A Brief Introduction to AsciiDoc

Table of Contents

The source code for this document is available at github.

Introduction

Asciidoc is a text document format similar to markdown but with extra features and a more standardized syntax. It can output to different formats like html, pdf, docx, epub, etc.

Like markdown, it’s a plain text format, so it is easy to read and edit.

I find it particularly attractive for documenting code because you can include code snippets and have them automatically formatted - by including tags in your source code, you can reference them in your asciidoc file. There are a couple of other ways to do this, but I like tags in the source code since you’ll know when you are editing this that the code is being used in documentation.

My main use-cases

Code Snippets

To include code snippets, you need to tag the code using comments like this:

```
    // tag::tag1[]
awesome code goes here
    // end::tag1[]
```

And then you can reference the tag in an include:

```
include::../../test/kotlin/org/example/LibraryTest.kt[tags=tag1]
```

And you'll get this in your generated document:

```
@Test fun someLibraryMethodReturnsTrue() {
    val classUnderTest = Library()
    assertTrue(classUnderTest.someLibraryMethod(), "someLibraryMethod should return
'true'")
}
```

- See https://docs.asciidoctor.org/asciidoc/latest/verbatim/source-blocks/

Variables

Being able to declare variables and reuse them in the document is also nice, as is being able to include other documents and even CSV files (as tables!).

I can define a variable like this:

```
// define a variable which can be used later
:pauls-software-url: https://paulr70.substack.com
```

And then use it in the document like this:

```
Visit my newsletter here {pauls-software-url}[Pauls Software Substack].
```

Which will render like this:

Visit my newsletter here Pauls Software Substack.

- See https://docs.asciidoctor.org/asciidoc/latest/attributes/custom-attributes/

CSV files

You can also include CSV files as tables:

```
.Sample CSV
[source,csv]
,===
include::./lib/src/docs/data/simple.csv[]
,===
```

Which will render like this:

Table 1. Sample CSV

id	firstname	desc
1	Homer	Dad
2	Marge	Mother

- See https://docs.asciidoctor.org/asciidoc/latest/tables/data-format/

Including other documents

You can also include other asciidoc files using the `include` directive:

```
include::lib/src/docs/includes/references.adoc[]
```

