

AsciiDoc Writer's Guide

Table of Contents

.....	vii
1. Writing in AsciiDoc	1
1.1. "It's just text, mate."	1
1.2. Content is king!	1
1.3. Admonitions	3
1.4. Mild punctuation, strong impact	3
1.4.1. Quoted text	3
1.4.2. Preventing substitution	5
1.4.3. Replacements	6
1.5. Lists, lists, lists	7
1.5.1. Lists of things	7
1.5.2. Ordering the things	9
1.5.3. Titling a list	11
1.5.4. Labeled lists	11
1.5.5. Hybrid lists	13
1.5.6. Complex list content	15
1.5.7. Dividing lists	17
1.6. Links and images	17
1.6.1. External links	17
1.6.2. Target window and role attributes for links	18
1.6.3. Links to relative files	19
1.6.4. Internal cross references	19
1.6.5. Images	19
1.7. Titles, titles, titles	21
1.7.1. Document title	21
1.7.2. Document attributes	23
1.7.3. Section titles	24
1.7.4. Block titles	27
2. Building blocks in AsciiDoc	28
2.1. Delimited blocks	28
2.2. Block metadata	30
2.3. Masquerading blocks	31
2.3.1. Admonition blocks	31
2.3.2. Listing and source code blocks	32
2.3.3. Open blocks	33
2.3.4. Passthrough blocks	34

2.4. Delimiters optional	34
2.5. A new perspective on tables	35
3. What else can AsciiDoc do?	40
4. Rendering your document	41
4.1. Installing AsciiDoc	41
4.1.1. Installing AsciiDoc via a package manager	41
4.1.2. Installing AsciiDoc manually	42
4.2. Converting a document to HTML 5	43
4.3. Converting a document to DocBook	46
4.4. Converting the document to PDF	47
4.5. Producing HTML-based presentations	47
4.6. Output galore	48
4.7. Where else is AsciiDoc rendered?	48
5. Wrap-up	49
Glossary	50

List of Figures

1.1. A mountain sunset 20

List of Tables

1.1. Ordered list numbering scheme by level 10

List of Examples

1.1. A nested ordered list	10
1.2. A basic labeled list	12
1.3. A hybrid list	14
1.4. A list with complex content	16
1.5. A hyperlinked image with caption	20
2.1. Source code with callouts	33

This guide provides a gentle introduction to [AsciiDoc](#)¹, 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?](#)²

If you're looking for a concise survey of the AsciiDoc syntax, consult the [AsciiDoc Syntax Quick Reference](#)³.

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 render 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!

¹ <http://asciidoc.org>

² <http://asciidoctor.org/docs/what-is-asciidoc-why-use-it>

³ <http://asciidoctor.org/docs/asciidoc-syntax-quick-reference/>

Chapter 1. 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.

1.1. “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 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](#)¹.



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](#)² tutorial.

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

1.2. Content is king!

The bulk of the content in an AsciiDoc document is paragraph text. In fact, all it takes to create a document in AsciiDoc is a single sentence. No special markup is required to write a normal paragraph. Just type away!

¹ http://asciidoc.org/#_editor_support

² <http://asciidoc.org/docs/editing-asciidoc-with-live-preview/>

In AsciiDoc, the main structural element is the paragraph. A paragraph consists of adjacent lines of text.

To start a new paragraph, leave a blank line by hitting Enter twice, then just keep on typing.

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 `.ad` or `.adoc`.



Although the file extension doesn't matter to the AsciiDoc processor, it's used in other contexts (such as by your editor or GitHub) to indicate the format of the document.



If you want to find out how to render the document to HTML, DocBook or PDF, skip ahead to the section on [Chapter 4, Rendering your document](#).

Wrapped text and hard line breaks

Adjacent lines like these are combined as a single paragraph. That means you can wrap paragraph text or put each sentence on a separate line and the line breaks won't appear in the output.

Here's how the previous lines look when rendered:

Adjacent lines like these are combined as a single paragraph. That means you can wrap paragraph text or put each sentence on a separate line and the line breaks won't appear in the output.

If you want line breaks in a paragraph to be preserved, simply add a space followed by a plus sign at the end of the sentence:

Roses are red, +
Violets are blue.

