AsciiSpec Userguide

Table of Contents

List of Examples

This userguide covers the basics of common and practical AsciiDoc syntax along with tips for using AsciiSpec.

Chapter 1. Document Structure

In order that we understand the use of AsciiSpec processors, it's important to know the context in which they function. This section provides a brief overview of how an AsciiDoc document is structured. The following list is a simplified overview of the AsciiDoctor AST:

`Document`	The document contains `Sections` and `Blocks` that make up the document and holds the document attributes.
`Section`	Models sections in the document and dictates the structure of the `Document` tree.
`Blocks`	Content within a `Section`, differentiated by **context** such as 'paragraph' or 'image'.
`Lists`, `Tables`, `ListItems`…	Nested content within a `Block`. Can also themselves be `Blocks`.

1.1. Sections

Section levels are set using equals symbols (`= title`) followed by a space and the title. They must be preceded by an empty line:

Section Levels

```
= Document Title (Level 0)

== Level 1 Section Title

=== Level 2 Section Title

==== Level 3 Section Title

===== Level 4 Section Title

== Another Level 1 Section Title
```

Documents with two Level 0 (`=`) Sections need the `:doctype: book` attribute set.

It's illegal to skip section:

```
== Level 1 Section

==== Level 3 Section - Error!
```

1.1.1. Styling Sections

The most useful styles that can be added to a section are `[bibliography]` and `[appendix]`. The next example demonstrates how to style a section as an Appendix:

Example 1.1. Styling Sections

Source	Output

`[appendix]`
`== Common Terms`
 	
`The following is a list of common terms used...`
 	
`[appendix]`
`== Comparison of Frameworks`
 	
`Let's examine the effectiveness of each...`
	Appendix A: Common Terms
	
	The following is a list of common terms used…
	
	Appendix B: Comparison of Frameworks
	
	Let's examine the effectiveness of each…

Chapter 2. Blocks

Usage

```
[quote]
Before I came here I was confused about this subject.
Having listened to your lecture I am still confused.
But on a higher level.
```

The above content will be rendered as follows:

> Before I came here I was confused about this subject. Having listened to your lecture I am still confused. But on a higher level.
>
> — Enrico Fermi *Notes on Quantum Mechanics (1954)*

Blocks are content in a section with styles or **`contexts`** such as paragraphs, source listings, images, etc. Square brackets **`[]`** are used to indicate the style of the block and an empty line indicates that the block has finished. All plain text of one or more lines will be parsed as a **`block`** with the 'paragraph' style by default, therefore:

```
It was the best of times..

// Is the same as writing the following:

[paragraph]
It was the best of times..
```

A list of built-in block types can be found in the AsciiDoctor User Manual: built-in blocks summary.

2.1. Titles & attributes

Adding a title to a block of content is done by adding a fullstop followed by the title text in the line previous to the block.

To style a block with a source **`listing`** context, we use **`[source]`** as with this example:

```
.Fibonacci.n4js ❶
[source,n4js] ❷
---- ❸
export public class Fibonacci {
 public seq()  {

      var arr = [];
// etc...
----
```

❶ In the first line we add a *title* to a block. This is done using a full stop followed by the title **`Fibonacci.n4js`** (note there is no space). A title can be added in this way to many different block types by default.

❷ Setting a **`source`** context and the language is **`N4JS`**.

❸ Notice the use of four hyphens to delimit the block: **`----`** (see Section 2.2, “Delimiters”) this indicates to the parser where the block begins and ends. The listing block can then also include the empty line:

Output:

Fibonacci.n4js

```
export public class Fibonacci {
 public seq()  {

      var arr = [];
```

```
// etc...
```

We can add more attributes relevant to the type of block. In the case of a `[verse]` block, we can set the author and the source separated with commas like so: `[verse, Carl Sagan, Cosmos]`.

```
.Deep Thought of the Day
[verse, Carl Sagan, Cosmos: A Personal Voyage]
If you want to make an apple pie from scratch, you must first create the universe.
```

