AsciiSpec Userguide

Table of Contents

List of Figures

List of Examples

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

Chapter 1. Converting Documents

To convert this document with AsciiSpec, `cd` to the docs directory and run:

```
asciispec userguide.adoc
```

This will use the default HTML and create a file called `userguide.html`.
To convert to HTML using a different CSS stylesheet, document attributes are passed using the `-a` flag:

```
asciispec -a stylesheet=mystyle.css userguide.adoc
```

Passing a document attribute via the command line will override that value if it has already been set in the source document.

1.1. Docbook / XML

To convert to docbook (`.xml`), use the `-b docbook` flag:

`asciispec -b docbook userguide.adoc`

Most built-in CLI parameters are described in the help page by running `asciispec -h` and in further detail in the CLI Options section of the AsciiDoctor user manual.

1.2. PDF

Apache FOP (Formatting Objects Processor) is required for higher quality **PDF** generation. A fork with custom PDF styling and syntax highlighting can be found at the following location:

https://github.com/NumberFour/asciidoctor-fopub

1. Clone the Asciidoctor Fopub repository

   ```
   git clone https://github.com/NumberFour/asciidoctor-fopub.git
   ```

2. Add the `asciidoctor-fopub/` directory to your shell profile:

   ```
   export PATH=$PATH:~/path/to/asciidoctor-fopub/
   ```

3. Confirm successful install using the `which fopub` command.

   ```
   $ which fopub
   ~/path/to/fopub
   ```

4. Convert XML to PDF using the following:

   ```
   fopub myfile.xml
   ```

An example command to convert from asciidoc to PDF would then look something like the following:

```
asciispec -b docbook myfile.adoc && fopub myfile.xml
```

1.3. Configuration File

AsciiSpec processors with configurable target URLs have to be set up by means of a configuration file. For this case, `config.adoc` can be copied to the location of the source document and used as a template. The following line of code must be included at the top of your source document:

```
include::config.adoc[]
```

> To configure a specific processor, see <<https://numberfour.github.io/asciispec,AsciiSpec Processors>>.

Chapter 2. AsciiDoctor Syntax

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`.

2.1. 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!
```

2.2. 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.

2.2.1. 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 2.2.3, "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*

2.2.2. 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 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 2.1. 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..

2.2.3. 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 2.2. 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.

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

(1) Outer delimiter uses 5 characters

(2) Inner delimiter uses 4 characters

2.2.4. 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...
```
```

2.2.5. Apply 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 2.4. 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
Downloading: https://repo.maven.apache.org/maven2/com/google/guava/guava-parent/13.0-rc2/guava-parent-13.0-rc2.pom
Downloaded: https://repo.maven.apache.org/maven2/com/google/guava/guava-parent/13.0-rc2/guava-parent-13.0-rc2.pom (3 KB at 29.0 KB/
sec)
Downloading: http://www2.ph.ed.ac.uk/maven2/com/google/guava/guava/13.0/guava-13.0.pom
Downloading: https://repo.maven.apache.org/maven2/com/google/guava/guava/13.0/guava-13.0.pom
Downloaded: https://repo.maven.apache.org/maven2/com/google/guava/guava/13.0/guava-13.0.pom (6 KB at 60.0 KB/sec)
...
```

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.

Inline

When syntax highlighting using Prism.js (AsciiSpec default), CSS classes can be set inline and the language of inline source code can be set as follows:

Example 2.5. Change Inline Syntax Highlighting Language

Source	Output
`The favicon is located [language-html]`<a href="icons/favicon.ico"/>` in your header.` `The next thing was the menubar: ` ` ...`	The favicon is located **`<a href="icons/favicon.ico"/>`** in your header. The next thing was the menubar: **`<div id="menubar">`** ...

[1] Section 2.2.4, "Github Flavored Markdown"

Example 2.6. 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="icons/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="icons/favicon.ico"/>` in your header.
The next thing was the menubar: `<div id="menubar">` which contained a list `<ul class="fa-ul">`...

2.2.6. 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	Output
`[.language-css]` `== Write less!` `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...`	Write less! Everything enclosed in backticks in this section gets styled with correct CSS `@media print {code {text-shadow: none;}}` syntax highlighting. .1. The `font-weight: bold;` attribute Always use `font-weight: bold;` to get your point across…

`[.todo]`
`== Summary`
 	
`. First Item`
`.. Second Item`
 	
`=== Feature A`
 	
`This feature needs documentation!`
 	
`== Overview` (1)
 	
`Fully Documented, see:`
	Summary
	
	1. First Item
	   a. Second Item
	
	.1. Feature A
	
	This feature needs documentation!
	
	Overview
	
	Fully Documented, see:

(1) Notice that **subsections** will inherit the CSS class specified, but the next section of the same level or higher will not. In this example, the `Overview` section does not have the `todo` class.

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

Source	Output

`--`
 	
`The next paragraph will be styled as usual`	

2.3. Block Macro

Usage:

`toc::[]`

Block macros are used to create a block member in a document. The above example creates a table of contents block at that position in the document (to enable this feature, see setting document attributes below).

`Block` vs. `Block Macro`

The difference here is that with a block macro, all parameters that dictate how the block is rendered are contained within the macro delcaration. The already-existing lines of AsciiDoc source that follow the macro are not formatted or changed by its use.

As with `Blocks`, we must also prepend and append the `Block Macro` with an empty line. The following is an example of a block macro to insert a new block with the `image` context followed by the resulting output:

Source:

Example 2.7. Blocks and empty lines

Source	Output

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

2.4. Inline Macro

Inline macros are similar to block macros except that the macro is replaced by inline content.
The syntax is different in that we use a single colon `:` instead of two `::`

Source,	Output
`We can insert a logo image:images/logo.png[Logo] directly into our paragraph...`	We can insert a logo AsciiSpec directly into our paragraph…

Optional Attributes

In the above example, we have included some optional attributes in the square brackets that close the inline macro. The first attribute is the 'Alt Text' of the image, followed by the width and height of the image. The same method of passing attributes can be applied to the block macro above e.g.

`image::images/logo.png[Logo,15,18]`.

2.5. Document 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. Here are two examples of setting an attribute via an attribute entry line and 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!

An example of a common document attribute is `imagesdir` which specifies the images directory. `imagesdir` is empty by default, therefore, `image:a.jpg[]` will look for `a.jpg` in the same directory as the source document.

```
:imagesdir: images
```

2.5.1. 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.

2.6. Tables

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

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

Hello	world

Example 2.8. 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.

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:

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

Formatting tables:

```
[cols="h,d"]
|===
| Backend 3+h| 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. Media

The above video is embedded with the following syntax: `video::3NjQ9b3pgIg[youtube,800,600]`

```
.A lovely screenshot
image::images/logo.png[]
```

Figure 2.1. A lovely screenshot

2.8. Includes

Type	Source	Output

Include tags	`tag::tagname[]`
	
	`end::tagname[]`
		N/A - Usually these tags are added in commented lines

Include	`include::filename.adoc[leveloffset=offset]`	Includes referenced file(s) dictated by the comma-separated attributes.

Type	Source	Output

	`include::file2.adoc[tags=tagname,tagname2]`
	
	`include::file3.adoc[lines=ranges,indent=depth]`	

Chapter 3. 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
	`--`

For the full specification of all AsciiSpec features, see AsciiSpec Processors.

Chapter 4. Tips

4.1. 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.

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.

Appendix B. Bibliography

bibliography::[]