

# A Markdown Interpreter for T<sub>E</sub>X

Vít Novotný  
witiko@mail.muni.cz

Version 2.15.0-0-g9296cf1  
2022/03/31

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>	<b>3</b>	<b>Implementation</b>	<b>84</b>
1.1	Requirements . . . . .	2	3.1	Lua Implementation . . . . .	85
1.2	Feedback . . . . .	5	3.2	Plain T <sub>E</sub> X Implementation	189
1.3	Acknowledgements . . . . .	5	3.3	L <sup>A</sup> T <sub>E</sub> X Implementation . . . . .	206
<b>2</b>	<b>Interfaces</b>	<b>6</b>	3.4	ConT <sub>E</sub> Xt Implementation	224
2.1	Lua Interface . . . . .	6			
2.2	Plain T <sub>E</sub> X Interface . . . . .	34			
2.3	L <sup>A</sup> T <sub>E</sub> X Interface . . . . .	70			
2.4	ConT <sub>E</sub> Xt Interface . . . . .	84			
				<b>References</b>	<b>230</b>
				<b>Index</b>	<b>231</b>

## List of Figures

1	A block diagram of the Markdown package . . . . .	7
2	A sequence diagram of typesetting a document using the T <sub>E</sub> X interface . . . . .	31
3	A sequence diagram of typesetting a document using the Lua CLI . . . . .	32
4	Various formats of mathematical formulae . . . . .	76
5	The banner of the Markdown package . . . . .	77
6	A pushdown automaton that recognizes T <sub>E</sub> X comments . . . . .	152

## 1 Introduction

The Markdown package<sup>1</sup> converts markdown<sup>2</sup> markup to T<sub>E</sub>X commands. The functionality is provided both as a Lua module and as plain T<sub>E</sub>X, L<sup>A</sup>T<sub>E</sub>X, and ConT<sub>E</sub>Xt macro packages that can be used to directly typeset T<sub>E</sub>X documents containing markdown markup. Unlike other converters, the Markdown package does not require any external programs, and makes it easy to redefine how each and every markdown element is rendered. Creative abuse of the markdown syntax is encouraged. 😊

This document is a technical documentation for the Markdown package. It consists of three sections. This section introduces the package and outlines its prerequisites. Section 2 describes the interfaces exposed by the package. Section 3 describes the

<sup>1</sup>See <https://ctan.org/pkg/markdown>.

<sup>2</sup>See <https://daringfireball.net/projects/markdown/basics>.

implementation of the package. The technical documentation contains only a limited number of tutorials and code examples. You can find more of these in the user manual.<sup>3</sup>

```
1 local metadata = {
2   version   = "(((VERSION)))",
3   comment   = "A module for the conversion from markdown to plain TeX",
4   author    = "John MacFarlane, Hans Hagen, Vít Novotný",
5   copyright = {"2009-2016 John MacFarlane, Hans Hagen",
6               "2016-2022 Vít Novotný"},
7   license   = "LPPL 1.3c"
8 }
9
10 if not modules then modules = { } end
11 modules['markdown'] = metadata
```

## 1.1 Requirements

This section gives an overview of all resources required by the package.

### 1.1.1 Lua Requirements

The Lua part of the package requires that the following Lua modules are available from within the LuaTeX engine:

**LPeg  $\geq$  0.10** A pattern-matching library for the writing of recursive descent parsers via the Parsing Expression Grammars (PEGs). It is used by the Lunamark library to parse the markdown input. LPeg  $\geq$  0.10 is included in LuaTeX  $\geq$  0.72.0 (TeXLive  $\geq$  2013).

```
12 local lpeg = require("lpeg")
```

**Selene Unicode** A library that provides support for the processing of wide strings. It is used by the Lunamark library to cast image, link, and footnote tags to the lower case. Selene Unicode is included in all releases of LuaTeX (TeXLive  $\geq$  2008).

```
13 local ran_ok, unicode = pcall(require, "unicode")
```

If the Selene Unicode library is unavailable and we are using Lua  $\geq$  5.3, we will use the built-in support for Unicode.

```
14 if not ran_ok then
15   unicode = {"utf8"}={char=utf8.char}}
16 end
```

---

<sup>3</sup>See <http://mirrors.ctan.org/macros/generic/markdown/markdown.html>.

**MD5** A library that provides MD5 crypto functions. It is used by the Lunamark library to compute the digest of the input for caching purposes. MD5 is included in all releases of LuaTeX (TeXLive  $\geq$  2008).

```
17 local md5 = require("md5")
```

All the abovelisted modules are statically linked into the current version of the LuaTeX engine [1, Section 3.3]. Beside these, we also carry the following third-party Lua libraries:

**api7/luatinyaml** A library that provides a regex-based recursive descent YAML (subset) parser that is used to read YAML metadata when the `jekyllData` option is enabled.

### 1.1.2 Plain TeX Requirements

The plain TeX part of the package requires that the plain TeX format (or its superset) is loaded, all the Lua prerequisites (see Section 1.1.1), and the following package:

**expl3** A package that enables the expl3 language from the L<sup>A</sup>T<sub>E</sub>X3 kernel in TeX Live  $\leq$  2019. It is not used for anything... yet.

```
18 \ifx\ExplSyntaxOn\undefined
19   \input expl3-generic\relax
20 \fi
```

The plain TeX part of the package also requires the following Lua module:

**Lua File System** A library that provides access to the filesystem via OS-specific syscalls. It is used by the plain TeX code to create the cache directory specified by the `\markdownOptionCacheDir` macro before interfacing with the Lunamark library. Lua File System is included in all releases of LuaTeX (TeXLive  $\geq$  2008).

The plain TeX code makes use of the `isdir` method that was added to the Lua File System library by the LuaTeX engine developers [1, Section 3.2].

The Lua File System module is statically linked into the LuaTeX engine [1, Section 3.3].

Unless you convert markdown documents to TeX manually using the Lua command-line interface (see Section 2.1.5), the plain TeX part of the package will require that either the LuaTeX `\directlua` primitive or the shell access file stream 18 is available in your TeX engine. If only the shell access file stream is available in your TeX engine (as is the case with pdfTeX and XeTeX) or if you enforce the use of shell using the `\markdownMode` macro, then unless your TeX engine is globally configured to enable shell access, you will need to provide the `-shell-escape` parameter to your engine when typesetting a document.