The above is rendered as follows:

> If you want to make an apple pie from scratch, you must first create the universe.
>
> — Carl Sagan *Cosmos: A Personal Voyage*

2.2. Delimiters

For all built-in blocks, the square brackets containing the block type (e.g. `[source]`) can be omitted and their delimiters will be used to determine the block type instead. For source blocks, this is four hyphens (`----`);

This is convenient, but, of course, no positional attributes i.e. `[blocktype,attr1,attr2]` can be specified. In the case of listing blocks, this means no language can be specified for highlighting in the default manner e.g. `[source,java]`.

Example 2.1. Set Block Type by Delmiter

Source	Output
---- my code() { string example... ----	my code() { string example...

For a full list of delimiters, refer to the Asciidoctor User Manual: Built-in Block Summary.

2.3. Admonition Blocks

Usage:

```
WARNING: Don't divide by zero...
```

A useful feature built-in to AsciiDoctor is the inclusion of admonition blocks. By default, the following admonition blocks are available;

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

They render as with the `WARNING` block below, except with different Section 7.5, “Icons”.

Don't divide by zero. In ordinary arithmetic, the expression has no meaning, as there is no number which…

The standard block syntax can also be used if the admonition spans multiple paragraphs:

Example 2.2. Delimited Admonition Block

Source	Output

`[WARNING]`
`--`
`Don't divide by zero.`
 	
`In ordinary arithmetic, the expression has no meaning,`
`as there is no number which, multiplied by 0..`
`--`
	 Don't divide by zero.
	
	In ordinary arithmetic, the expression has no meaning, as there is no number which, multiplied by 0..

2.4. Nesting Blocks

Blocks can contain other blocks:

Source	Output

`.Nested Listing`
`====`
`See the following code snippet:`
 	
`----`
`my code() {`
 	
`----`
 	
`====`
	Example 2.3. Nested Listing
	
	See the following code snippet, for example:
	
	`my code() {`

Nesting blocks of the same type is done using a different number of delimiters:

Source	Output

`.Outer Example`
`=====` (1)
 	
`.Inner Example`
`====` (2)
 	
`See the following code snippet:`
 	
`[source,html]`
`----`
`<div id="footnotes">`
`----`
 	
`====`
 	
`=====`
	Example 2.4. Outer Example
	
	Example 2.5. Inner Example
	
	See the following code snippet:
	
	`<div id="footnotes">`

(1) Outer delimiter uses 5 characters

(2) Inner delimiter uses 4 characters

2.5. Block Macro

Usage:

```
macrotype::attributes[additional parameters]
```

Block macros are used to create a block member in a document.
A block macro must be on a single line by itself with an empty line before and after.

The `toc::[]` macro creates a table of contents block at that position in the document (to enable this feature, see setting document attributes below).

Another common block macro is the `image::[]` macro;

Example 2.6. Block Macro

Source	Output

`The following image is considered`
`the last line of this pargraph.`
`image::{find}images/logo.png[]`
 	
`Leaving an empty line before and after the`
`image block macro will create a block as expected:`
 	
`image::{find}images/logo.png[]`
 	
`beginning of next block...`
	The following image macro is considered the last line of this pargraph. image::images/logo.png[]
	
	Leaving an empty line before and after the image block macro will create a block as expected:
	
	 
	
	beginning of next block…

2.6. Tables

Table blocks are typically delimited by a character (usually a pipe `|`) and three equals symbols (`|===`);

```
|===
| Hello | world
|===
```

Hello	world

Example 2.7. Tables and CSV

A comma can be used exactly as above to separate cells in the following way:

```
,===
, Hello , World
,===
```

A different character can be used to delimit cells by substituting the pipe with the separator you wish to use.

```
[cols=6]
,===
include::music-collection.csv[]
,===
```

Using commas in this way can provide an easy solution to including CSV values (`include::mydata.csv[]`) into a table without having to reformat the data:

Formatting tables:

```
|===
| Backend 3+^| Description

| html (or html5) 3+| HTML5, styled with CSS3 (default).
| pdf 3+| PDF, a portable document format. Requires the asciidoctor-pdf gem.
|===
```

In the above table, formatting attributes **3+** are used. The ^ caret symbol is used to centre-align the text and **3+** indicates that the cell spans three consecutive columns.

