AsciiSpec Specification

Numberfour AG

Table of Contents

The following is the specification for AsciiSpec, a toolchain based on Asciidoctor with custom modifications for task management, math support, API documentation and more.

Chapter 1. Definition Block Documentation

Usage

```
.definitionTitle
[def]
--
The content of the definition
--
```

Attributes

- **definitionTitle** (required): An anchor is derived from the definition title and embedded at the beginning of the rendered output.
- **delimiter**: Lines containing only two hyphens `--` delimit the block. This is required if the block contains empty lines or nested formatting.

Example

```
.Definition Site Structural Typing
[def]
--
If a type T is declared as structural at its definition, _T.defStructural_ is true.

1. The structurally defined type cannot be used on the right hand side of the `instanceof`
2. A type X is a subtype of a structurally defined type T...

Furthermore...
--
```

Result

Definition: Definition Site Structural Typing

If a type T is declared as structural at its definition, *T.defStructural* is true.

1. The structurally defined type cannot be used on the right hand side of the **`instanceof`**
2. A type X is a subtype of a structurally defined type T…

Furthermore…

Chapter 2. Requirements Block

Usage

```
.title
[req,id=RSL-3,version=1]
--
Contents of the requirement
--
```

Attributes

- **title** (required): An anchor is derived from the requirement title and embedded at the beginning of the rendered output.
- **ID:** (required) The ID in the form **<Prefix>-<Number>**, used to generate an anchor
- **version**: (required) value is a non-negative integer.

Omitting any of the above attributes will print an error to the console and insert a warning text in the generated document.
* **delimiter**: Lines containing only two hyphens `--` delimit the block. This is required if the block contains empty lines or nested formatting.

2.1. ID Pattern

The purpose of the `<Prefix>-<Number>` ID is to ensure that Requirements are both unique and easily referenceable. Currently, the ID may be any string, but should conform to the following conventions:

<Prefix>:
`R` (requirement) followed by the project prefix (i.e. `SL` for `stdlib`)

<Number>:
The requirement number, currently not validated. A validation stage for requirement IDs (detecting duplicates, for instance) is planned.

Example
The following example demonstrates how to document Requirement #3 for stdlib Version 1;

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

Req. RSL-3: This is the title (ver. 1)

My Super Requirement

Chapter 3. Inline Task Macro

Usage

```
task:target[]
```

The `inline task macro` creates hyperlinks to Jira task management and GitHub issue-tracking systems.

Attributes

- **target:** The project prefix followed by a hyphen and the task number or ID (e.g. `AS-23`).

3.1. URL Pattern

Specifying which repository to link to is done by adding a URL pattern to the config file in the following format:

`:task_def_<Prefix>: <Name>;<Description>;<URL-Pattern>;<Icon>;<TextPatterns>`

Example:

```
:task_def_AS-: GitHub;AsciiSpec Bugs;https://github.com/NumberFour/asciispec/issues/{TASK_ID};images/icons/github.png;AS-{TASK_ID}
:task_def_JIRA-: Jira;My Jira Board;https://jira.myorg.com/browse/JIRA-{TASK_ID};images/icons/jira.png;JIRA-{TASK_ID}
```

<Prefix>
The unique prefix by which this repository is identified (`AS-` in `task:AS-40[]`).

<Name>
The name of the repository.

<Description>
A description of the repository, used to generated a tooltip.

<URL-Pattern>

The pattern to generate in the links - can contain the following placeholders:

- `{TASK_ID}`: The suffix of the macro target (`40` in `task:AS-40[]`).

<Icon>
The relative path to an image used as an icon for the repository.

<Text-Pattern>
The text pattern to match to inline task macros in the AsciiDoc source - can also contain the `{TASK_ID}` placeholder.

3.2. Examples

Jira Task:

AsciiDoc was decided as a suitable syntax for documentation `task:JIRA-35[]`.

AsciiDoc was decided as a suitable syntax for documentation.

JIRA-35

Github Issue:

A bug has already `task:AS-35[]` been filed…

A bug has already been filed…

AS-35

Chapter 4. Inline BibTex Macro

Load references from a BibTeX file.

Usage:

```
cite:[ref(pages)]

bibliography::[]
```

Attributes:

- **ref** (required): At least one reference must be stated e.g. `ECMA15a`. Multiple references may be added by separating with commas e.g. `ECMA15a,Canning89a`.
- **pages** (optional): The specific pages of a reference.