### 1.1.3 L<sup>A</sup>T<sub>E</sub>X Requirements

The L<sup>A</sup>T<sub>E</sub>X part of the package requires that the L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> format is loaded,

```
21 \NeedsTeXFormat{LaTeX2e}%
```

a T<sub>E</sub>X engine that extends ε-T<sub>E</sub>X, all the plain T<sub>E</sub>X prerequisites (see Section 1.1.2), and the following L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> packages:

**keyval** A package that enables the creation of parameter sets. This package is used to provide the `\markdownSetup` macro, the package options processing, as well as the parameters of the `markdown*` L<sup>A</sup>T<sub>E</sub>X environment.

```
22 \RequirePackage{keyval}
```

**xstring** A package that provides useful macros for manipulating strings of tokens.

```
23 \RequirePackage{xstring}
```

The following packages are soft prerequisites. They are only used to provide default token renderer prototypes (see sections 2.2.4 and 3.3.4) or L<sup>A</sup>T<sub>E</sub>X themes (see Section 2.3.2.2) and will not be loaded if the `plain` package option has been enabled (see Section 2.3.2.1):

**url** A package that provides the `\url` macro for the typesetting of links.

**graphicx** A package that provides the `\includegraphics` macro for the typesetting of images.

**paralist** A package that provides the `compactitem`, `compactenum`, and `compactdesc` macros for the typesetting of tight bulleted lists, ordered lists, and definition lists.

**ifthen** A package that provides a concise syntax for the inspection of macro values. It is used in the `witiko/dot` L<sup>A</sup>T<sub>E</sub>X theme (see Section 2.3.2.2), and to provide default token renderer prototypes.

**fancyvrb** A package that provides the `\VerbatimInput` macros for the verbatim inclusion of files containing code.

**csvsimple** A package that provides the `\csvautotabular` macro for typesetting CSV files in the default renderer prototypes for iA Writer content blocks.

**gobble** A package that provides the `\@gobblethree` T<sub>E</sub>X command that is used in the default renderer prototype for citations. The package is included in T<sub>E</sub>XLive ≥ 2016.

**amsmath and amssymb** Packages that provide symbols used for drawing ticked and unticked boxes.

**catchfile** A package that catches the contents of a file and puts it in a macro. It is used in the `witiko/graphicx/http` L<sup>A</sup>T<sub>E</sub>X theme, see Section 2.3.2.2.

**grffile** A package that extends the name processing of package graphics to support a larger range of file names in  $2006 \leq \text{T<sub>E</sub>X Live} \leq 2019$ . Since  $\text{T<sub>E</sub>X Live} \geq 2020$ , the functionality of the package has been integrated in the L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> kernel. It is used in the `witiko/dot` and `witiko/graphicx/http` L<sup>A</sup>T<sub>E</sub>X themes, see Section 2.3.2.2.

**etoolbox** A package that is used to polyfill the general hook management system in the default renderer prototypes for YAML metadata, see Section 3.3.4.6, and also in the default renderer prototype for attribute identifiers.

**expl3** A package that enables the expl3 language from the L<sup>A</sup>T<sub>E</sub>X 3 kernel in  $\text{T<sub>E</sub>X Live} \leq 2019$ . It is used in the default renderer prototypes for links (see Section ??), YAML metadata (see Section 3.3.4.6), and in the implementation of L<sup>A</sup>T<sub>E</sub>X themes (see Section 3.3.2.1).

```
24 \RequirePackage{expl3}
```

#### 1.1.4 ConT<sub>E</sub>Xt Prerequisites

The ConT<sub>E</sub>Xt part of the package requires that either the Mark II or the Mark IV format is loaded, all the plain T<sub>E</sub>X prerequisites (see Section 1.1.2), and the following ConT<sub>E</sub>Xt modules:

**m-database** A module that provides the default token renderer prototype for iA Writer content blocks with the CSV filename extension (see Section 2.2.4).

## 1.2 Feedback

Please use the Markdown project page on GitHub<sup>4</sup> to report bugs and submit feature requests. If you do not want to report a bug or request a feature but are simply in need of assistance, you might want to consider posting your question to the T<sub>E</sub>X-L<sup>A</sup>T<sub>E</sub>X Stack Exchange.<sup>5</sup> community question answering web site under the `markdown` tag.

## 1.3 Acknowledgements

The Lunamark Lua module provides speedy markdown parsing for the package. I would like to thank John Macfarlane, the creator of Lunamark, for releasing Lunamark under a permissive license, which enabled its use in the Markdown package.

---

<sup>4</sup>See <https://github.com/witiko/markdown/issues>.

<sup>5</sup>See <https://tex.stackexchange.com>.

Extensive user documentation for the Markdown package was kindly written by Lian Tze Lim and published by Overleaf.

Funding by the Faculty of Informatics at the Masaryk University in Brno [2] is gratefully acknowledged.

Support for content slicing (Lua options `shiftHeadings` and `slice`) and pipe tables (Lua options `pipeTables` and `tableCaptions`) was graciously sponsored by David Vins and Omedym.

The  $\text{T}_{\text{E}}\text{X}$  implementation of the package draws inspiration from several sources including the source code of  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\epsilon}$ , the minted package by Geoffrey M. Poore, which likewise tackles the issue of interfacing with an external interpreter from  $\text{T}_{\text{E}}\text{X}$ , the filecontents package by Scott Pakin and others.

## 2 Interfaces

This part of the documentation describes the interfaces exposed by the package along with usage notes and examples. It is aimed at the user of the package.

Since neither  $\text{T}_{\text{E}}\text{X}$  nor Lua provide interfaces as a language construct, the separation to interfaces and implementations is a *gentlemen's agreement*. It serves as a means of structuring this documentation and as a promise to the user that if they only access the package through the interface, the future minor versions of the package should remain backwards compatible.

Figure 1 shows the high-level structure of the Markdown package: The translation from markdown to  $\text{T}_{\text{E}}\text{X}$  *token renderers* is exposed by the Lua layer. The plain  $\text{T}_{\text{E}}\text{X}$  layer exposes the conversion capabilities of Lua as  $\text{T}_{\text{E}}\text{X}$  macros. The  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  and  $\text{ConT}_{\text{E}}\text{Xt}$  layers provide syntactic sugar on top of plain  $\text{T}_{\text{E}}\text{X}$  macros. The user can interface with any and all layers.

