ASCIIdoc

Introduction, Common Constructs, Content Reuse and Conditional Content

TechPubs Tools Desk, written in ASCIIdoc

Revision 1.0
2018-11

Table of Contents

1

Introduction

We're going to look at some simple examples of content re-use and conditional content using the ASCIIdoc lightweight markup language (LML). Before we get into that, a quick introduction and some simple constructs.

ASCIIdoc is a plaintext LML with some simple formatting marks to create things like lists, tables, figures, and other constructs. The idea with LMLs was to provide something like XML's semantic tagging without the "angle bracket tax" and the extreme complexity of constructs like namespaces. ASCIIdoc is exhaustively documented here[1] and all over the web.

ASCIIdoc semantics are based on the DocBook XML dictionary; ASCIIdoc can therefore leverage the vast DocBook ecosystem for specialized formatting and for other powerful functionality, notably S1000D 4.X interoperability[2] and RedPen[3], which is capable of STE[4] checking.

ASCIIdoc files are text files and can be managed accordingly. The minimal markup means that the source files are easily readable, so CMS infrastructure options are varied. Possibilities include: an ASCIIdoc front end for technical publications, like Antora[5]; a stock Version Control System (VCS) like Git, Mercurial, Perforce, Team Foundation, etc.; a simple CMS; or an enterprise Wiki like FosWiki[6] or M2[7]. Finally, you could use a flat file structure, so long as your recordkeeping was solid.

1 https://ASCIIdoctor.org/docs/ASCIIdoc-syntax-quick-reference/
2 https://github.com/kibook/s1kd-tools
3 https://github.com/redpen-cc/redpen
4 https://en.wikipedia.org/wiki/Simplified_Technical_English
5 https://antora.org/
6 https://en.wikipedia.org/wiki/Foswiki

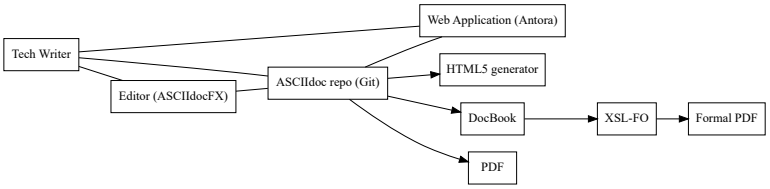

Figure 1.1. Example ASCIIdoc Infrastructure

A modern VCS is recommended; there's just too much useful infrastructure built up around tools like Git to let it go to waste. Avoid frontends that insist on customized data structures.

Incidentally, this document was written in ASCIIdoc with the ASCIIdocFX[8] writing tool.

It's worth noting I don't really know what I am doing here, but I'm nevertheless managing some pretty complex constructs and constraints.

[7] https://en.wikipedia.org/wiki/MoinMoin
[8] https://ASCIIdocfx.com/

2

Common constructs

Let’s look at some common constructs in ASCIIdoc.

2.1. Admonitions

Let’s look at some admonitions

This is a warning

This is a note. This note also has a reference anchor, which we’ll be using later.

This is a caution

In ASCIIdoc, that is rendered by the following

```
[WARNING]
====
This is a warning
====

[NOTE]
====
[[Reference]]This is a note. This note also has a reference anchor,
 which we'll be using later.
====
```

```
[CAUTION]
====
This is a caution
====
```

2.2. Internal reference

Here is an internal link, like an `xref`. Clicking [Reference] takes you back to the NOTE example, which was tagged with an anchor.

That's rendered in ASCIIdoc by the following, with "Reference" being the text of the target anchor.

```
Here is an internal link, like an `+xref+`. Clicking <<Reference>> takes
 you back to the NOTE example, which was tagged with an anchor.
```

Note that the "code literal" inline with the text (in this case, 'xref') is declared with a backtick and a plus sign.