Alternatively, if you're using AsciiDoctor and you want line breaks in a paragraph to be preserved, add the following attribute entry to the header of your document below the title, author and revision lines:

```
:hardbreaks:
```

This feature is only available in AsciiDoctor.

1.3. Admonitions

There are certain statements that you might want to draw attention to by taking them out of the flow and labeling them with a priority. These are called *admonition paragraphs*. To make a paragraph an admonition, prefix it with an uppercase label, such as in this note:

```
NOTE: Admonition paragraphs call attention to special words of advice.
```

The following labels are recognized:

- NOTE
- TIP
- WARNING
- CAUTION
- IMPORTANT

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**.

1.4. 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*.

1.4.1. 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

One asterisk (`*`) on either side of a word or phrase makes it bold.

'Italic'

Single quotes around a word or phrase makes it italic.

Italic

One underscore (`_`) on either side of a word or phrase also makes it italic.

Bold italic

Bold markup around a word or phrase in italic makes it bold italic. *Reversing the order of the markup won't produce the same result.*

+Monospace+

One plus (`+`) on either side of a word or phrase makes it monospaced (i.e., constant width).

+*Monospace bold*+

Monospace markup around a word or phrase in bold makes it monospace bold.

+_Monospace italic_+

Monospace markup around a word or phrase in italic makes it monospace italic. *Reversing the order of the markup won't produce the same result.*

#Open style#

One hash (`#`) on either side of a word or phrase makes it possible to assign it a role (i.e., CSS class).

^Superscript^

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

~Subscript~

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

Single `smart quotes`

One leading backtick (```) and one trailing single quote (`'`) around a word or phrase encloses it in single ‘smart quotes’.

Double ``smart quotes''

Two leading backticks (``) and two trailing single quotes (') around a word or phrase encloses it in double "smart quotes".

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. For instance, to emphasis the first letter of a word, you need to surround it in double asterisks:

```
**F**our score and seven years ago...
```



The double punctuation applies for all types of quoted text except smart quotes, subscript and superscript.

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 rendering to HTML, the word "asciidoc" is wrapped in `` 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.

1.4.2. Preventing substitution

If you are getting quoted text behavior where you don't want it, there are several approaches you can use to prevent it.

Backslash escaping

To prevent punctuation from being interpreted, precede it with a backslash:

```
\*Stars* will not be bold, but rather appear as *Stars*.  
The backslash character is automatically removed.
```

Double dollar enclosure

To exclude a phrase from substitutions, enclose it in double dollars (`$$`):

`$$*Stars*$$` will not be bold, but rather appear as `*Stars*`.
 Special characters are still escaped so, `$$<p>$$` appears as `<p>`.
 Double dollar is commonly used to wrap URLs containing punctuation.

Triple plus enclosure & inline pass macro

To exclude a phrase from substitutions and disable escaping of special characters, enclose it in triple pluses (`+++`) or the inline `pass: []` macro:

The markup `+++<u>underline me</u>+++` renders as underlined text.
 The markup `pass:[<u>underline me</u>]` produces the same result.
 Triple plus and `pass:[]` are often used to output custom HTML or XML.

Backticks enclosure

To exclude a phrase from substitutions, disable escaping of special characters and render it as monospaced text, enclose it in backticks (```):

This ``*literal*`` will appear as `*literal*` in a monospace font.
 Backticks are commonly used around inline code containing markup.

1.4.3. Replacements

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

Name	AsciiDoc Source	As Rendered
copyright	(C)	©
registered trademark	(R)	®
trademark	(TM)	™
em dash (between words)	--	—
ellipses	...	…
arrows	-> => <- <=	→ ⇒ ← ⇐
apostrophe	Sam's	Sam’s
XML entity (e.g., dagger)	†	†

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!*

1.5. Lists, lists, lists

There are three types of lists supported in AsciiDoc:

1. *Unordered*
2. `Ordered`
3. **Labeled**

Unordered and `ordered` lists are structurally very similar. You can think of them as outline lists that use different types of markers (i.e., bullets). In contrast, **labeled** lists—also called variable or term-definition lists—are a collection of labels that each have supporting content and they are rarely nested.