Backend	Description
html (or html5)	HTML5, styled with CSS3 (default).
pdf	PDF, a portable document format. Requires the asciidoctor-pdf gem.

A full overview of the possibilities to create complex tables can be found in the tables section of the User Manual.

2.7. Custom AsciiSpec Blocks

For examples of custom AsciiSpec blocks (definition and requirements), refer to the Chapter 6, *AsciiSpec Cheat Sheet* and for comprehensive documentation, see the AsciiSpec processor Specification.

Chapter 3. Attributes & Variables

Usage:

```
:attribute: value

{attribute}
```

Setting document attributes is done by adding an attribute entry line as `:attribute: value` above. Variables are declared using `{}` curly brackets and can be used for substitutions. Attributes can be inserted anywhere in a document unless they are specific *header attributes* as described in the next section.

Example 3.1. Document Attributes

Source	Output
`:revnumber: 2.0` (1) `Last modified:: {docdatetime}` (2) `Revision:: {revnumber}.`	**Last modified** 2017-02-03 11:45:27 CET **Revision** 2.0.

(1) The revision number needs to be set, otherwise the attribute in the last line will be empty.
(2) `{docdatetime}` is automatically set to last time the source document is modified

Document attributes can be evaluated using the `ifdef::[]` macro to create some interesting logic. The line below checks if the backend is `HTML5` and if it is, the content in the square brackets will be included:

```
ifdef::backend-html5[This content is included in HTML5 output.]
```

Output:

Example 3.2. Evaluating Statements

The previous example shows how to include a single line of content into a document using an inline ifdef.
The same can also be done using blocks as follows:

Source	Output

`ifeval::[2 + 2 != 5]`
 	
`- [x] Two plus two does not a five make.`
 	
`- [x] Rules of mathematics still behave as expected.`
 	
`endif::[]`
	✓ Two plus two does not a five make.
	✓ Rules of mathematics still behave as expected.

AS-13

Example 3.3. Setting Attributes Inline

The following two examples set an attribute using inline shorthand:

Source	Output
`Attributes can be set inline too:` `We should {set:country:France} travel to {country}!`	Attributes can be set inline too: We should travel to France!

3.1. Header Attributes

A header starts with a document title followed by two optional lines defining author and revision information.
Finally, document-wide settings are defined by means of *header attributes* :

```
= AsciiSpec Documentation
Brian Thomas Smith
First Draft
:toc: right
```

An example header attribute is `:toc:` which sets the position of the Table of Contents in the destination document. The above example right-aligns the Table of Contents. Another option is to enable the use of the `toc::[]` block macro to insert a Table of Contents block in any section:

```
= AsciiSpec Documentation
Brian Thomas Smith
First Draft
:toc: macro

// A Table of Contents is rendered here by default

== Section two

toc::[] // But will be rendered here instead
```

A full table of the available built-in document attributes, see the Built-in Attributes section in the AsciiDoctor User Manual.

3.2. Special Variables

AsciiSpec introduces the concept of a special `{find}` variable which is designed to resolve paths. The `{find}` variable provides the means to specify files relative to the `.adoc` file no matter from where this `.adoc` file was included.

Usage:

`:myImageVar: {find}path/to/picture.png`

`image::{find}path/to/picture.png[]`

For complete documentation on this variable, see Special Variables.

3.2.1. Using `{find}` on GitHub

The `{find}` variables *cannot be resolved by GitHub* and consequently, the adoc file might not be displayed correctly, especially with respect to images.

As a solution, the adoc file should define the find variable to an empty string using the following line:

`:find:`

Using the line above, the image include resolves to `image::picture.png[]`.