Bibliography:

The bibliography macro can be added at any part of the source document, but the convention for block macros is to have empty lines before and after as follows:

…end of previous block

`bibliography::[]`

Beginning of next block…

Examples

Source	Output

`:bib-file: ../biblio.bib`
 	
Two of my favourite books are the ECMAScript Language Specification and F-bounded polymorphism `cite:[ECMA15a,Canning89a]`.
 	
Thanks for reading, be sure to check my biblio.
 	
`[appendix]`
`== Bibliography`
 	
`bibliography::[]`
	Two of my favourite books are the ECMAScript Language Specification and F-bounded polymorphism [ECMA15a]; [Canning89a].
	
	Thanks for reading, be sure to read my biblio.
	
	Bibliography
	
	ECMA. (2015). ECMAScript 2015 Language Specification. Retrieved from http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf
	
	Canning, Peter and Cook, William and Hill, Walter and Olthoff, Walter and Mitchell, John C.. (1989). F-bounded Polymorphism for Object-oriented Programming. Retrieved from http://doi.acm.org/10.1145/99370.99392

4.1. Configuration

The following line should be added to the **Configuration File**:

`:bib-file: <path>`

Attributes:

`<path>` can be an absolute path or a path relative to the location of the adoc file being processed. In absence of this option, or if the file denoted by `<path>` cannot be found, the processor will try to find a `.bib` file in the directory tree beginning at the location of the adoc file recursively.

Chapter 5. Inline Cwiki Macro

Usage

```
cwiki:target[title="cwikiTitle"]
```

The Inline Cwiki Macro creates hyperlinks to Confluence wiki entries. The string 'title=' and the quotation marks are optional.

Attributes

- **target:** Used to define which Confluence URL to link to. Can be one of the following:
  1. pageID (e.g. `56885484`).
  2. path (e.g. `BR/Continuous+Integration`).
- **cwikiTitle:** (optional) The title that will be displayed as an anchor in the generated hyperlink. If no title is declared, the `target` will be used instead.

5.1. URL Pattern

With most Confluence pages, the **URL path** is derived from the page title, like so:

`confluence.numberfour.eu/display/N4/Continuous+Integration`

Not all Confluence pages have associated paths and will otherwise have a **page ID** as with the following:

`confluence.numberfour.eu/pages/viewpage.action?pageId=56885484`

A target that contains only numbers (e.g., `cwiki:1234[]`) will be interpreted as a page ID rather than a page path. To configure the Inline Cwiki Macro to resolve to correct targets, the format of the URL pattern should be set in the **Configuration File** as follows:

```
:cwiki_def: <Path-URL-Pattern>;<ID-URL-Pattern>;<Icon>;<Title-Pattern>
```

<Path-URL-Pattern>
: Used to generate the target URL when a path is specified - can contain the placeholder `{PATH}`.

<ID-URL-Pattern>
: Used to generate the link target URL when a page ID is specified - can contain the placeholder `{PAGE_ID}`.

<Icon>
: A path to an icon, e.g., `images/icons/confluence.png`.

<Title-Pattern>
: the pattern used to generate the link's text - can contain the placeholder `{TITLE}`.

Examples

Documented at **cwiki:BR/Continuous+Integration[title=Continuous Integration]**…

Documented at Confluence entry:Continuous Integration…

See the **cwiki:56885484["MacOS Devices wiki entry"]** to connect to…

See the Confluence entry:MacOS Devices wiki entry to connect to…

Chapter 6. Inline Math (`math`) Macro

An inline macro for rendering LaTeX math expressions using MathML. The macro supports two syntaxes: The asciidoctor syntax and the LaTeX shorthand syntax.

Usage:

`math:EXPRESSION[]`

`$EXPRESSION$`

Attributes:

- **EXPRESSION** (required): A LaTeX math expression that does not contain `$` characters. To use a `$` character within a math expression, it must be escaped using a backslash.

Attention:

The math macro must be specified in a single line. Line breaks within this macro are not supported.

Examples

Source	Output
`Here is some text with famous inline math formulae such as math:E=mc^2[] and $C=2 \Pi r$.`	Here is some text with famous inline math formulae such as $E = mc^2$ and $C = 2\Pi r$.

6.1. Configuration

No configuration required.

It is possible to use the mathinclude::[] macro to include custom LaTeX commands.

Chapter 7. Math Block Documentation

Description

The math block is used to write blocks of mathematical formulae.

- **delimiter**: Lines containing four plus symbols `++++` delimit the block. This is required if the block contains empty lines or nested formatting.

Usage

```
[math]
++++
Some formulae.
++++
```

Example

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

7.1. Example

The above source will create the following output:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

It is possible to use the mathinclude::[] macro to include custom LaTeX commands.