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.

1.5.1. Lists of things

If you were to create a list of items in an e-mail, how would you do it? Chances are, what you'd type is exactly how you define an outline list in AsciiDoc.

Here's an example of a grocery list written as an unordered list in AsciiDoc:

-
- * Milk
 - * Eggs
 - * Bread
 - * Bananas
-

Was your instinct to use a hyphen instead of an asterisk? Guess what? That works too:

-
- Milk
 - Eggs
 - Bread
 - Bananas
-

In either case, you don't need to see the output. You already know how it will look ;)



You are free to indent the list however it suits you.



The item's first line of text must be offset from the marker by at least one space.

What if you wanted to group the grocery list by aisle? Then you might organize it as a nested list.

To get a nested item, just add another asterisk in front of the item:

-
- * Diary
 - ** Milk
 - ** Eggs
 - * Bakery
 - ** Bread
 - * Produce
 - ** Bananas
-

You can have up to five levels of nesting:

-
- * Kingdom
 - ** Phylum
 - *** Class
 - **** Order
 - ***** Family
-

The hyphen doesn't work for nested lists since repeating hyphens are used for other purposes in AsciiDoc.



Since a hyphen only works for a single level nesting in an AsciiDoc list, I recommend reserving the hyphen for lists that only have a single level:

List without nested items

- Fedora
 - Ubuntu
 - Slackware
-

For lists that have more than one level, use asterisks:

List with nested items

- * Linux

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

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 this convention:

of asterisks = level of nesting

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

Using a different marker, we can create an ordered list in the same way.

1.5.2. 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
-

Since the numbering is obvious, the AsciiDoc processor will insert the numbers for you if you omit them:

-
- . Protons
 - . Electrons
 - . Neutrons
-

Like with unordered lists, you create a nested item by using one or more dots in front of each the item:

-
- . Lists
 - .. Outline
 - ... Unordered
 - ... Ordered
 - .. Labeled
-

- . Titles
- .. Document
- .. Section
- .. Block



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*.

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

Example 1.1. A nested ordered list

1. Lists
 - a. Outline
 - i. Unordered
 - ii. Ordered
 - b. Labeled
2. Titles
 - a. Document
 - b. Section
 - c. Block

The following table shows the number scheme used by default for each nesting level:

Table 1.1. Ordered list numbering scheme by level

Level	Numbering Scheme	Examples	CSS class (HTML backend)
1	Arabic	1. 2. 3.	arabic
2	Lower Alpha	a. b. c.	loweralpha
3	Lower Roman	i. ii. iii.	lowerroman

Level	Numbering Scheme	Examples	CSS class (HTML backend)
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
```

1.5.3. Titling a list

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

Here are examples of two lists with titles:

```
.Shopping list
* Milk
* Eggs
* Bread

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

Not all lists have punctuation markers. Let's look at lists that use terms to tag each item.

1.5.4. Labeled lists

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 newline
- the item's content

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

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

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

Example 1.2. 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.

`[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.
```

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.

```
Diary::
* 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:

```
Diary::

    * Milk
    * Eggs

Bakery::

    * Bread

Produce::

    * Bananas
```

1.5.5. Hybrid lists

You can mix and match the three list types within a single hybrid list. AsciiDoc 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
```

Again, 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
-

Here's a list that mixes all three types of lists:

-
- Operating Systems::
- . Linux
 - * Fedora
 - * Ubuntu
 - * Slackware
 - . BSD
 - * FreeBSD
 - * NetBSD
- Cloud Providers::
- . PaaS
 - * OpenShift
 - * CloudBees
 - . IaaS
 - * Amazon EC2
 - * Rackspace
-

Here's how the list is rendered:

Example 1.3. A hybrid list

Operating Systems

1. Linux
 - Fedora
 - Ubuntu
 - Slackware

2. BSD

- FreeBSD
- NetBSD

Cloud Providers

1. PaaS

- OpenShift
- CloudBees

2. IaaS

- Amazon EC2
- Rackspace

You can include more complex content in a list item as well.

1.5.6. 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.



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.
```

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
----
```

Here’s how the source is rendered:

Example 1.4. 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
```

1.5.7. Dividing 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:

```
* Apples
* Oranges
* Bananas
```

```
//^
```

```
* Walnuts
* Almonds
* Cashews
```

1.6. Links and images

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

1.6.1. 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.

```
You can learn more about AsciiDoc at http://asciidoc.org.
```

The trailing period will not get caught up in the link. AsciiDoc is smart like that.