Chapter 4. Includes

Type	Source	Output

Include	`include::filename.adoc[leveloffset=offset]`
	
	`include::file2.adoc[tags=tagname,tagnames]`
	
	`include::file3.adoc[lines=ranges,indent=depth]`
		Includes referenced file(s) dictated by the comma-separated attributes.
		The `{find}` variable may be used. See Special Variables.

Include tags	`tag::tagname[]`
	
	`end::tagname[]`
		These tags should be added in commented lines in the **included** document.

When including multiple documents that begin with Level 0 Sections (`= Section Title`), the following method can be used to ensure the sections are offset correctly;

Example 4.1. Level Offset for Multiple Includes

```
:leveloffset: +1

include::userguide-fragments/structure.adoc[]

include::userguide-fragments/blocks.adoc[]

// etc...

:leveloffset: -1
```

Chapter 5. CSS classes

CSS classes can be added to blocks in Asciidoc by using the the 'dot-prefix' syntax `[.css-class]` on the preceding line or by using the `role=` attribute:

Example 5.1. Applying CSS Classes

Source	Output

`[.xx-large]`
`This paragraph is assigned the `xx-large` CSS class.`
 	
`[role=blue]`
`Lovely Calming Blue Text on every character`
`of the brief, yet poignant sentence.`
	This paragraph is assigned the `xx-large` CSS class.
	
	Lovely Calming Blue Text on every character of the brief, yet poignant sentence.

Let's set the CSS class `xx-small` on the source block below using the using the `role=` attribute to change the text size for this long console log:

`[source,bash,role=xx-small]`

Output:

```
Downloading: https://repo.maven.apache.org/maven2/com/google/guava/guava/13.0-rc2/guava-13.0-rc2.pom
Downloaded: https://repo.maven.apache.org/maven2/com/google/guava/guava/13.0-rc2/guava-13.0-rc2.pom (6 KB at 60.0 KB/sec)
Downloading: http://www2.ph.ed.ac.uk/maven2/com/google/guava/guava-parent/13.0-rc2/guava-parent-13.0-rc2.pom
...
```

When using GFM [1], it's possible with the following syntax:

`[.xx-small]`
```` ```bash ````
`Downloaded: https://repo.maven.apache....`
```` ``` ````
	`[.xx-small.language-bash]` (1)
	```` ``` ````
	`Downloaded: https://repo.maven.apache....`
	```` ``` ````

(1) An example of setting multiple CSS classes on a block: `xx-small` and `language-bash` for Prism syntax highlighting.

5.1. Inline Syntax Highlighting (HTML)

When using Prism.js (AsciiSpec default), CSS classes can be set inline using `[language-name]` before the code.
Note that with blocks, a full stop before the CSS class is necessary, see the above example.

Usage: Make sure the charset `[language-html]`<meta charset="utf-8" />` is set correctly.

Output: Make sure the charset `<meta charset="utf-8" />` is set correctly.

The following example demonstrates some common cases of switching the language of inline syntax highlighting;

[1] Section 7.3, "Github Flavored Markdown"

Example 5.2. Change Inline Syntax Highlighting Language

Source

```
Check the [language-html]`<a href="favicon.ico"/>` favicon in the header. (1)
Set the following id: `<div id="menubar">` on the menubar. + (2)
Inline AsciiDoc macros (`macro:[]`) are useful! (3)
```

Output

The favicon is located `<a href="favicon.ico"/>` in the header.
The menu bar was given the following id: `<div id="menubar">`.
Inline AsciiDoc macros (`macro:[]`) are useful!

(1) Only the first inline HTML listing is highlighted.

(2) The second piece of code is highlighted as AsciiDoc, hence the coloration (or lack thereof).

(3) Asciidoc is highlighted correctly.

The `[language-html]` CSS class is overriding one already set on this section, see Section 5.2, “Applying CSS Classes to Sections”.
There is a lot of AsciiDoc to highlight in this document and so the parent section Chapter 5, *CSS classes* has been assigned the class `[.language-adoc]`.