### 2.1 Lua Interface

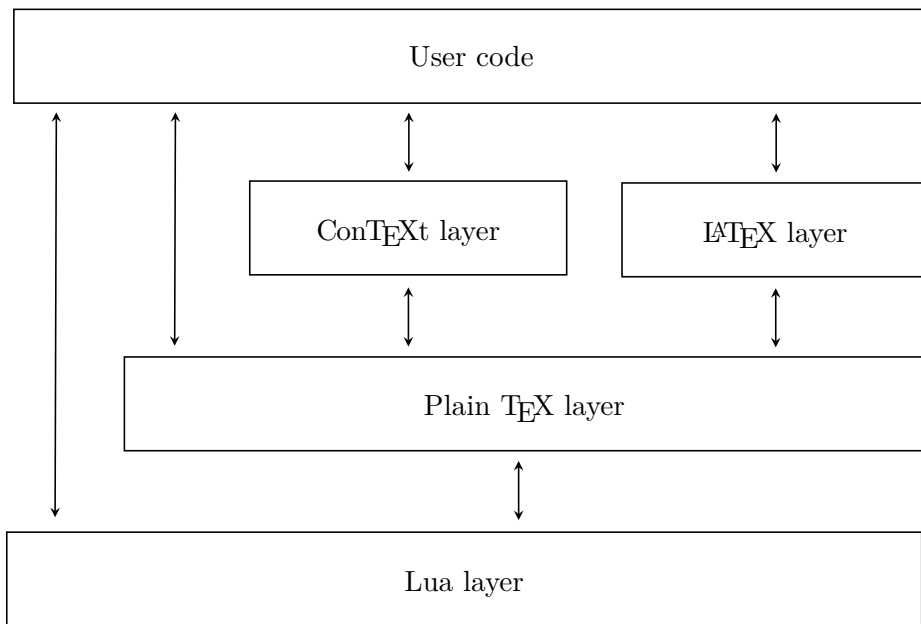
The Lua interface provides the conversion from UTF-8 encoded markdown to plain  $\text{T}_{\text{E}}\text{X}$ . This interface is used by the plain  $\text{T}_{\text{E}}\text{X}$  implementation (see Section 3.2) and will be of interest to the developers of other packages and Lua modules.

The Lua interface is implemented by the `markdown` Lua module.

```
25 local M = {metadata = metadata}
```

#### 2.1.1 Conversion from Markdown to Plain $\text{T}_{\text{E}}\text{X}$

The Lua interface exposes the `new(options)` method. This method creates converter functions that perform the conversion from markdown to plain  $\text{T}_{\text{E}}\text{X}$  according to the table `options` that contains options recognized by the Lua interface. (see Section 2.1.2). The `options` parameter is optional; when unspecified, the behaviour will be the same as if `options` were an empty table.



**Figure 1: A block diagram of the Markdown package**

The following example Lua code converts the markdown string `Hello *world*!` to a TEX output using the default options and prints the TEX output:

```

local md = require("markdown")
local convert = md.new()
print(convert("Hello *world*!"))

```

### 2.1.2 Options

The Lua interface recognizes the following options. When unspecified, the value of a key is taken from the `defaultOptions` table.

```
26 local defaultOptions = {}
```

To enable the enumeration of Lua options, we will maintain the `\g_@@_lua_options_seq` sequence.

```
27 \ExplSyntaxOn
```

```
28 \seq_new:N \g_@@_lua_options_seq
```

To enable the reflection of default Lua options and their types, we will maintain the `\g_@@_default_lua_options_prop` and `\g_@@_lua_option_types_prop` property lists, respectively.

```
29 \prop_new:N \g_@@_lua_option_types_prop
```

```
30 \prop_new:N \g_@@_default_lua_options_prop
```

### 2.1.3 File and Directory Names

`cacheDir`= $\langle path \rangle$  default: .

A path to the directory containing auxiliary cache files. If the last segment of the path does not exist, it will be created by the Lua command-line and plain T<sub>E</sub>X implementations. The Lua implementation expects that the entire path already exists.

When iteratively writing and typesetting a markdown document, the cache files are going to accumulate over time. You are advised to clean the cache directory every now and then, or to set it to a temporary filesystem (such as `/tmp` on UN\*X systems), which gets periodically emptied.

```
31 \seq_put_right:Nn
32   \g_@@_lua_options_seq
33   { cacheDir }
34 \prop_put:Nnn
35   \g_@@_lua_option_types_prop
36   { cacheDir }
37   { string }
38 \prop_put:Nnn
39   \g_@@_default_lua_options_prop
40   { cacheDir }
41   { . }
42 defaultOptions.cacheDir = "."
```

`frozenCacheFileName`= $\langle path \rangle$  default: frozenCache.tex

A path to an output file (frozen cache) that will be created when the `finalizeCache` option is enabled and will contain a mapping between an enumeration of markdown documents and their auxiliary cache files.

The frozen cache makes it possible to later typeset a plain T<sub>E</sub>X document that contains markdown documents without invoking Lua using the `\markdownOptionFrozenCache` plain T<sub>E</sub>X option. As a result, the plain T<sub>E</sub>X document becomes more portable, but further changes in the order and the content of markdown documents will not be reflected.

```
43 \seq_put_right:Nn
44   \g_@@_lua_options_seq
45   { frozenCacheFileName }
46 \prop_put:Nnn
47   \g_@@_lua_option_types_prop
48   { frozenCacheFileName }
49   { string }
```



```

50 \prop_put:Nnn
51   \g_@@_default_lua_options_prop
52   { frozenCacheFileName }
53   { frozenCache.tex }
54 defaultOptions.frozenCacheFileName = "frozenCache.tex"

```

### 2.1.4 Parser Options

`blankBeforeBlockquote=true, false` default: false

- true**      Require a blank line between a paragraph and the following blockquote.
- false**     Do not require a blank line between a paragraph and the following blockquote.

```

55 \seq_put_right:Nn
56   \g_@@_lua_options_seq
57   { blankBeforeBlockquote }
58 \prop_put:Nnn
59   \g_@@_lua_option_types_prop
60   { blankBeforeBlockquote }
61   { boolean }
62 \prop_put:Nnn
63   \g_@@_default_lua_options_prop
64   { blankBeforeBlockquote }
65   { false }
66 defaultOptions.blankBeforeBlockquote = false