Chapter 8. Math Include Documentation

Usage

`mathinclude::some/file/path.tex[]`

`mathinclude::{find}some_other_file.tex[]`

Description

The `math include` can include `.tex` files containing custom LaTeX commands to be used in mathematical expressions. It is used in conjunction with inline math and math block. Multiple math include directives are appended, so it is possible to include several files. All math include directives are processed before the inline math expressions and the math blocks are processed.

The include path may contain the `{find}`variable. In either case, the processor will search the included file by looking at each parent folder of the indicated path.

```
[math]
++++
Some maths that can use the LaTeX commands from the included file.
++++
```

Chapter 9. Inline Source Link (`srclnk`) Macro

An inline macro for generating hyperlinks from program elements such as types or methods to their source code in the code management system (i.e. GitHub). This macro is used extensively by the Eclipse export wizard.

Usage:

```
srclnk:PQN[label]
```

Attributes:

- **PQN** (required): The **p**artially **q**ualified **n**ame. Specifies the code element that is linked. See PQN Definition for the PQN's syntax.
- **label** (optional): The label of the link. The label can start/end with markups e.g. for monospace.

In case the PQN or the label contain special characters, the AsciiDoc pass macro (`++`) can be used to escape PQN and/or label.

Attention:
The srclnk macro must be specified in a single line. Line breaks within this macro are not supported.

Examples

Source	Output

The following source link of the field accessor `size()` contains a complete PQN. The link's label is the methods signature.
`srclnk:++stdlib_api:packages:`
`eu.myorg.stdlib.model.base.api:`
`src/project/model/collections/DataList:`
`DataList#<sizes++[``++public get size(): int[]++``].`
	The following source link of the field accessor `size()` contains a complete PQN. The link's label is the methods signature.
	public get sizes(): int[].

9.1. Configuration

The following commands should be added to the **Configuration File**:

9.1.1. Generated Documentation Directory

The `gen_adoc` variable specifies the directory which contains all generated documentation. It has to be set to parse the `index.idx` file and the generated adoc files, as well.

Variable:
`:gen_adoc: <path>`

Attributes:

- `<path>` can be an absolute or relative path to the location of the generated adoc documentation directory. Usually, its name is `gen_adoc`.

Example:
`:gen_adoc: data/doc/gen_adoc`

9.1.2. Repository Location

The repository location command specifies one repository, that is, its name, description and url. The url is later used to generate complete urls which point to specific source code files of the repository. The repository location command can be used multiple times to add multiple repository locations. It must be used at least once.

Variable:
`:srclnk_repo_def: <repoName>; <description>; <urlPrefix>`

Attributes:

- **`<ID>`** is the unique identifier of the variable. It has no further semantics.
- **`<urlName>`** is the name of the repository. It can be referred from the PQN.
- **`<description>`** is a short description of the repository.
- **`<urlPrefix>`** is a prefix of every url that navigates to a source code file within the repository. It contains the placeholders **`CMS_PATH`** and **`LINE_NO`** which later will be replaced by a specific file and line number, respectively.

Example:

```
:srclnk_repo_def: stdlib_api; Standard lib API; https://github.com/NumberFour/asciispec/api/blob/master/{CMS_PATH}#L{LINE_NO}
```

Chapter 10. Extended Include Macro

Asciidoctor already provides an include macro which can still be used as usual. However, if the include macro starts with specific variables after the double colon, a special handling of the include macro is activated. As of now, only one specific variable exists, which is `{find}` as in `include::{find}myFile[]`. Despite having some special handling when using specific variables, the usual behaviour of the built-in include is still preserved and all its attributes can still be used. The following documentation focuses on the include macro with activated special handling.

10.1. `{find}` Include Macro

The general idea of the `{find}` variable in an include macro is to search the given file and replace the variable with the path to the matched file. In short, it behaves the same like the `{find}` variable (see: Special Variables - Find), but enables two additional attributes which can be given in the squared brackets of the include macro.

Usage:

`include::{find}Target[Attributes]`

With:

- **Target** (required): The name of the file whose contents shall be included. The file name can contain directories. The extension `.adoc` can be omitted.
- **Attributes** (optional):
  - # **FILE_ONCE**: Includes the same file only once.
  - # **TARGET_ONCE** (alias **ONCE**): Includes the same target only once.
  - # (all other include attributes such as 'lines')

Warnings and Errors:

- **Error: File Not Found** Is issued in case the given target could not be found.
- **Error: Circular Dependency** Is issued in case the included files depend on each other. The last file which would create a circular dependency cycle is omitted. This error is only issued if all include macros of the cycle use the `{find}` macro.
- **Warning: Cant Find Circular Dependencies** Is issued in case one or more includes do not use the `{find}` macro. Despite this issue, circular dependencies can still be found if all of the causing files are included using the `{find}` macro.
- **Warning: Multiple File Matches** Is issued in case the given target is found at multiple locations. The first match is chosen to be included.
- **Warning: Inconsistent Use of Attributes** Is issued in case one of the attributes is used inconsistently. The TARGET_ONCE attribute is supposed to be used at none or at all includes with the same target. The FILE_ONCE attribute is supposed to be used at none or at all includes matching the same file.