Example 5.3. Set Syntax Highlighter Language per Block

Instead of writing `[language-html]` before every piece of inline code, a CSS class can be set to a paragraph or block. All inline source code within that paragraph will then be highlighted with the language specified:

```
[.language-html]
The favicon was set at `<a href="favicon.ico"/>` in your header.
The next thing was the menubar: `<div id="menubar">` which contained a list `<ul class="fa-ul">`...
```

The favicon was set at `<a href="favicon.ico"/>` in your header.
The next thing was the menubar: `<div id="menubar">` which contained a list `<ul class="fa-ul">`...

5.2. Applying CSS Classes to Sections

It’s also possible to set a class on a section. The highest section level that a CSS class can be applied on is the Level 1 (`==`) and all contained sections will inherit this class.

Source

```
[.language-css]
== CSS
Everything enclosed in backticks in this
section gets styled with correct CSS `@media
print {code {text-shadow: none;}}`
syntax highlighting.

=== The `font-weight: bold;` attribute

Always use `font-weight: bold;` to get your point across...
```

Output

1. CSS

Everything enclosed in backticks in this section gets styled with correct CSS `@media print {code {text-shadow: none;}}` syntax highlighting.

1.1. The `font-weight: bold;` attribute

Always use `font-weight: bold;` to get your point across…

Some custom CSS has been added to the Foundation stylesheet specifically for styling large sections in this manner.
One useful class is `[.todo]` which is demonstrated here;

Source	Output

`[.todo]`
`== Summary`
 	
`. First Item`
`.. Second Item`
 	
`=== Feature A` ❶
 	
`This feature needs documentation!`
 	
`== Overview` ❷
 	
`Fully Documented, see...`
	1. Summary
	1. First Item
	a. Second Item
	1.1. Feature A
	This feature needs documentation!
	2. Overview
	Fully Documented, see…

❶ **Subsections** will inherit the CSS class specified

❷ The next section of the same level or higher will not inherit this class.
In this example, the `Overview` section does not have the `todo` class.

5.3. Delimited by Open Blocks

CSS classes can span multiple blocks or paragraphs when delimited by two hyphens `--`:

Source	Output

`[.red]`
`--`
`This paragraph is assigned the `red` CSS class.`
 	
`- [x] All these list items will be red, too!`
`--`
 	
`The next paragraph will be styled as usual`
	This paragraph is assigned the `red` CSS class.
	✓ All these list items will be red, too!
	The next paragraph will be styled as usual

Chapter 6. AsciiSpec Cheat Sheet

Name	Source
Inline Task Macro	`task:taskId[]`

Inline BibTeX Macro	`cite:[ref,ref2(optionalPage)]`
	
	`bibliography::[]`

Inline Cwiki Macro	`cwiki:path[title=Hyperlinked Text]`
	
	`cwiki:pageID[title=Hyperlinked Text]`

Definition Block	`.title`
	`[def]`
	`--`
	My Definition
	`--`

Requirements Block	`.This is the title`
	`[req,id=RSL-3,version=1]`
	`--`
	My Super Requirement
	`--`

Extended Include	`include::{find}myfile.adoc[]`
Inline Source Link	`srclnk:[DataList#<sizes]`
Inline Math	`math:E=mc^2[]` `$C=2 \Pi r$`

Math Block	`[math]`
	`++++`
	`\sum_{i=1}^n i = {n(n+1)\over{2}}`
	`++++`

Chapter 7. Tips

7.1. Newlines & Line Breaks

New lines are consumed by AsciiDoctor, so in order to preserve line breaks, add a plus symbol at the end of a line or the `[%hardbreaks]` attribute;

Example 7.1. Line Breaks

Source	Output

`A short line. +`
`A slightly longer line. +`
`A conclusion.`
 	
`[%hardbreaks]`
`I believe in the power`
`of the imagination`
`to remake the world`
	A short line.
	A slightly longer line.
	A conclusion.
	
