

Attributes in Markdown

Vít Novotný

Abstract

Markup languages provide only a finite set of elements, whereas the wants of users are infinite. To bridge this gap, markup languages allow users to extend them with attributes.

In this article, we introduce attributes in the lightweight markup language of Markdown. We also show how writers can type them and how coders can style them using the Markdown package for \TeX .

Introduction

Markup languages provide only a finite set of elements to writers. This is especially true in lightweight markup languages such as AsciiDoc, Org-mode, and Markdown, which use ASCII punctuation marks and other non-letter symbols for tags. As a result, writers are often left unable to express their intent using the markup language.

In many markup languages, users can add new elements using syntax extensions. For example, in the Markdown package for \TeX , writers can add tables using `pipeTables` and `tableCaptions` options:

```
\documentclass{article}
\usepackage[pipeTables, tableCaptions]
{markdown}
\begin{document}
\begin{markdown}
  Right | Left | Center
  -----|:-----|:-----:
      12 |  12 |   12
     123 | 123 |  123
       1 |   1 |    1
: Example table
\end{markdown}
\end{document}
```

Possible output:

Table: Example table		
Right	Left	Center
12	12	12
123	123	123
1	1	1

Since version 2.17.0 from October 2022, users can also write their own syntax extensions in Lua [1, Section 2.2]. However, writing syntax extension is a costly process that requires advanced coding skills.

Furthermore, in some markup languages, users can also mix different markup languages. For example, in the Markdown package for \TeX , writers can easily mix Markdown with YAML metadata, \TeX commands, and HTML tags:

```
\documentclass{article}
\usepackage[jekyllData, html,
  rawAttribute, texMathDollars]
{markdown}
```

```
\begin{document}
\begin{markdown}
---
title: |
  Example Document in YAML,  $\text{\TeX}$ ,
  <abbr>HTML</abbr>, and Markdown
author: Vít Novotný
date: 2023-03-24
---
# Introduction
Use YAML for metadata, *Markdown* for text,
 $\text{\TeX}$  for  $\$math\$$  and formatting, and
<abbr>HTML</abbr> for extended markup.
\end{markdown}
\end{document}
```

However, mixing different markup languages makes the text more difficult to read, typeset, and less suitable for multitarget publishing.

Lastly, in most markup languages, users can attach attributes to elements to denote additional information. For example, in version 2.22.0 of the Markdown package for \TeX from March 2023, writers can easily attach attributes to Markdown headings, text spans and blocks, inline code spans and code blocks, links, and images:

```
\documentclass{article}
\usepackage[
  bracketedSpans, fencedCode, fencedDivs,
  fencedCodeAttributes, headerAttributes,
  inlineCodeAttributes, linkAttributes
]{markdown}
\begin{document}
\begin{markdown}
Introduction {#introduction}
=====
Here is an [important]{color=red} text
that describes the implementation of the
[Quicksort][1] algorithm in Haskell:

~~~~~ haskell {.numberLines startFrom=100}
qsort [] = []
qsort (x:xs) = qsort (filter (< x) xs)
              ++ [x]
              ++ qsort (filter (>= x) xs)
~~~~~

[1]: https://wikipedia.org/wiki/Quicksort
     {.external-link .wikipedia}
\end{markdown}
\end{document}
```

In the above example, attributes are written between pairs of curly braces (`{}`). When used in moderation, attributes can work around the shortcomings of markup languages without decreasing readability.

In this article, we introduce attributes in Markdown. In Section 1, we describe the kinds of attributes available in Markdown and how writers can use them. In Section 2, we show how coders can style attributes using the Markdown package for T_EX. In Section 3, we discuss the changes to attributes in the next major version of the Markdown package. We conclude in Section 4 by summarizing the contributions of the article.

1 Writer’s workshop

In Markdown, writers can use three different kinds of attributes: identifiers, class names, and key-values.

Identifiers are the most common type of attributes. Writers can use identifiers to assign a unique label to an element and refer to it, similar to the `\label` and `\ref` commands in L^AT_EX:

```
\documentclass{article}
\usepackage[headerAttributes,
            relativeReferences]{markdown}
\begin{document}
\begin{markdown}
We conclude in Section <#conclusion>.
```

```
Conclusion {#conclusion}
=====
```

```
In this paper, we have discovered that most
grandmas would rather eat dinner with their
grandchildren than get eaten. Begone, wolf!
\end{markdown}
\end{document}
```

Possible output:

<p>We conclude in Section X.</p> <p>X Conclusion</p> <p>In this paper, we have discovered that most grandmas would rather eat dinner with their grandchildren than get eaten. Begone, wolf!</p>
--

Unlike identifiers, class names do not uniquely identify an element. Instead, they place an element into a category. For example, writers can use a class name such as `fruit` to mark all occurrences of fruit:

```
\documentclass{article}
\usepackage[bracketedSpans]{markdown}
\begin{document}
\begin{markdown}
[Mango]{.fruit} is the king of all fruits.
[Oranges]{.fruit} are full of Vitamin C.
An [apple]{.fruit} a day keeps doctor away.
\end{markdown}
\end{document}
```

Even if we are undecided how the output should look, adding the attributes allows us to easily style all occurrences of fruit in our document later on.