```

`blankBeforeCodeFence=true, false` default: false

- true**      Require a blank line between a paragraph and the following fenced code block.
- false**     Do not require a blank line between a paragraph and the following fenced code block.

```

67 \seq_put_right:Nn
68   \g_@@_lua_options_seq
69   { blankBeforeCodeFence }
70 \prop_put:Nnn
71   \g_@@_lua_option_types_prop
72   { blankBeforeCodeFence }
73   { boolean }
74 \prop_put:Nnn
75   \g_@@_default_lua_options_prop
76   { blankBeforeCodeFence }
77   { false }

```

```
78 defaultOptions.blankBeforeCodeFence = false
```

`blankBeforeHeading=true, false` default: false

- `true`      Require a blank line between a paragraph and the following header.
- `false`     Do not require a blank line between a paragraph and the following header.

```
79 \seq_put_right:Nn
80 \g_@@_lua_options_seq
81 { blankBeforeHeading }
82 \prop_put:Nnn
83 \g_@@_lua_option_types_prop
84 { blankBeforeHeading }
85 { boolean }
86 \prop_put:Nnn
87 \g_@@_default_lua_options_prop
88 { blankBeforeHeading }
89 { false }
90 defaultOptions.blankBeforeHeading = false
```

`breakableBlockquotes=true, false` default: false

- `true`      A blank line separates block quotes.
- `false`     Blank lines in the middle of a block quote are ignored.

```
91 \seq_put_right:Nn
92 \g_@@_lua_options_seq
93 { breakableBlockquotes }
94 \prop_put:Nnn
95 \g_@@_lua_option_types_prop
96 { breakableBlockquotes }
97 { boolean }
98 \prop_put:Nnn
99 \g_@@_default_lua_options_prop
100 { breakableBlockquotes }
101 { false }
102 defaultOptions.breakableBlockquotes = false
```

`citationNbsps=true, false`

default: `false`

- `true` Replace regular spaces with non-breaking spaces inside the prenotes and postnotes of citations produced via the pandoc citation syntax extension.
- `false` Do not replace regular spaces with non-breaking spaces inside the prenotes and postnotes of citations produced via the pandoc citation syntax extension.

```
103 \seq_put_right:Nn
104   \g_@@_lua_options_seq
105   { citationNbsps }
106 \prop_put:Nnn
107   \g_@@_lua_option_types_prop
108   { citationNbsps }
109   { boolean }
110 \prop_put:Nnn
111   \g_@@_default_lua_options_prop
112   { citationNbsps }
113   { true }
114 defaultOptions.citationNbsps = true
```

`citations=true, false`

default: `false`

- `true` Enable the pandoc citation syntax extension:

Here is a simple parenthetical citation [`@doe99`] and here is a string of several [`see @doe99, pp. 33-35; also @smith04, chap. 1`].

A parenthetical citation can have a [`prenote @doe99`] and a [`@smith04 postnote`]. The name of the author can be suppressed by inserting a dash before the name of an author as follows [`-@smith04`].

Here is a simple text citation `@doe99` and here is a string of several `@doe99` [`pp. 33-35; also @smith04, chap. 1`]. Here is one with the name of the author suppressed `-@doe99`.

- `false` Disable the pandoc citation syntax extension.

```
115 \seq_put_right:Nn
```

```

116 \g_@@_lua_options_seq
117 { citations }
118 \prop_put:Nnn
119 \g_@@_lua_option_types_prop
120 { citations }
121 { boolean }
122 \prop_put:Nnn
123 \g_@@_default_lua_options_prop
124 { citations }
125 { false }
126 defaultOptions.citations = false

```

`codeSpans=true, false`

default: true

**true** Enable the code span syntax:

```

Use the printf() function.
There is a literal backtick (``) here.

```

**false** Disable the code span syntax. This allows you to easily use the quotation mark ligatures in texts that do not contain code spans:

```

This is a quote.

```

```

127 \seq_put_right:Nn
128 \g_@@_lua_options_seq
129 { codeSpans }
130 \prop_put:Nnn
131 \g_@@_lua_option_types_prop
132 { codeSpans }
133 { boolean }
134 \prop_put:Nnn
135 \g_@@_default_lua_options_prop
136 { codeSpans }
137 { true }
138 defaultOptions.codeSpans = true

```

`contentBlocks=true, false`

default: false

**true** Enable the iA Writer content blocks syntax extension [3]:

```

http://example.com/minard.jpg (Napoleon's
disastrous Russian campaign of 1812)
/Flowchart.png "Engineering Flowchart"

```

```
/Savings Account.csv 'Recent Transactions'  
/Example.swift  
/Lorem Ipsum.txt
```

`false` Disable the iA Writer content blocks syntax extension.

```
139 \seq_put_right:Nn  
140 \g_@@_lua_options_seq  
141 { contentBlocks }  
142 \prop_put:Nnn  
143 \g_@@_lua_option_types_prop  
144 { contentBlocks }  
145 { boolean }  
146 \prop_put:Nnn  
147 \g_@@_default_lua_options_prop  
148 { contentBlocks }  
149 { false }  
  
150 defaultOptions.contentBlocks = false
```

`contentBlocksLanguageMap=<filename>`  
default: markdown-languages.json

The filename of the JSON file that maps filename extensions to programming language names in the iA Writer content blocks. See Section 2.2.3.11 for more information.

```
151 \seq_put_right:Nn  
152 \g_@@_lua_options_seq  
153 { contentBlocksLanguageMap }  
154 \prop_put:Nnn  
155 \g_@@_lua_option_types_prop  
156 { contentBlocksLanguageMap }  
157 { string }  
158 \prop_put:Nnn  
159 \g_@@_default_lua_options_prop  
160 { contentBlocksLanguageMap }  
161 { markdown-languages.json }  
  
162 defaultOptions.contentBlocksLanguageMap = "markdown-languages.json"
```

`definitionLists=true, false`

default: false

`true` Enable the pandoc definition list syntax extension:

```
Term 1
```

```

: Definition 1

Term 2 with inline markup

: Definition 2

    { some code, part of Definition 2 }

Third paragraph of definition 2.