- See https://docs.asciidoctor.org/asciidoc/latest/directives/include/

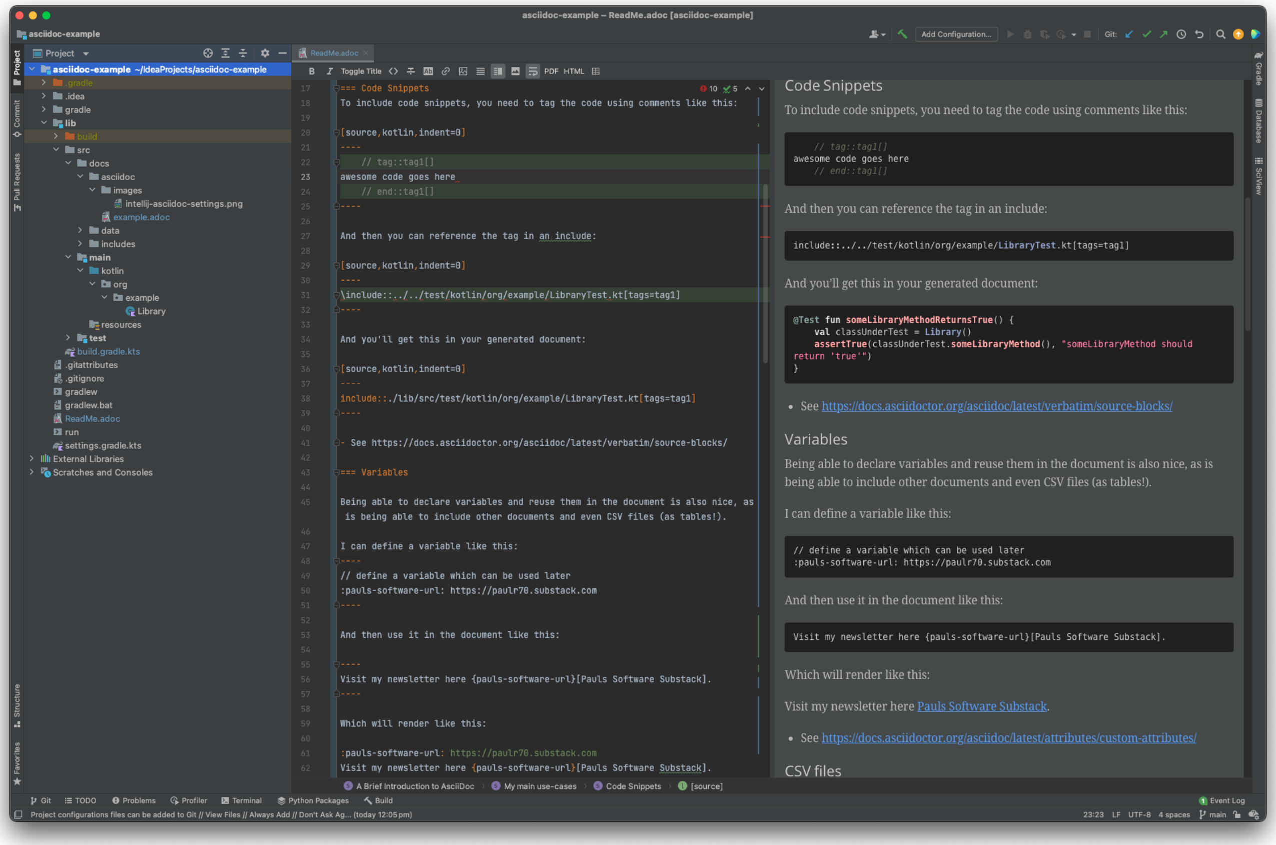
Table of contents

Create an automatic table of contents with the `toc` attribute:

```
:toc:
:toclevels: 3
```

Intellij Plugin

Intellij has a plugin for Asciidoc. It supports previewing the document in a preview pane, and you can click on the tag to jump to it. If the link doesn't resolve, you'll see the error highlighted in the editor.



```
=== Code Snippets
To include code snippets, you need to tag the code using comments like this:

[source,kotlin,indent=0]
----
    // tag::tag1[]
awesome code goes here
    // end::tag1[]
----

And then you can reference the tag in an include:

[source,kotlin,indent=0]
----
\include::../../test/kotlin/org/example/LibraryTest.kt[tags=tag1]
----

And you'll get this in your generated document:

[source,kotlin,indent=0]
----
include::./lib/src/test/kotlin/org/example/LibraryTest.kt[tags=tag1]
----

- See https://docs.asciidoctor.org/asciidoc/latest/verbatim/source-blocks/

=== Variables

Being able to declare variables and reuse them in the document is also nice, as
 is being able to include other documents and even CSV files (as tables!).

I can define a variable like this:
----
// define a variable which can be used later
:pauls-software-url: https://paulr70.substack.com
----

And then use it in the document like this:

----
Visit my newsletter here {pauls-software-url}[Pauls Software Substack].
----

Which will render like this:

:pauls-software-url: https://paulr70.substack.com
Visit my newsletter here {pauls-software-url}[Pauls Software Substack].
```

Code Snippets

To include code snippets, you need to tag the code using comments like this:

```
    // tag::tag1[]
awesome code goes here
    // end::tag1[]
```

And then you can reference the tag in an include:

```
include::../../test/kotlin/org/example/LibraryTest.kt[tags=tag1]
```

And you'll get this in your generated document:

```
@Test fun someLibraryMethodReturnsTrue() {
    val classUnderTest = Library()
    assertTrue(classUnderTest.someLibraryMethod(), "someLibraryMethod should
return 'true'")
}
```

- See https://docs.asciidoctor.org/asciidoc/latest/verbatim/source-blocks/

Variables

Being able to declare variables and reuse them in the document is also nice, as is being able to include other documents and even CSV files (as tables!).

I can define a variable like this:

```
// define a variable which can be used later
:pauls-software-url: https://paulr70.substack.com
```

And then use it in the document like this:

```
Visit my newsletter here {pauls-software-url}[Pauls Software Substack].
```

Which will render like this:

Visit my newsletter here Pauls Software Substack.

