chapter of the AsciiDoc User Guide for a full list.

AsciiDoc tables can also be created directly from CSV data. Just set the `format` block attribute to `csv` and insert CSV data inside the block delimiters, either directly:

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

or using an `include::[]` directive:

```
[%header,format=csv]
|===
include::tracks.csv[]
|===
```

Asciidoctor 0.1.3 also recognizes shorthand notation for setting CSV and DSV table formats. The first position of the table block delimiter (i.e., `|===`) can be replaced by a data delimiter to set the table format accordingly.

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

```
,===
a,b,c
,===
```

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

```
:===
a:b:c
:===
```

That's a pretty powerful option.

What else can AsciiDoc do?

We've covered many of the features of the AsciiDoc syntax, but it still has much more depth. AsciiDoc is simple enough for a README, yet can scale to meet the requirements of a publisher.

Here are some of the features that the AsciiDoc syntax supports:

- footnotes
- indexes
- appendix, preface, dedication, partintro
- multi-line attributes
- preprocessor directive (conditional markup)
- mathematical formulas
- musical notation
- diagrams
- block filters
- themes
- custom blocks, macros and output formats

Consult the Asciidoctor User Manual to continue exploring the syntax and processor capabilities.

That's enough syntax for now. You've created your first AsciiDoc document. Now it's time to convert the document into a presentable format. This will give you a real appreciation for the power that AsciiDoc puts in your hands.

Converting your document

While 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 processor comes in (**after** you have done the writing).

The Asciidoctor processor parses the document and translates it into a backend format, such as HTML, ePub, DocBook or PDF. Asciidoctor ships with a set of default templates in the tin, but you can customize the templates or create your own to get exactly the output you want.

Before you can use the Asciidoctor processor, you have to install the Asciidoctor Ruby Gem. Review the Asciidoctor Installation Guide if you need helping installing the gem.

Converting a document to HTML 5

Asciidoctor provides both a command line tool and a Ruby API for converting AsciiDoc documents to HTML 5, Docbook 5.0 and custom output formats.

To use Asciidoctor to generate an HTML document, type `asciidoctor` followed by your document's name on the command line.

```
$ asciidoctor mysample.adoc
```

In Asciidoctor, the **html5** backend is the default, so there's no need to specify a backend explicitly to generate an HTML 5 document.

Asciidoctor also provides a Ruby API, so you can generate an HTML document directly from a Ruby application:

```
require 'asciidoctor'

Asciidoctor.convert_file 'mysample.adoc'
```

Alternatively, you can capture the HTML output into a variable instead of writing it to a file:

```
html = Asciidoctor.convert_file 'mysample.adoc', to_file: false, header_footer: true
puts html
```

To generate DocBook, just specify the backend option:

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

One of the strengths of Asciidoctor is that it can output to a variety of formats, not just HTML.

Converting a document to DocBook

Despite the fact that writing in DocBook is inhumane, it's useful as a portable document format. Since 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.

Asciidoctor provides a Docbook 5.0 backend out of the box. To convert the document to Docbook 5.0, call the processor with the backend flag set to `docbook5`:

```
$ asciidoctor -b docbook5 mysample.adoc
```

A new XML document, named `mysample.xml`, will now be present in the current directory:

```
$ ls -1
mysample.adoc
mysample.html
mysample.xml
```

If you're on Linux, you can view the DocBook file using Yelp:

```
$ yelp mysample.xml
```

DocBook is only an intermediary format in the Asciidoctor toolchain. You'll either feed it into a system that processes DocBook (like publican), or you can convert it to PDF using the asciidoctor-fopub tool.

Output galore

There's really no end to the customization you can do to the output the Asciidoctor processor generates. We've just scratched the surface here.

Check out the Asciidoctor User Manual and the Asciidoctor Docs Page to learn more.

Where else is AsciiDoc supported?

The easiest way to experiment with AsciiDoc is online. AsciiDoc document in a GitHub repository or a gist is automatically converted to HTML and rendered in the web interface.

If you have a project on GitHub, you can write the README or any other documentation in AsciiDoc and the GitHub interface will show the HTML output for visitors to view.

Gists, in particular, are a great way to experiment with AsciiDoc. Just create a new gist, name the file with the extension `.adoc` and enter AsciiDoc markup. You can save the document as public or secret. If you want to try AsciiDoc without installing any software, a gist is a great way to get started.

While there's plenty more of the AsciiDoc syntax and toolchain to explore, you know more than enough about it to write a range of documentation, from a simple README to a comprehensive user guide.

Wrap-up

Writing in AsciiDoc should be no more complex than writing an e-mail. All you need to compose a document in AsciiDoc is open a text editor and type regular paragraphs. Only when you need additional semantics or formatting do you need to introduce markup. Let your instinct guide you when you need to remember what punctuation to use. The AsciiDoc syntax is based on time-tested plain-text conventions from the last several decades of computing. Hopefully you agree that the markup does not detract from the readability of the text in raw form, as that's a key goal of lightweight markup languages like AsciiDoc.

As humans, communication is what connects us through the ages and allows us to pass on knowledge. AsciiDoc enables you to focus on communicating rather than distracting you with other stuff that just gets in the way. Copy the text of an e-mail into a document and see how easy it to repurpose it as documentation. Almost immediately, you'll find your writing zen and enjoy the rewarding experience of producing.

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

[1] The default doctype is `article`, which only allows one level 0 section (i.e., the document title).