AsciiDoc recognizes URLs that begin with `http://`, `https://`, `ftp://`, and `irc://`.

To turn a word or phrase into a link, just enclose it in square brackets at the end of the URL:

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

1.6.2. Target window and role attributes for links

You often need to set the target attribute on a link element (`<a>`) so the link opens in a new window (e.g., ``).

This type of configuration is normally specified using attributes. However, AsciiDoc does not parse attributes in the link macro by default. In AsciiDoctor 0.1.3, you can enable parsing of link macro attributes by setting the `linkattrs` document attribute in the header.

```
:linkattrs:
```

You can also specify the name of the target window using the `window` attribute:

```
http://google.com[Google, window="_blank"]
```

AsciiDoctor includes shorthand for `_blank`, since it is the most common window name. Just end the link text with a caret (`^`):

```
http://google.com[Google^]
```

Since AsciiDoctor is parsing the attributes, that opens the door for adding a role (i.e., CSS class) to the link:

```
http://google.com[Google^, role="external"]
```

You can now have fun styling your links.

1.6.3. Links to relative files

If you want to link to a file relative to the current document, use the `link:` prefix in front of the file name:

```
link:editing-asciidoc-with-live-preview[Editing with Live Preview]
```

To link directly to a section in the document (a “deep” link), append a hash (`#`) followed by the id of the section to the end of the file name:

```
link:editing-asciidoc-with-live-preview/#livereload[LiveReload]
```

You can also create links that refer to sections within the current document.

1.6.4. Internal cross references

A link to another location in the current document is called a *cross reference*. You create a cross reference by enclosing the element’s id in double angled brackets:

```
The section <<content-is-king>> covers paragraphs in AsciiDoc.
```

In some backends, the text of the link will be automatically generated from the title of the element, if one exists. If you want to customize the linked text, include it after the id, separated by a comma:

```
Learn how to create <<content-is-king,paragraphs>> in AsciiDoc.
```

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.

1.6.5. Images

To include an image on it’s 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  
image::sunset.jpg[Sunset, 300, 200, link="http://www.flickr.com/photos/  
javh/5448336655"]
```

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

Example 1.5. A hyperlinked image with caption



Figure 1.1. 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 [Audio and video block macros](#)³ 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.

1.7. 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.

1.7.1. 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
```

³ <http://asciidoc.org/userguide.html#X98>

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 option 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 be formatted when the document is rendered.



The header, including the document title, is *not required*. If absent, the AsciiDoc processor will happily render whatever content is

present. The header is only used when rendering a full document. It's excluded from the output of an embedded document.

The document header can also be used to define attributes.

1.7.2. 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/asciidoc
```

Here's the attribute in use:

```
Information about the AsciiDoc package in Fedora is 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 on the left side the document, you can define the `toc2` attribute:

```
.....  
:toc2:  
.....
```

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  
.....
```

For a complete list of which attributes can be assigned to control the output, consult the [Backend Attributes](#)⁴ chapter of the AsciiDoc User Guide. To see which intrinsic attributes are available, consult the [Intrinsic Attributes](#)⁵ chapter.



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.

1.7.3. Section titles

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

⁴ <http://asciidoc.org/userguide.html#X88>

⁵ <http://asciidoc.org/userguide.html#X60>

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 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 (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`.⁶
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

⁶The default doctype is `article`, which only allows one level 0 section (i.e., the document title).

==== 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 proceeds 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 render just fine as an AsciiDoc document.

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

:numbered:

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:

:numbered!:

== 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.

1.7.4. 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.

Chapter 2. 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.

2.1. 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 rendered 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 rendered and what elements are used to wrap the content in the output.

Here's how the block above appears when rendered as HTML:

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

Here's the HTML that gets generated:

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

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.

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
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 <i>pass</i> or <i>table</i>)	Varies
pass	++++	Raw text to be passed through unprocessed	None
quote, verse	_____	A quotation or verse with optional attribution	Normal
sidebar	****	Aside text rendered outside the flow of the document	Normal

Name (Style)	Line delimiter	Purpose	Substitutions
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	Special characters	Callouts	Quotes	Attributes	Replacements	Macros	Post replacements
Normal	Yes	No	Yes	Yes	Yes	Yes	Yes
Verbatim	Yes	Yes	No	No	No	No	No
None	No	No	No	No	No	No	No

You can control how blocks are displayed using block metadata.

2.2. Block metadata

Metadata can be assigned to any blocks. 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, Soldiers' National Cemetery Dedication]
```