We can also reference any heading by entering the text of the heading in double angle brackets, like this link back to the first paragraph: Chapter 1, *Introduction*

2.3. Procedure steps (numbered list)

This is a sample numbered list

SampleList

1. Procedure step 1
    a. Procedure step 1 secondary
        i. Procedure step 1 tertiary
2. Procedure step 2
3. Procedure step 3

In ASCIIdoc

```
.SampleList
```

```
. Procedure step 1
.. Procedure step 1 secondary
... Procedure step 1 tertiary
. Procedure step 2
. Procedure step 3
```

2.4. Tables

Here is a sample table

Table 2.1. Sample table

sample	**sample**	**sample**
sample	sample	sample
sample	**sample**	**sample**

Rendered in ASCIIdoc from

```
.Sample table
[width="100%",options="header,footer"]
|====================
| sample | sample | sample
| sample | sample | sample
| sample | sample | sample
|====================
```

But there's much better ways of drawing tables. Here's another more useful way of rendering a table. This is pretty neat. This takes an external CSV file and renders it as a table in the content. Death to Copy and Paste!

Table 2.2. Sample table as an external include from a CSV file

ExcelFirstRow	**ExcelFirstRow**	**ExcelFirstRow**
I am from Excel	I am from Excel	I am from Excel
I am from Excel	I am from Excel	I am from Excel
I am from Excel	I am from Excel	I am from Excel

This is accomplished via an `include`, as shown below. I've escaped the include with a `(\)` so it won't bring it in again.

```
.Sample table as an external include from a CSV file
[format="csv", options="header"]
|===
include::tools.csv[]
|===
```

2.5. Graphics

This is a sample figure in SVG format

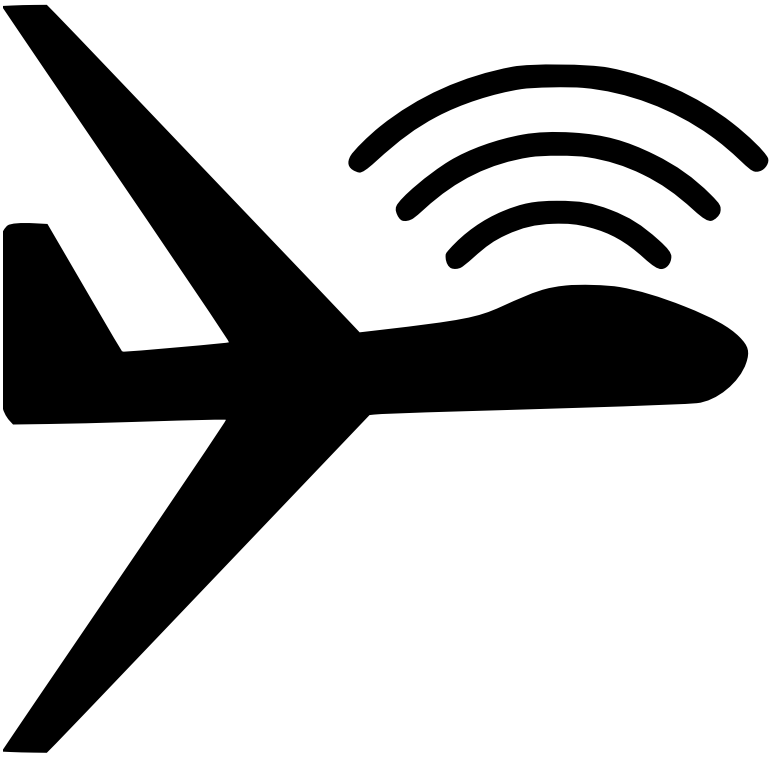

Figure 2.1. SE2

Which looks like this in ASCIIdoc:

```
.SE2

image::SE2.svg[]
```

Let's see a graphviz DOT diagram. This is rendering graphics from a textual graph description, in this case a flow diagram, that's inline with the content. Rendering as SVG.

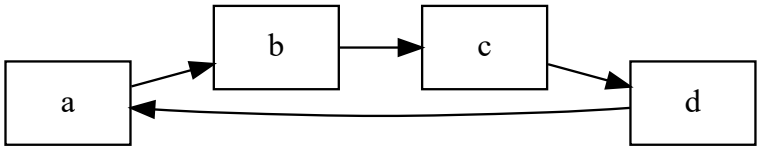

Figure 2.2. DOT diagram example

Here's the same diagram as a text file. More informatio on the DOT graphing language here[1].

```
.DOT diagram example
[graphviz, dot-example, svg]
digraph g {
    a -> b
    b -> c
    c -> d
    d -> a
}
```

Using PlantUML, we can also render UML diagrams from inline text, like so:

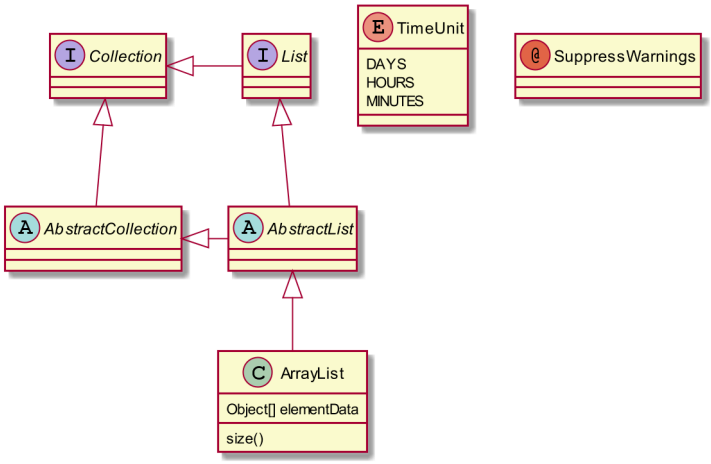

Figure 2.3. UML Diagram Example

[1] http://www.graphviz.org/doc/info/lang.html

This is rendered from the following inline UML

```
abstract class AbstractList
abstract AbstractCollection
interface List
interface Collection

List <|-- AbstractList
Collection <|-- AbstractCollection

Collection <|- List
AbstractCollection <|- AbstractList
AbstractList <|-- ArrayList

class ArrayList {
  Object[] elementData
  size()
}

enum TimeUnit {
  DAYS
  HOURS
  MINUTES
}

annotation SuppressWarnings
```

Pretty neat, and without opening so much as a single graphics application. There are many such graph languages that ASCIIdoc can render out-of-the-box.

ASCIIdoc can `include` from a large variety of different file types and source code repositories. There's a lot more capability than what's being shown here.

OK, now for the hard stuff.

3

Conditional content and content re-use

3.1. Content fragment re-use

Content re-use isn't restricted to whole documents (i.e., like a PM referencing DMs). ASCIIdoc can also re-use *chunks* of documents by by using tagged regions with the `include` directive.

Let's reference an admonition from a master warning list that's stored in an outside document called `warnlist.adoc`.

> This is the second warning in a giant list of warnings. Once again, another fragment

That was easy! That was produced by the following ASCIIdoc.

```
include::warnlist.adoc[tag=warn002]
```

You can see the include to the external file `warnlist.adoc`, and the bracketed tag command `warn002` picks out the individual content element. This mechanism can use multiple tags separated by commas, or it can use groups of nested tags.

> As with S1000D, conditions and tags require careful curation and an awareness of the audience's appetite for product complexity

3.2. Conditional content

Conditional content is handled in DITA by Conrefs, in S1000D via the Applicability mechanism, Docbook by Profiles. In ASCIIdoc it's handled via Conditional directives[1]. The one we're looking at today is the `ifeval` directive.

The following section is dynamically generated depending on the document attributes Engine and CrazySpark. These attributes were declared up in the beginning of the master document.

This information is only displayed when the Engine attribute is set to N30

3.2.1. Test include for N30

This is an include of an external document inside of a condition tag. In this include, we have a block of conditional text that is driven by the condition declared in the containing document. In this case, "Crazy Spark Plug" or CrazySpark.

CrazySpark special instructions

This information is only displayed for the CRAZY SPARK PLUG part number AAARRRGH. Here is an image inside of the ifeval directive

[1] https://github.com/asciidoctor/asciidoctor.org/blob/master/docs/_includes/conditional-preprocessor-directives.adoc

Figure 3.1. Identification of faulty ignition system

This is where we get out of the `include`

Here’s some content after our conditional block

That conditional include looked like this. We’re escaping everything with `(\)` in the example below. There is another conditional block with Engine set to N30, which only shows when the Engine attribute is set to N30.

```
ifeval::["{Engine}" == "N20"]
\This information is only displayed when the Engine attribute is set to
 N20
include::testinclude.adoc[]
```

It’s also possible to do this with multiple tags or groups of tags, or to perform an include based on document position, or to pull source code based on methods and functions. Again, there is a lot of functionality not shown in this introduction, much of it shared with DocBook.