Example:

The following line includes the file `file.adoc` in the subdir `dir`. The location of that file is found by the `{find}` directive (see: ???). The attribute `ONCE` is the shorthand form for `TARGET_ONCE`.

```
include::{find}dir/file.adoc[ONCE]
```

10.2. `{api}` Include Macro

Includes generated adoc (of StdLib API) text which previously was generated by the exporter wizard. Uses a PQN to reference the section which is to be included. (tbd)

10.3. `{src}` Include Macro

Includes source code (of StdLib API) from GitHub. Uses a PQN to reference the source element (e.g. a method) which is to be included. (tbd)

Chapter 11. Generated Anchors

The N4JS-N4 project **eu.numberfour.n4js.jsdoc2spec** generates adoc files for the documentation. These adoc files are used to create an online reference and a standalone PDF file. Moreover, the generated documentation is included into the stdlib API documentation. This stdlib API document merges both generated and manual written documentation.

When including generated adoc content into manual written documents, we might want to reference sections that are located within the included contents. Since these references rely on generated anchors used in the generated contents, an understanding of the structure of these anchors would be helpful.

11.1. Scope

Anchors in the generated documentation are created for every property of a Class or Interface. For example, for every property such as datafields or methods, a section is generated that can be referenced using a generated anchor. The following excerpt shows the beginning of the generated contents of the module n4.lang.Comparable.

```
= Module n4.lang.Comparable

== Interface Comparable

Any class that supports equality checks should implement this interface.
Comparable is not similar to Java's comparable. In N4JS, it only provides the equals method.



[[sec:spec_n4.lang.Comparable.Comparable.equals]]
[role=memberdoc]
=== ++Method equals++

[.language-n4js]
==== Signature
srclnk:++stdlib_api:packages:eu.numberfour.n4js.base.api:src/n4js/n4/lang/Comparable:Comparable#equals++[``++public abstract equals(other:
 any): boolean++``]

==== Description

returns true if the provided entity is equal to the current object by loose equality rules.
Other may be null, in that case, false is to be returned.
```

The example above contains one generated anchor: `sec:spec_n4.lang.Comparable.Comparable.equals` which references the `equals` method in the interface `Comparable` in the module `n4.lang.Comparable`.

11.2. General Structure

Generally, the structure of anchors is similar to the structure of PQNs. However, the anchor structure is shorter and of defined length:

```
sec:spec_<module><type><delimiter><property>
```

module
: specifies the name of a module

type
: specifies the name of a type, e.g. a class in the module

delimiter
: specifies the delimiter between type and property

property
: specifies the name of one property in the type

The delimiters in anchors differ from the PQN definition. PQNs use strings for example like `#` or `@>` to access non static properties or static setters. Since these strings contains characters which are permitted in adoc anchors, they are replaced as follows:

Delimiters

Name	PQN	Anchor
non-static property	#	.
non-static getter	#<	.getter.
non-static setter	#>	.setter.
static property	@	.static.
static getter	@<	.static.getter.
static setter	@>	.static.setter.

Example:

`sec:spec_n4.lang.Identifiable.Identifiable.getter.id`

11.3. Special Characters

Special characters occur in anchors since property names can contain any characters. In addition, the iterator symbol, which is used for iterators, often contains the `#` symbol. Consequently, anchors have to escape these characters. This is done by replacing the special character with a colon and its corresponding unicode number. For example, the property name `#iterator` is transformed to `:23iterator`. The resulting complete anchor is: `sec:spec_n4.lang.Delegate.Delegate.:23iterator`.

Chapter 12. Partially Qualified Names (PQNs)

A partially qualified name is a string that uniquely identifies a source element. The following types of source elements can be specified with a PQN:

- properties of types like classes or interfaces,
- top level elements such as functions and variables.