```

**false** Disable the pandoc definition list syntax extension.

```

163 \seq_put_right:Nn
164   \g_@@_lua_options_seq
165   { definitionLists }
166 \prop_put:Nnn
167   \g_@@_lua_option_types_prop
168   { definitionLists }
169   { boolean }
170 \prop_put:Nnn
171   \g_@@_default_lua_options_prop
172   { definitionLists }
173   { false }
174 defaultOptions.definitionLists = false

```

**eagerCache=true, false**

default: true

**true** Converted markdown documents will be cached in `cacheDir`. This can be useful for post-processing the converted documents and for recovering historical versions of the documents from the cache. However, it also produces a large number of auxiliary files on the disk and obscures the output of the Lua command-line interface when it is used for plumbing. This behavior will always be used if the `finalizeCache` option is enabled.

**false** Converted markdown documents will not be cached. This decreases the number of auxiliary files that we produce and makes it easier to use the Lua command-line interface for plumbing.

This behavior will only be used when the `finalizeCache` option is disabled. Furthermore, this behavior is planned to be the new default in the next major release of the Markdown package.

```

175 \seq_put_right:Nn
176   \g_@@_lua_options_seq
177   { eagerCache }
178 \prop_put:Nnn
179   \g_@@_lua_option_types_prop
180   { eagerCache }
181   { boolean }
182 \prop_put:Nnn
183   \g_@@_default_lua_options_prop
184   { eagerCache }
185   { true }
186 defaultOptions.eagerCache = true

```

`expectJekyllData=true, false`

default: `false`

**false** When the `jekyllData` option is enabled, then a markdown document may begin with YAML metadata if and only if the metadata begin with the end-of-directives marker (`---`) and they end with either the end-of-directives or the end-of-document marker (`...`):

```

\documentclass{article}
\usepackage[jekyllData]{markdown}
\begin{document}
\begin{markdown}
---
- this
- is
- YAML
...
- followed
- by
- Markdown
\end{markdown}
\begin{markdown}
- this
- is
- Markdown
\end{markdown}
\end{document}

```

**true** When the `jekyllData` option is enabled, then a markdown document may begin directly with YAML metadata and may contain nothing but YAML metadata.

```

\documentclass{article}
\usepackage[jekyllData, expectJekyllData]{markdown}
\begin{document}
\begin{markdown}
- this
- is
- YAML
...
- followed
- by
- Markdown
\end{markdown}
\begin{markdown}
- this
- is
- YAML
\end{markdown}
\end{document}

```

```

187 \seq_put_right:Nn
188 \g_@@_lua_options_seq
189 { expectJekyllData }
190 \prop_put:Nnn
191 \g_@@_lua_option_types_prop
192 { expectJekyllData }
193 { boolean }
194 \prop_put:Nnn
195 \g_@@_default_lua_options_prop
196 { expectJekyllData }
197 { false }
198 defaultOptions.expectJekyllData = false

```

`fencedCode=true, false`

default: false

`true`

Enable the commonmark fenced code block extension:

```

~~~ js
if (a > 3) {
    moveShip(5 * gravity, DOWN);
}
~~~~~

```



```


... html
<pre>
  <code>
    // Some comments
    line 1 of code
    line 2 of code
    line 3 of code
  </code>
</pre>
...


```

`false`      Disable the commonmark fenced code block extension.

```

199 \seq_put_right:Nn
200   \g_@@_lua_options_seq
201   { fencedCode }
202 \prop_put:Nnn
203   \g_@@_lua_option_types_prop
204   { fencedCode }
205   { boolean }
206 \prop_put:Nnn
207   \g_@@_default_lua_options_prop
208   { fencedCode }
209   { false }
210 defaultOptions.fencedCode = false

```

`finalizeCache=true, false`

default: `false`

Whether an output file specified with the `frozenCacheFileName` option (frozen cache) that contains a mapping between an enumeration of markdown documents and their auxiliary cache files will be created.

The frozen cache makes it possible to later typeset a plain T<sub>E</sub>X document that contains markdown documents without invoking Lua using the `\markdownOptionFrozenCache` plain T<sub>E</sub>X option. As a result, the plain T<sub>E</sub>X document becomes more portable, but further changes in the order and the content of markdown documents will not be reflected.

```

211 \seq_put_right:Nn
212   \g_@@_lua_options_seq
213   { finalizeCache }
214 \prop_put:Nnn
215   \g_@@_lua_option_types_prop
216   { finalizeCache }
217   { boolean }

```

```

218 \prop_put:Nnn
219   \g_@@_default_lua_options_prop
220   { finalizeCache }
221   { false }

222 defaultOptions.finalizeCache = false

```

`footnotes=true, false`

default: false

**true** Enable the pandoc footnote syntax extension:

```

Here is a footnote reference, [^1] and another. [^longnote]

[^1]: Here is the footnote.

[^longnote]: Here's one with multiple blocks.

    Subsequent paragraphs are indented to show that they
    belong to the previous footnote.

        { some.code }

    The whole paragraph can be indented, or just the
    first line. In this way, multi-paragraph footnotes
    work like multi-paragraph list items.

This paragraph won't be part of the note, because it
isn't indented.

```

**false** Disable the pandoc footnote syntax extension.

```

223 \seq_put_right:Nn
224   \g_@@_lua_options_seq
225   { footnotes }
226 \prop_put:Nnn
227   \g_@@_lua_option_types_prop
228   { footnotes }
229   { boolean }
230 \prop_put:Nnn
231   \g_@@_default_lua_options_prop
232   { footnotes }
233   { false }

234 defaultOptions.footnotes = false

```

`frozenCacheCounter`= $\langle number \rangle$

default: 0

The number of the current markdown document that will be stored in an output file (frozen cache) when the `finalizeCache` is enabled. When the document number is 0, then a new frozen cache will be created. Otherwise, the frozen cache will be appended.