- See https://docs.asciidoctor.org/asciidoc/latest/attributes/custom-attributes/

CSV files

- See https://intellij-asciidoc-plugin.ahus1.de

Gradle

Gradle is supported, so with the right plugins configured you can generate HTML, PDF, EPUB from your asciidoc files with some simple gradle configuration like this:

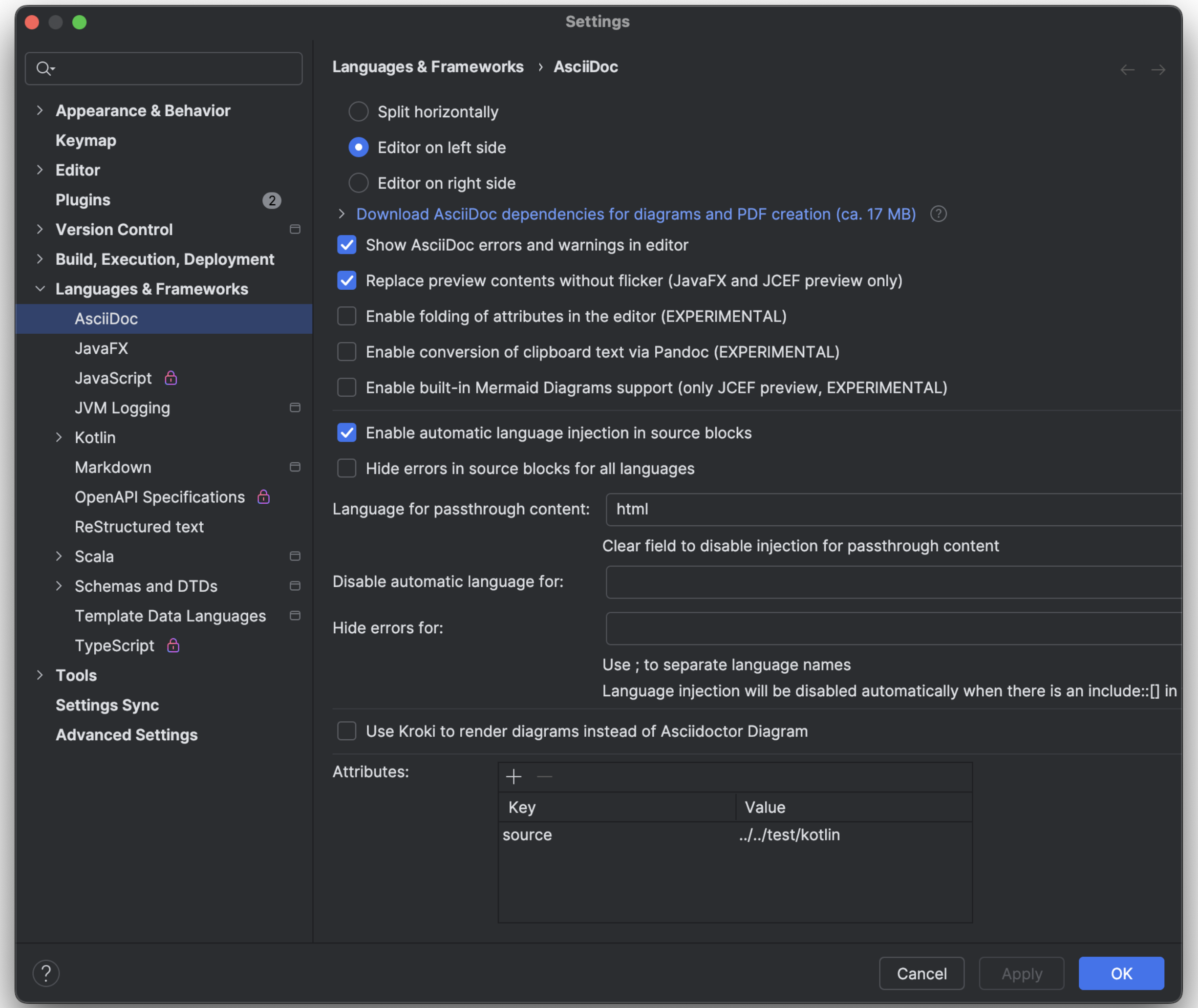
```
tasks {

    val asciidocAttributes = mapOf(
        // define a custom attribute to be used in the document eg as {source}
        // unfortunately these won't work in the intellij preview, only in the gradle
output
        // so you would need to separately define these attributes in the intellij
settings
        "source" to project.sourceSets.test.get().kotlin.srcDirs.first(),
    )

    "asciidoctor"(AsciidoctorTask::class) {
        baseDirFollowsSourceDir()
        attributes(asciidocAttributes)
    }
    "asciidoctorPdf"(AsciidoctorPdfTask::class) {
        baseDirFollowsSourceDir()
        attributes(asciidocAttributes)
    }
}
```