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

Dedication of the Soldiers' National Cemetery



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 style. Consult the [Delimited Blocks¹](#) chapter in the AsciiDoc User Guide for a full list of the metadata that is applicable for each block.

2.3. 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.

2.3.1. Admonition blocks

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

¹ <http://asciidoc.org/userguide.html#X104>

[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

====

2.3.2. 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.render_file('sample.adoc', :header_footer => true)
----
```

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.render_file('sample.adoc', :header_footer => true)
----
```

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

```
[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>

puts Asciidoctor.render_file('sample.adoc', :header_footer => true) <2>
----
\<1> Imports the library
\<2> Reads, parses and renders the file
----
```

Here's how the callouts appear when rendered:

Example 2.1. Source code with callouts

```
require 'asciidoctor' ❶

puts Asciidoctor.render_file('sample.adoc', :header_footer => true)
❷
```

- ❶ Imports the library
- ❷ Reads, parses and renders the file

2.3.3. 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.

--

2.3.4. Passthrough blocks

The "anything goes" mechanism in AsciiDoc is the passthrough block. As its name implies, this block passes its contents through directly to the output document. When you've encountered a complex requirement that you can meet using the AsciiDoc syntax, just put the output you want inside a passthrough block.

```
++++
<video poster="images/movie-reel.png">
  <source src="videos/writing-zen.webm" type="video/webm">
</video>
++++
```



Using a passthrough block couples your document to a specific output format, such as HTML. You can use [conditional inclusion macros²](#) to declare passthrough markup for each of the backends you need to support.

The block style can be used in the absence of block delimiters to promote a paragraph to a block element.

2.4. 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 yum install asciidoc
```

or single-line quotes:

² http://asciidoc.org/userguide.html#_conditional_inclusion_macros

```
[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.

2.5. 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 determines the number of columns. Since we are putting each column on a separate line, we have to be explicit about the number of columns in the `cols` block attribute.

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

...
|===
```

We could define the header cells on one line so that the `cols` attribute is not required.

```
[options="header"]
|===
|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.

```
[options="header"]
|===
|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", options="header"]
```

```
|===
```

```
|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", options="header"]
```

```
|===
```

```
|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 [Tables³](#) 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:

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

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

```
[format="csv", options="header"]  
|===  
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 (`,`).

³ http://asciidoc.org/userguide.html#_tables

```
,===  
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.

Chapter 3. 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 [AsciiDoc User Guide](http://asciidoc.org/userguide.html)¹ to continue exploring the AsciiDoc syntax and processor.

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

¹ <http://asciidoc.org/userguide.html>

Chapter 4. Rendering your document

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

An AsciiDoc processor parses the document and translates it into a backend format, such as HTML, ePub, DocBook or PDF. AsciiDoc 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. We'll cover the most common usages of the processor in this document.

Before you can use an AsciiDoc processor, you have to install one.

4.1. Installing AsciiDoc

The original AsciiDoc processor is a Python script and can be run on any system that has Python 2 installed.¹



There's a modern implementation of AsciiDoc, written in Ruby, named AsciiDoctor. If decide to use AsciiDoctor, you'll still need to install the Python version for some conversions—such as generating a PDF—that AsciiDoctor doesn't handle at the time of this writing.

4.1.1. Installing AsciiDoc via a package manager

The quickest way to install AsciiDoc is through the package manager on your system. Here are the commands for several popular package management systems:

Ubuntu / Debian

```
$> sudo apt-get install asciidoc
```

Fedora

```
$> sudo yum -y install asciidoc
```

MacOSX

¹At the time of writing, the AsciiDoc processor does not work with Python 3.