Each frozen cache entry will define a T<sub>E</sub>X macro `\markdownFrozenCache` $\langle number \rangle$  that will typeset markdown document number  $\langle number \rangle$ .

```
235 \seq_put_right:Nn
236   \g_@@_lua_options_seq
237   { frozenCacheCounter }
238 \prop_put:Nnn
239   \g_@@_lua_option_types_prop
240   { frozenCacheCounter }
241   { counter }
242 \prop_put:Nnn
243   \g_@@_default_lua_options_prop
244   { frozenCacheCounter }
245   { 0 }
246 defaultOptions.frozenCacheCounter = 0
```

`hardLineBreaks`=true, false

default: false

**true** Interpret all newlines within a paragraph as hard line breaks instead of spaces.

**false** Interpret all newlines within a paragraph as spaces.

```
247 \seq_put_right:Nn
248   \g_@@_lua_options_seq
249   { hardLineBreaks }
250 \prop_put:Nnn
251   \g_@@_lua_option_types_prop
252   { hardLineBreaks }
253   { boolean }
254 \prop_put:Nnn
255   \g_@@_default_lua_options_prop
256   { hardLineBreaks }
257   { false }
258 defaultOptions.hardLineBreaks = false
```

`hashEnumerators=true, false`

default: `false`

`true` Enable the use of hash symbols (#) as ordered item list markers:

```
#. Bird
#. McHale
#. Parish
```

`false` Disable the use of hash symbols (#) as ordered item list markers.

```
259 \seq_put_right:Nn
260   \g_@@_lua_options_seq
261   { hashEnumerators }
262 \prop_put:Nnn
263   \g_@@_lua_option_types_prop
264   { hashEnumerators }
265   { boolean }
266 \prop_put:Nnn
267   \g_@@_default_lua_options_prop
268   { hashEnumerators }
269   { false }
270 defaultOptions.hashEnumerators = false
```

`headerAttributes=true, false`

default: `false`

`true` Enable the assignment of HTML attributes to headings:

```
# My first heading {#foo}

## My second heading ##   {#bar .baz}

Yet another heading   {key=value}
=====
```

These HTML attributes have currently no effect other than enabling content slicing, see the `slice` option.

`false` Disable the assignment of HTML attributes to headings.

```
271 \seq_put_right:Nn
272   \g_@@_lua_options_seq
273   { headerAttributes }
274 \prop_put:Nnn
275   \g_@@_lua_option_types_prop
276   { headerAttributes }
```

```

277 { boolean }
278 \prop_put:Nnn
279 \g_@@_default_lua_options_prop
280 { headerAttributes }
281 { false }

282 defaultOptions.headerAttributes = false

```

`html=true, false` default: `false`

- true** Enable the recognition of inline HTML tags, block HTML elements, HTML comments, HTML instructions, and entities in the input. Inline HTML tags, block HTML elements and HTML comments will be rendered, HTML instructions will be ignored, and HTML entities will be replaced with the corresponding Unicode codepoints.
- false** Disable the recognition of HTML markup. Any HTML markup in the input will be rendered as plain text.

```

283 \seq_put_right:Nn
284 \g_@@_lua_options_seq
285 { html }
286 \prop_put:Nnn
287 \g_@@_lua_option_types_prop
288 { html }
289 { boolean }
290 \prop_put:Nnn
291 \g_@@_default_lua_options_prop
292 { html }
293 { false }

294 defaultOptions.html = false

```

`hybrid=true, false` default: `false`

- true** Disable the escaping of special plain  $\TeX$  characters, which makes it possible to intersperse your markdown markup with  $\TeX$  code. The intended usage is in documents prepared manually by a human author. In such documents, it can often be desirable to mix  $\TeX$  and markdown markup freely.
- false** Enable the escaping of special plain  $\TeX$  characters outside verbatim environments, so that they are not interpreted by  $\TeX$ . This is encouraged when typesetting automatically generated content or markdown documents that were not prepared with this package in mind.

```

295 \seq_put_right:Nn
296   \g_@@_lua_options_seq
297   { hybrid }
298 \prop_put:Nnn
299   \g_@@_lua_option_types_prop
300   { hybrid }
301   { boolean }
302 \prop_put:Nnn
303   \g_@@_default_lua_options_prop
304   { hybrid }
305   { false }
306 defaultOptions.hybrid = false

```

`inlineFootnotes=true, false`

default: false

**true** Enable the pandoc inline footnote syntax extension:

```

Here is an inline note.^[Inlines notes are easier to
write, since you don't have to pick an identifier and
move down to type the note.]

```

**false** Disable the pandoc inline footnote syntax extension.

```

307 \seq_put_right:Nn
308   \g_@@_lua_options_seq
309   { inlineFootnotes }
310 \prop_put:Nnn
311   \g_@@_lua_option_types_prop
312   { inlineFootnotes }
313   { boolean }
314 \prop_put:Nnn
315   \g_@@_default_lua_options_prop
316   { inlineFootnotes }
317   { false }
318 defaultOptions.inlineFootnotes = false

```

`jeekyllData=true, false`

default: false

**true** Enable the Pandoc `yml_metadata_block` syntax extension for entering metadata in YAML:

```

---
title: 'This is the title: it contains a colon'
author:

```

```

- Author One
- Author Two
keywords: [nothing, nothingness]
abstract: |
  This is the abstract.

  It consists of two paragraphs.
---
```

`false` Disable the Pandoc `yaml_metadata_block` syntax extension for entering metadata in YAML.

```

319 \seq_put_right:Nn
320   \g_@@_lua_options_seq
321   { jekyllData }
322 \prop_put:Nnn
323   \g_@@_lua_option_types_prop
324   { jekyllData }
325   { boolean }
326 \prop_put:Nnn
327   \g_@@_default_lua_options_prop
328   { jekyllData }
329   { false }
330 defaultOptions.jekyllData = false
```

`pipeTables=true, false`

default: `false`

`true` Enable the PHP Markdown table syntax extension:

Right	Left	Default	Center
12	12	12	12
123	123	123	123
1	1	1	1

`false` Disable the PHP Markdown table syntax extension.

```

331 \seq_put_right:Nn
332   \g_@@_lua_options_seq
333   { pipeTables }
334 \prop_put:Nnn
335   \g_@@_lua_option_types_prop
336   { pipeTables }
337   { boolean }
```

```

338 \prop_put:Nnn
339   \g_@@_default_lua_options_prop
340   { pipeTables }
341   { false }
342 defaultOptions.pipeTables = false

```

`preserveTabs=true, false`

default: false

- `true`      Preserve tabs in code block and fenced code blocks.
- `false`     Convert any tabs in the input to spaces.

```

343 \seq_put_right:Nn
344   \g_@@_lua_options_seq
345   { preserveTabs }
346 \prop_put:Nnn
347   \g_@@_lua_option_types_prop
348   { preserveTabs }
349   { boolean }
350 \prop_put:Nnn
351   \g_@@_default_lua_options_prop
352   { preserveTabs }
353   { false }
354 defaultOptions.preserveTabs = false

```

`relativeReferences=true, false`

default: false

- `true`      Enable relative references<sup>6</sup> in autolinks:

```

I 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!