In its longest form, a PQN structure can look like this:

```
<repository>:<repository-path>:<project>:<src-folder>/<module>:<classifier>#<member>
```

However, the goal of PQNs is to be able to identify source elements with very short names as long as they are still unique. Therefore, if a source repository contains only one class with the name "PathSelector", then the string "PathSelector" should also be a PQN of that class. If, however, a repository contains another class of the same name at another location, then it is necessary to add further information to the name, beginning at its end. Let's assume that two different modules contain a class with the name `PathSelector`. In that case, it is necessary to include the module in the PQNs of the classes, e.g. `a/b/module1:PathSelector` and `a/c/module2:PathSelector`.

The syntax of a PQN is defined by the following BNF specifications:

```
PQN                   ::= (((((
        REPOSITORY_NAME ":")?
        REPOSITORY_PATH ":")?
        PROJECT_NAME ":")?
        SRC_FOLDER_SPEC "/")?
        MODULE_SPEC ":")?
        LINKABLE_ELEMENT_SPEC

LINKABLE_ELEMENT_SPEC ::= TOPLEVEL_ELEMENT_SPEC | LINKABLE_MEMBER_SPEC
TOPLEVEL_ELEMENT_SPEC ::= CLASSIFIER_NAME | FUNCTION_NAME
LINKABLE_MEMBER_SPEC  ::= ((CLASSIFIER_NAME)? DELIMITER)? MEMBER_NAME
DELIMITER             ::= "#" | "#<" | "#>" | "@" | "@<" | "@>"

REPOSITORY_PATH       ::= PATH
SRC_FOLDER_SPEC       ::= PATH
MODULE_SPEC           ::= PATH

PATH                  ::= PATH_ELEMENT ( "/" PATH_ELEMENT )*
PATH_ELEMENT          ::= CHAR*

REPOSITORY_NAME       ::= CHAR*
PROJECT_NAME          ::= CHAR*
CLASSIFIER_NAME       ::= CHAR*
MEMBER_NAME           ::= CHAR*
```

Note the different versions of the DELIMITER to differentiate between static and non-static members, and also to indicate getters and setters. To identify non-static data fields or methods of a class (or interface), a `#` is used. However, N4JS allows for multiple members to have the same name, e.g. there can be a static member with the same name as an instance member, or a getter/setter pair sharing the same name. In such cases, the member is differentiated using the following DELIMITERs: `pass:[#<+` indicates a getter, `#>` indicates a setter, `@` indicates a static member, `@<` indicates a static getter, and `@>` indicates a static setter.

Note that the PQN is strictly structured from right to left. This means that both, DELIMITER and CLASSIFIER_NAME, must be specified when the MODULE_SPEC is specified, although the BNF indicates otherwise.

Chapter 13. Special Variables in Asciispec

Some macros of Asciispec implement special variables which can be used by AsciiDoc authors. These special variables are readonly.

13.1. The `{find}` Variable

13.1.1. Introduction

There exist several use cases for adoc documents. They are used to generate html pdf files, or they are written manually. Moreover, some documents consist of more than one adoc file and thus rely on including other adoc documents. In the latter case, also the included documents can be translated separately to generate a PDF for a single chapter only for example.

Several macros such as include, image, cross reference, or other custom macros rely on paths to work appropriately. Since we have different use cases mentioned above, we need to specify paths relative to the including document. As an example, the adoc file `doc.adoc` specifies the location of the bibliography using the directive `:bib-file: ../biblio.bib`. This path can not always be found when the doc.adoc file is translated both on its own and included from onother files.

The `{find}` variable provides the means to specify files relative to the adoc file no matter from where this adoc file was included.

13.1.2. Usage

Syntax:

`{find}Target`

Target:

an arbitrary file which can include a path

Example 1:

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

Example 2:

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

13.1.3. Semantics

During the preprocessing of the document, all find variables are replaced by a concrete path to the targeted file. This path is relative to the directory of the base/master file. In order to replace the target, a search is performed which can have three outcomes:

- No file was found, which results in an error.
- One file was found.
- Multiple files were found, which results in a warning.

AsciiSpec

The search is performed in several directories. It starts in the directory of the including file and walks up the folder structure. In each of these directories, the target file is searched. All matches are collected and the first is returned. The search algorithm never descends into a subfolder (except if the target file specifies subfolders).

13.1.4. Warnings and Errors

- **Error: File Not Found** Is issued in case the given target could not be found.
- **Warning: Multiple File Matches** Is issued in case the given target is found at multiple locations. The first match is chosen to be included.

13.1.5. Using `{find}` on GitHub

In case the an adoc file should also be able to be viewed on GitHub directly, the `{find}` variables can not be resolved by GitHub. Consequently, the adoc file might not be displayed correctly, especially with respect to images included via `image::`{find}`picture.png[]`.

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[]`.

Appendix A. Bibliography