The `asciidocAttributes` map is used to pass variables to the asciidoc processor so you can use them in your document.

If you want these variables to also work in the Intellij preview pane you'll need to configure them in the Intellij plugin settings:


Settings

Languages & Frameworks › AsciiDoc

- Split horizontally
- [✓ Editor on left side]
- Editor on right side

› Download AsciiDoc dependencies for diagrams and PDF creation (ca. 17 MB)

- [✓] Show AsciiDoc errors and warnings in editor
- [✓] Replace preview contents without flicker (JavaFX and JCEF preview only)
- [ ] Enable folding of attributes in the editor (EXPERIMENTAL)
- [ ] Enable conversion of clipboard text via Pandoc (EXPERIMENTAL)
- [ ] Enable built-in Mermaid Diagrams support (only JCEF preview, EXPERIMENTAL)
- [✓] Enable automatic language injection in source blocks
- [ ] Hide errors in source blocks for all languages

Language for passthrough content: html

Clear field to disable injection for passthrough content

Disable automatic language for:

Hide errors for:

Use ; to separate language names

Language injection will be disabled automatically when there is an include::[] in

- [ ] Use Kroki to render diagrams instead of Asciidoctor Diagram

Attributes:

Key	Value
source	../../test/kotlin

In this example I've used defaults for everything - so my directory structure for the source looks like this:

```
└──── src
    ├──── docs
    │    ├──── asciidoc
    │    │    ├──── example.adoc
    │    │    └──── images
    │    │        └──── intellij-asciidoc-settings.png
    │    ├──── data
    │    │    └──── simple.csv
    │     └──── includes
    │         └──── references.adoc
```

Here, example.adoc is the main document, and references.adoc is included in example.adoc.

I can run gradle tasks to produce HTML or PDF like this (note that my documentation is in the `lib` sub project):

```
./gradlew lib:asciidoc
./gradlew lib:asciidocPdf
```

This produces output in the build folder:

```
├──── build
│     ├──── docs
│     │     ├──── asciidoc
│     │     │     ├──── example.html
│     │     │     └──── images
│     │     │         └──── intellij-asciidoc-settings.png
│     │     └──── asciidocPdf
│     │         └──── example.pdf
```

Conclusion

Have fun documenting your code! There's much more to Asciidoc than what I've covered here, but this is a good starting point.

GitHub will (somewhat) render asciidoc files, so you can see the output of this file here: https://github.com/prule/asciidoc-example - unfortunately github doesn't support includes, something which is mentioned here https://github.com/github/markup/issues/1095

There is a copy of the gradle generated output in `lib/example-output` if you want to see what it looks like.

References

- Asciidoctor
  - https://asciidoctor.org
  - https://docs.asciidoctor.org
- Gradle Support
  - https://asciidoctor.github.io/asciidoctor-gradle-plugin/master/
  - https://github.com/asciidoctor/asciidoctor-gradle-examples/
- Intellij Support

    - https://matthewsetter.com/asciidoc-plugin-for-intellij-review/
    - https://github.com/asciidoctor/asciidoctor-intellij-plugin
- More
    - Spring Rest Docs
        - https://docs.spring.io/spring-restdocs/docs/current/reference/htmlsingle/#working-with-asciidoctor
        - https://intellij-asciidoc-plugin.ahus1.de/docs/users-guide/features/advanced/spring-rest-docs.html