```

- `false`     Disable relative references in autolinks.

```

355 \seq_put_right:Nn
356   \g_@@_lua_options_seq
357   { relativeReferences }
358 \prop_put:Nnn

```

---

<sup>6</sup>See <https://datatracker.ietf.org/doc/html/rfc3986#section-4.2>.



```

359 \g_@@_lua_option_types_prop
360 { relativeReferences }
361 { boolean }
362 \prop_put:Nnn
363 \g_@@_default_lua_options_prop
364 { relativeReferences }
365 { false }
366 defaultOptions.relativeReferences = false

```

`shiftHeadings`= $\langle shift\ amount \rangle$  default: 0

All headings will be shifted by  $\langle shift\ amount \rangle$ , which can be both positive and negative. Headings will not be shifted beyond level 6 or below level 1. Instead, those headings will be shifted to level 6, when  $\langle shift\ amount \rangle$  is positive, and to level 1, when  $\langle shift\ amount \rangle$  is negative.

```

367 \seq_put_right:Nn
368 \g_@@_lua_options_seq
369 { shiftHeadings }
370 \prop_put:Nnn
371 \g_@@_lua_option_types_prop
372 { shiftHeadings }
373 { number }
374 \prop_put:Nnn
375 \g_@@_default_lua_options_prop
376 { shiftHeadings }
377 { 0 }
378 defaultOptions.shiftHeadings = 0

```

`slice`= $\langle the\ beginning\ and\ the\ end\ of\ a\ slice \rangle$  default:  $\wedge$   $\$$

Two space-separated selectors that specify the slice of a document that will be processed, whereas the remainder of the document will be ignored. The following selectors are recognized:

- The circumflex ( $\wedge$ ) selects the beginning of a document.
- The dollar sign ( $\$$ ) selects the end of a document.
- $\wedge\langle identifier \rangle$  selects the beginning of a section with the HTML attribute  $\#\langle identifier \rangle$  (see the `headerAttributes` option).
- $\$\langle identifier \rangle$  selects the end of a section with the HTML attribute  $\#\langle identifier \rangle$ .
- $\langle identifier \rangle$  corresponds to  $\wedge\langle identifier \rangle$  for the first selector and to  $\$\langle identifier \rangle$  for the second selector.

Specifying only a single selector,  $\langle identifier \rangle$ , is equivalent to specifying the two selectors  $\langle identifier \rangle\langle identifier \rangle$ , which is equivalent to  $\wedge\langle identifier \rangle\$\langle identifier \rangle$ , i.e. the entire section with the HTML attribute  $\#\langle identifier \rangle$  will be selected.

```

379 \seq_put_right:Nn
380   \g_@@_lua_options_seq
381   { slice }
382 \prop_put:Nnn
383   \g_@@_lua_option_types_prop
384   { slice }
385   { string }
386 \prop_put:Nnn
387   \g_@@_default_lua_options_prop
388   { slice }
389   { ^~$ }
390 defaultOptions.slice = "^ $"

```

`smartEllipses=true, false` default: false

**true** Convert any ellipses in the input to the `\markdownRendererEllipsis` T<sub>E</sub>X macro.

**false** Preserve all ellipses in the input.

```

391 \seq_put_right:Nn
392   \g_@@_lua_options_seq
393   { smartEllipses }
394 \prop_put:Nnn
395   \g_@@_lua_option_types_prop
396   { smartEllipses }
397   { boolean }
398 \prop_put:Nnn
399   \g_@@_default_lua_options_prop
400   { smartEllipses }
401   { false }
402 defaultOptions.smartEllipses = false

```

`startNumber=true, false` default: true

**true** Make the number in the first item of an ordered lists significant. The item numbers will be passed to the `\markdownRenderer01ItemWithNumber` T<sub>E</sub>X macro.

**false** Ignore the numbers in the ordered list items. Each item will only produce a `\markdownRenderer01Item` T<sub>E</sub>X macro.

```

403 \seq_put_right:Nn
404   \g_@@_lua_options_seq
405   { startNumber }

```

```

406 \prop_put:Nnn
407   \g_@@_lua_option_types_prop
408   { startNumber }
409   { boolean }
410 \prop_put:Nnn
411   \g_@@_default_lua_options_prop
412   { startNumber }
413   { true }
414 defaultOptions.startNumber = true

```

`stripIndent=true, false`

default: `false`

**true** Strip the minimal indentation of non-blank lines from all lines in a markdown document. Requires that the `preserveTabs` Lua option is `false`:

```

\documentclass{article}
\usepackage[stripIndent]{markdown}
\begin{document}
  \begin{markdown}
    Hello *world*!
  \end{markdown}
\end{document}

```

**false** Do not strip any indentation from the lines in a markdown document.

```

415 \seq_put_right:Nn
416   \g_@@_lua_options_seq
417   { stripIndent }
418 \prop_put:Nnn
419   \g_@@_lua_option_types_prop
420   { stripIndent }
421   { boolean }
422 \prop_put:Nnn
423   \g_@@_default_lua_options_prop
424   { stripIndent }
425   { false }
426 defaultOptions.stripIndent = false

```

`tableCaptions=true, false`

default: `false`

**true** Enable the Pandoc `table_captions` syntax extension for pipe tables (see the `pipeTables` option).

Right	Left	Default	Center
12	12	12	12
123	123	123	123
1	1	1	1

: Demonstration of pipe table syntax.