```
$> brew install asciidoc
```



If Python isn't already installed, it should be installed as a dependency of the package.

If the install was successful, you should be able to execute the `asciidoc` and `a2x` commands from a command prompt:

```
$> asciidoc --version
asciidoc 8.6.8
$> a2x --version
a2x 8.6.8
```

4.1.2. Installing AsciiDoc manually

If you can't get the AsciiDoc processor using a package manager, you can install it manually following these steps:

1. Make sure you have Python 2 installed on your system
2. [Download the distribution archive²](#) from the AsciiDoc website.
3. Extract the archive into a directory where you keep your programs
4. Find the `asciidoc.py` and `a2x.py` files in the extracted directory
5. Rename these files to `asciidoc` and `a2x`, respectively
6. Add the extracted directory to your PATH environment variable

For more details on the manual install, see the [AsciiDoc Installation Guide³](#).

If the install was successful, you should be able to execute the `asciidoc` and `a2x` commands from a command prompt:

```
$> asciidoc --version
asciidoc 8.6.8
$> a2x --version
a2x 8.6.8
```

² <http://sourceforge.net/projects/asciidoc/files/latest/download>

³ <http://asciidoc.org/INSTALL.html>

Now you're ready to convert your document!

4.2. Converting a document to HTML 5

AsciiDoc comes with two HTML backends:

xhtml11

Creates an HTML document that adheres to the XHTML 1.1 specification and uses CSS2 for styling

html5

Create an HTML document that adheres to the HTML 5 specification and uses CSS3 for styling

The default backend in AsciiDoc is **xhtml11**. If you don't specify a backend, the AsciiDoc processor will assume you want to use **xhtml11**. We want to look forward, so we're going to use the **html5** backend.

In your command prompt, switch to the directory that contains the document and execute the following command:

```
$> asciidoc -b html5 sample.adoc
```

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 `sample.html`.

```
$> ls
sample.adoc  sample.html
```

AsciiDoc derives the name of the output document from the name of the input document by simply changing the file extension.

The document uses CSS for styling and JavaScript for generating parts of the document such as the table of contents and footnotes, which it embeds into the document so you don't have to worry about managing extra resources.

If you have image references in your document, you still have to move those with the output document. To have the images embedded into the document as well, pass the `data-uri` attribute to the processor:

```
$> asciidoc -b html5 -a data-uri sample.adoc
```

If you want the document to include a table of contents, pass the `toc` attribute to the processor:

```
$> asciidoc -b html5 -a data-uri -a toc sample.adoc
```

If you want to use `data-uri` and `toc` by default, you can add them to the header of the document:

```
= Document Title
Doc Writer <doc.writer@asciidoc.org>
2012-01-01
:data-uri:
:toc:
```

```
Content...
```

The default stylesheet that comes with AsciiDoc is quite...blue. AsciiDoc ships with two additional themes, flask and volnitsky. People tend to prefer the flask theme, which you can enable using the `theme` attribute:

```
$> asciidoc -b html5 -a theme=flask sample.adoc
```

See the [Themes⁴](#) chapter in the AsciiDoc User Guide to learn how to create your own theme.

If you'd rather supply your own stylesheet, you can specify the relative path to it using the `stylesheet` attribute:

```
$> asciidoc -b html5 -a stylesheet=mystyles.css sample.adoc
```

The custom styles will be embedded into the document, as with the default stylesheet.

Using Ascidoctor

Ascidoctor, an implementation of the AsciiDoc processor written in Ruby, is a drop-in replacement for AsciiDoc in most cases. Ascidoctor provides both a

⁴ <http://asciidoc.org/userguide.html#X99>

commandline tool and a Ruby API for converting AsciiDoc documents to HTML 5, DocBook 4.5 and custom output formats.

Asciidoctor is published to [RubyGems.org](https://rubygems.org/gems/asciidoctor)⁵ as a [Ruby Gem](http://docs.rubygems.org/read/chapter/1)⁶ (a packaging format for Ruby libraries). If you have Ruby and RubyGems on your system, you can install Asciidoctor using the `gem` command:

```
$> gem install asciidoctor
```

If you are running Fedora 17 or higher, you can install it using `yum`, which will also install Ruby if it's not yet installed on your system:

```
$> sudo yum install rubygem-asciidoctor
```

To use Asciidoctor, you simply replace `asciidoc` with `asciidoctor` in any of the commands in this section. For example, to generate an HTML document:

```
$> asciidoctor sample.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.render_file('sample.adoc', :in_place => true)
```

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

```
html = Asciidoctor.render_file('sample.adoc', :header_footer => true)  
puts html
```

To generate DocBook, just specify the backend option:

⁵ <https://rubygems.org/gems/asciidoctor>

⁶ <http://docs.rubygems.org/read/chapter/1>

```
Asciidoctor.render_file('sample.adoc', :in_place => true,  
  :backend => 'docbook')
```

You can learn more about Asciidoctor and how it differs from the AsciiDoc processor on the [Asciidoctor project page](#)⁷.

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

4.3. Converting a document to DocBook

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

AsciiDoc provides a DocBook 4.5 backend out of the box. To convert the document to this format, call the processor with the backend flag set to **docbook**:

```
$> asciidoc -b docbook sample.adoc
```

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

```
$> ls  
sample.adoc  sample.html  sample.xml
```

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

```
$> yelp sample.xml
```

DocBook is only an intermediary format in the AsciiDoc toolchain. You'll either feed it into a system that processes DocBook (like [publican](#)⁸), or you can convert it to PDF using the AsciiDoc processor.

⁷ <http://asciidoctor.org/>

⁸ <https://fedorahosted.org/publican>

4.4. Converting the document to PDF

PDF is a nice format for presenting a final version of a document. For legacy reasons, the conversion to PDF is handled by a separate program in the AsciiDoc distribution, `a2x`.

`a2x` can accept a DocBook file as input and produce a PDF:

```
$> a2x -f pdf --fop sample.xml
```

A new PDF document, named `sample.pdf`, will now be present in the current directory:

```
$> ls
sample.adoc  sample.html  sample.pdf  sample.xml
```

You can view the PDF using any PDF viewer. Go ahead, take a look!

Yes, you just created that PDF. It was that easy!

Rather than converting from AsciiDoc to DocBook and then from DocBook to PDF in two steps, `a2x` can go directly from AsciiDoc to PDF in a single call:

```
$> a2x -f pdf --fop sample.adoc
```

AsciiDoc can also be used for creating HTML-based presentations. This use further demonstrates the reusability of the AsciiDoc syntax.

4.5. Producing HTML-based presentations

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`⁹. You have to put the backend files where AsciiDoc can find them. Then you simply run the processor on your AsciiDoc document with the `deckjs` backend enabled:

⁹ <http://imakewebthings.com/deck.js>

```
$> asciidoc -b deckjs slides.adoc
```

Asciidoctor has backends for generating deck.js, dzslides and reveal.js presentations as well. Backends for other presentation frameworks are in the works.

4.6. Output galore

There's really no end to the customization you can do to the output that the AsciiDoc processor(s) generate. We've just scratched the surface here. Keep in mind that this magazine was produced from AsciiDoc, so never underestimate how far you can make the output stretch.

If you want to start tweaking the output that is generated, refer to instructions in the [AsciiDoc User Guide](#)¹⁰.

4.7. Where else is AsciiDoc rendered?

The easiest way to experiment with AsciiDoc is online. AsciiDoc document in a GitHub repository or a [gist](#)¹¹ is automatically rendered as HTML and displayed 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.

¹⁰ <http://asciidoc.org/userguide.html>

¹¹ <http://gist.github.com>

Chapter 5. 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 is to repurpose it as documentation. Almost immediately, you'll find your writing zen and enjoy the rewarding experience of producing.

If you get stuck working with the syntax, be sure to browse the [Gotchas](#)¹ section of the AsciiDoc User Guide and the [AsciiDoc FAQ](#)² to find answers. When all else fails, send your question to the [AsciiDoc discussion list](#)³.

¹ http://asciidoc.org/userguide.html#_gotchas

² <http://asciidoc.org/faq.html>

³ <http://groups.google.com/group/asciidoc>

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