Whereas class names denote coarse-grained category membership, key-values describe fine-grained traits of an element. For example, writers can use key-values such as `width` and `height` for image size:

```
\documentclass{article}
\usepackage[linkAttributes]{markdown}
\begin{document}
\begin{markdown}

![] (example-image){width=3cm height=2cm}

\end{markdown}
\end{document}
```

2 Coder’s cubicle

In version 2.22.0 of the Markdown package, writers can attach three different types of attributes to seven different types of elements. To prevent a combinatorial explosion, attributes and element types are encoded separately in the abstract syntax tree of a document. Consider the following example document:

```
\documentclass{article}
\usepackage[bracketedSpans, linkAttributes,
            inlineCodeAttributes]{markdown}
\begin{document}
\begin{markdown}

[This text]{.red} is so important it glows
red (grayscaled for print). So does this
<https://link>{.red} and this `code`{.red}.

\end{markdown}
\end{document}
```

The document would produce the following abstract syntax tree:

```
\bracketedSpanAttributeContextBegin
  \attributeClassName{red}%
  This text%
\bracketedSpanAttributeContextEnd{}
is so important it glows red
(grayscaled for print). So is this
\linkAttributeContextBegin
  \attributeClassName{red}%
  \link{https://link}%
    {https://link}%
    {https://link}%
  {}%
\linkAttributeContextEnd{}
and this
\codeSpanAttributeContextBegin
  \attributeClassName{red}%
  \codeSpan{code}%
\codeSpanAttributeContextEnd
```

This allows us to easily style the class name `red` independently on the element that it is attached to:

```
\ExplSyntaxOn
\markdownSetup {
  renderers = {
    *ContextBegin = {
      \color_group_begin:
    },
    attributeClassName = {
      \str_if_eq:nnT
      { #1 } { red }
      { \color_select:n { red } }
    },
    *ContextEnd = {
      \color_group_end:
    },
  }
}
\ExplSyntaxOff
```

Output:

This text is so important it glows red (grayscaled for print). So does this <https://link> and this code.

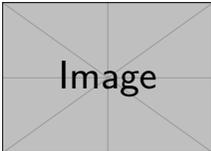
By contrast, consider the last document from the previous section, which would produce the following abstract syntax tree:

```
\imageAttributeContextBegin
  \attributeKeyValue{height}{2cm}%
  \attributeKeyValue{width}{3cm}%
  \image{}{example-image}{example-image}{}%
\imageAttributeContextEnd
```

Here, we want to style the key-values `width` and `height` only for images:

```
\RequirePackage{graphicx}
\ExplSyntaxOn
\markdownSetup {
  renderers = {
    imageAttributeContextEnd = {
      \group_end:
    },
    imageAttributeContextBegin = {
      \group_begin:
      \markdownSetup {
        renderers = {
          attributeKeyValue = {
            \setkeys % Pass the key-value
            { Gin } % to graphicx package
            { { ##1 } = { ##2 } }
          },
        },
      }
    },
  }
}
\ExplSyntaxOff
```

Output:



3 Developer's den

For all Markdown elements except headings, the `*AttributeContextBegin` and `End` renderers immediately surround the element in the abstract syntax tree. By contrast, the `headerAttributeContextEnd` renderer is placed after the end of the section implied by the heading. While this is practical for styling whole sections [2, Section 2.4], it is inconsistent and makes other common use cases such as expanding the `attributeIdentifier` renderer to the `\label` \LaTeX command more difficult to implement.

In version 2.21.0 of the Markdown package from February 2023, the `sectionBegin` and `End` renderers were added, which surround sections implied by headings regardless of whether attributes are used. In version 3.0.0 of the Markdown package, currently scheduled for release around June 2023, the `headerAttributeContextEnd` renderer will appear immediately after headings in abstract syntax tree.

4 Conclusion

Attributes provide a simple bottom-up mechanism for extending markup languages with new concepts. In this article, we have shown the types of attributes that are available in the lightweight markup language of Markdown. We have also shown how writers can type attributes in their documents and how coders can style attributes using the Markdown package for \TeX . With attributes, writers can produce beautiful documents without littering them with formatting commands.

References

- [1] V. Novotný. Markdown 2.17.1: What's new, what's next? *TUGboat* 43(3):276–278, 2022. doi.org/10.47397/tb/43-3/tb135novotny-markdown
- [2] V. Novotný, D. Reháček, et al. Markdown 2.15.0: What's new? *TUGboat* 43(1):10–15, 2022. doi.org/10.47397/tb/43-1/tb133novotny-markdown

◊ Vít Novotný
 Studená 453/15
 Brno, 638 00
 Czech Republic
 witiko (at) mail dot muni dot cz
 github.com/witiko