Tastydoc: a cocumentation tool for dotty using Tasty files

Bryan Abate

Supervised by Aleksander Boruch-Gruszecki

May 23, 2019

Abstract

The current documentation tool (dottydoc) relies on compiler internals and low level code. The tool introduced here aims to build a program not dependant on compiler internals but instead use Tasty files which are output when a Scala program is compiled.

It also aims at providing a tool with less bugs, more features and which is more easily maintenable.

For flexibility the output is in Markdown instead of the commonly used HTML.

Contents

Chapter 1

Introduction

In dotty the documentation generation tool is called `dotty-doc`. However the tool is flawed in many ways, these flaws include:

- Reliance on compiler internals
- Low level and unecessarily complex code which make it hard to maintain
- Not maintened and not documented
- Not flexible in its output, it outputs HTML/css in a hard to modify way
- Malformed type output such as `[32m"getOffset"[0m`
- Classes often show to be extending Object instead of their superclass. Example: `https://dotty.epfl.ch/api/scala/Conversion.HTML`

  abstract class Conversion [–T, +U] **extends** Object **with** Function1
- Annotations which sould not be display, sometimes are. Example:
- Lacks some features, especially:
  - No known subclasses
  –

The tool introduced here aims to address all these issues while providing with a code easy to maintain and easy to adapt to every need.

One major difference with currrent documentation tools is that the output is in Markdown instead of HTML, this has some pros and cons which will be discussed below.

Althought dotty-doc has some problems, the current tool will draw inspiration for some of its architecture which will allow for its parsing code for user documentation (modified to handle this structure and output Markdown) to be reused.

The report will follow the structure described in this introduction and will conclude with problems of the current implementation and further work.

Chapter 2

Features

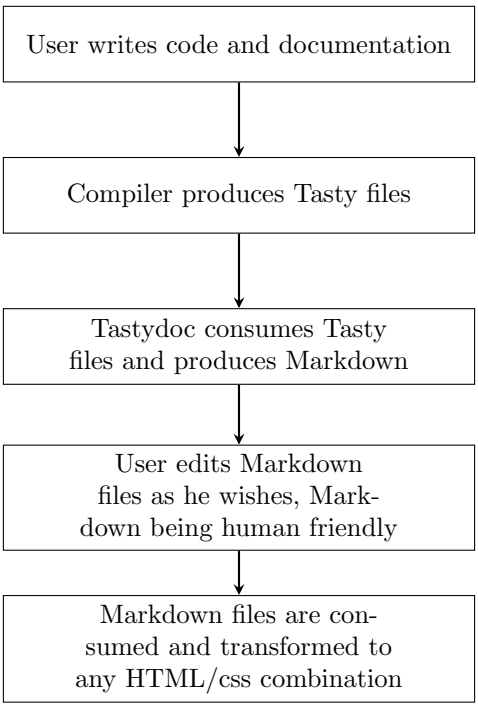
Chapter 3

Output format

Before describing the output, we list the reasons, pros and cons of using a Markdown output.

3.1 Reasoning behind Markdown

The major reason to use Markdown instead of HTML is to think of the documentation tool as a pipeline which would look like this:



With such a pipeline we do not limit the user to use a specific output for the documentation, he can easily remove, add parts, etc. manually and then use an HTML format of his choosing as Markdown only specify the structure not how it should be displayed.

3.2 Pros and Cons

Using Markdown as an intermediate output format comes with some pros and cons comparing to an HTML output:

- **Pros:**
  - Human readable and editable, meaning part of the documentation can be written manually
  - Easy to infer the format of the output to extend it manually
  - Live preview and editing
  - Only define the structure of the output not how to display it
  - Easily pipelined to another format (HTML, reStructuredText, etc.)
- **Cons:**
  - No Scala (or Java) library available to output markdown with escaping
  - Markdown fenced code block cannot contain links which forces us to use html code block, hence loose syntax highlight in Markdown and not output "pure" Markdown.
  - Markdown does not have "division" like `<div>` in HTML to better structure the output

3.3 Output structure

In the following section, we describe how the markdown output is structured.

3.3.1 class, object and trait

Have their own `.md` file.

1. Name (header 1)
2. Companion object (header 2)
3. Signature (HTML pre + code)
4. User documentation
5. Annotations (header 2)

6. Known subclasses (header 2)
7. Constructors (header 2)
   (a) Name + paremeters (HTML pre + code)
   (b) User documentation
8. Members (header 2) (In order: Abstract Type Members, Concrete Type Members, Abstract Value Members, Concrete Value Members)
   (a) Follows structure described in subsection 3.3.3

3.3.2 Package

Has its own `.md` file.

1. Name (header 1)
2. Members (header 2)
   (a) Follows structure described in subsection 3.3.3 and subsection 3.3.4

3.3.3 Type alias, class (simplified format), def, val and var

No `.md` file.

1. Name (header 3)
2. Annotations + signature (HTML pre + code)
3. User documentation

3.3.4 Package (simplified format)

No `.md` file.

1. name + link (HTML pre + code)

Chapter 4

Architecture

4.1 Use of dotty-doc parsing

Chapter 5

Problems encountered

Chapter 6

Further work

Chapter 7

Conclusion