	I believe in the power
	of the imagination
	to remake the world

Example 7.2. List Continuation

Source	Output

`* First`
 	
`Item`
 	
`* Second`
`+` (1)
`Item`
 	
`** And a source block:`
`+`
(2)
`[source,adoc]`
`task:ASC-123[]`
	• First
	
	Item
	
	• Second
	
	Item
	
	# And a source block:
	
	`task:ASC-123[]`

(1) Plus symbol continues the list indentation

(2) Empty lines are allowed.

7.2. Escaping Characters / Macros

Escaping characters characters used for formatting can be done using plus symbols:

Source	Output
`A single asterisk: +*+ And a few literal characters ++* _ `++`	A single asterisk: * And a few literal characters * _ `

Backslashes can also be used to escape AsciiDoc formatting:

Source	Output
`Escape \*bold* formatting +` `Escape dou\**ble aster**isks`	Escape *bold* formatting Escape dou**ble aster**isks

The pass macro is also a useful way of escaping long sequences of complicated formatting.

Source	Output
`pass:[*bold*, __unde**rsc__ores, double aster**isks] +` `\pass:[*bold*, __unde**rsc__ores, double aster**isks]` (1) `\{revnumber}` (2) `+\+{revnumber}` (3)	*bold*, __unde**rsc__ores, double aster**isks pass:[**bold**, *under**rsc**ores, double aster*isks] {revnumber} \2.0

(1) The pass macro and variables can be escaped using a backslash.
(2) A backslash escapes variable substitution
(3) Preventing the escaped variable

7.3. Github Flavored Markdown

Some common Github Markdown is also supported, such as backticks used for code listings:

```
```n4js
export public class Fibonacci {
 public seq()  {

      var arr = [];
// etc...
```
```

List items and checkboxes are also supported:

```
- [x] Done!
- [ ] Not Done!

# This is an <h1> tag
## This is an <h2> tag

> We're living the future so
> the present is our past.
```

- [x] Done!
- [ ] Not Done!

This is an <h1> tag

This is an <h2> tag

> We're living the future so the present is our past.

7.4. Literal Block Shorthand

A shorthand method of creating a literal block is to add indentation to a block. Spaces or tabs on the first line will indent a block as a literal:

Source:

```
 A Single Space
 Or a Tab will Suffice!
```

7.5. Icons

When the attribute `:icons: font` is set, Font Awesome icons can be used inline using the macro `icon:name[]`.

Example 7.3. Inline Icons Example

Source	Output

`Be Careful! icon:fire[]`
 	
`The source is on icon:github[] GitHub!`
	Be Careful!
	
	The source is on GitHub!

7.6. Colons

When learning AsciiDoc syntax, it can be confusing whether to use one or two colons for certain macros. The rule is as follows:

Type	Syntax	Example
Inline	`:`	We can include this `image:test.png[]` inline
Block	`::`	The following Table of Contents `toc::[]` cannot be used inline.

7.7. Media

Embed youtube content using the syntax `video::3NjQ9b3pgIg[youtube,800,600]`.

Vimeo can be embedded using a similar syntax: `video::67480300[vimeo]`

7.7.1. Audio

Audio files can also be added to a HTML document using the `audio` macro:

```
audio::soundfiles/Cherry-MX-Blue.mp3[options="autoplay,loop"]
```

Appendix A. Resources

AsciiSpec Docs - NumberFour AsciiSpec Documentation
AsciiDoc Syntax Quick Reference - Covers most standard formatting needs.
AsciiDoctor User Manual - Reference Manual detailing document attributes, conversion settings, extended features etc.

A.1. Sublime Text packages

OmniMarkup Preview - Serves a live preview to a browser for realtime editing.
OmniMarkup Custom Fork - A custom fork that provides styles and syntax highlighting aligned with AsciiSpec.
Sublime Text AsciiDoc Package - Syntax highlighting, snippets, keymaps and more.

A.2. Document Converters

Pandoc - A universal document converter.