**false** Disable the Pandoc `table_captions` syntax extension.

```

427 \seq_put_right:Nn
428   \g_@@_lua_options_seq
429   { tableCaptions }
430 \prop_put:Nnn
431   \g_@@_lua_option_types_prop
432   { tableCaptions }
433   { boolean }
434 \prop_put:Nnn
435   \g_@@_default_lua_options_prop
436   { tableCaptions }
437   { false }
438 defaultOptions.tableCaptions = false

```

`taskLists=true, false` default: false

**true** Enable the Pandoc `task_lists` syntax extension.

- [ ]	an unticked task list item
- [/]	a half-checked task list item
- [X]	a ticked task list item

**false** Disable the Pandoc `task_lists` syntax extension.

```

439 \seq_put_right:Nn
440   \g_@@_lua_options_seq
441   { taskLists }
442 \prop_put:Nnn
443   \g_@@_lua_option_types_prop
444   { taskLists }
445   { boolean }
446 \prop_put:Nnn
447   \g_@@_default_lua_options_prop
448   { taskLists }
449   { false }
450 defaultOptions.taskLists = false

```

`texComments=true, false`

default: false

`true` Strip T<sub>E</sub>X-style comments.

```
\documentclass{article}
\usepackage[texComments]{markdown}
\begin{document}
\begin{markdown}
Hello *world*!
\end{markdown}
\end{document}
```

Always enabled when `hybrid` is enabled.

`false` Do not strip T<sub>E</sub>X-style comments.

```
451 \seq_put_right:Nn
452   \g_@@_lua_options_seq
453   { texComments }
454 \prop_put:Nnn
455   \g_@@_lua_option_types_prop
456   { texComments }
457   { boolean }
458 \prop_put:Nnn
459   \g_@@_default_lua_options_prop
460   { texComments }
461   { false }
462 defaultOptions.texComments = false
```

`tightLists=true, false`

default: true

`true` Unordered and ordered Lists whose items do not consist of multiple paragraphs will be considered *tight*. Tight lists will produce tight renders that may produce different output than lists that are not tight:

```
- This is
- a tight
- unordered list.

- This is

  not a tight

- unordered list.
```

**false** Unordered and ordered lists whose items consist of multiple paragraphs will be treated the same way as lists that consist of multiple paragraphs.

```
463 \seq_put_right:Nn
464   \g_@@_lua_options_seq
465   { tightLists }
466 \prop_put:Nnn
467   \g_@@_lua_option_types_prop
468   { tightLists }
469   { boolean }
470 \prop_put:Nnn
471   \g_@@_default_lua_options_prop
472   { tightLists }
473   { true }
474 defaultOptions.tightLists = true
```

**underscores=true, false**

default: true

**true** Both underscores and asterisks can be used to denote emphasis and strong emphasis:

```
*single asterisks*
_single underscores_
**double asterisks**
__double underscores__
```

**false** Only asterisks can be used to denote emphasis and strong emphasis. This makes it easy to write math with the **hybrid** option without the need to constantly escape subscripts.

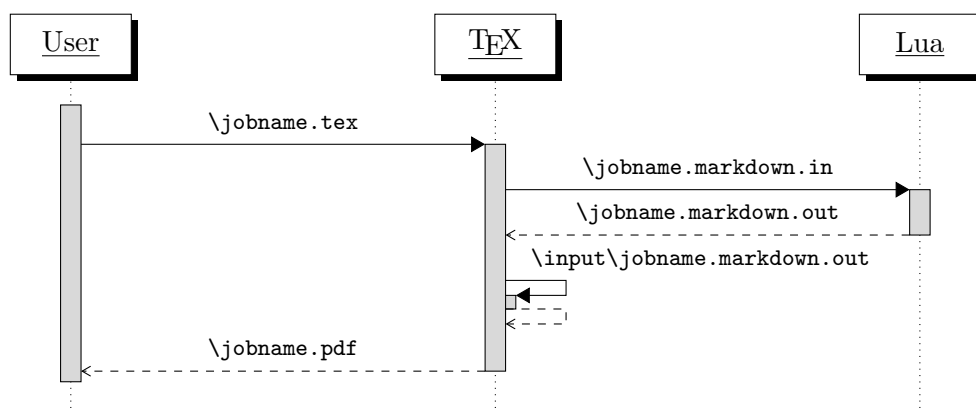
```
475 \seq_put_right:Nn
476   \g_@@_lua_options_seq
477   { underscores }
478 \prop_put:Nnn
479   \g_@@_lua_option_types_prop
480   { underscores }
481   { boolean }
482 \prop_put:Nnn
483   \g_@@_default_lua_options_prop
484   { underscores }
485   { true }
486 \ExplSyntaxOff
487 defaultOptions.underscores = true
```

### 2.1.5 Command-Line Interface

The high-level operation of the Markdown package involves the communication between several programming layers: the plain T<sub>E</sub>X layer hands markdown documents to the Lua layer. Lua converts the documents to T<sub>E</sub>X, and hands the converted documents back to plain T<sub>E</sub>X layer for typesetting, see Figure 2.

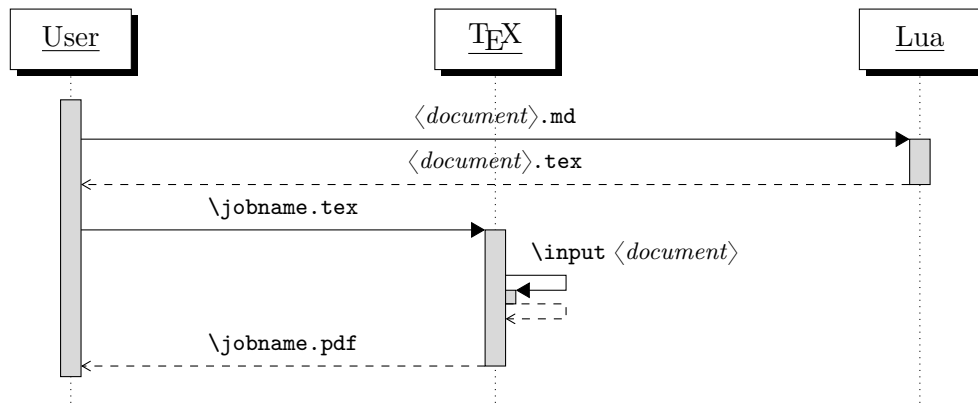
This procedure has the advantage of being fully automated. However, it also has several important disadvantages: The converted T<sub>E</sub>X documents are cached on the file system, taking up increasing amount of space. Unless the T<sub>E</sub>X engine includes a Lua interpreter, the package also requires shell access, which opens the door for a malicious actor to access the system. Last, but not least, the complexity of the procedure impedes debugging.

A solution to the above problems is to decouple the conversion from the typesetting. For this reason, a command-line Lua interface for converting a markdown document to T<sub>E</sub>X is also provided, see Figure 3.



**Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the T<sub>E</sub>X interface**

```
488
489 HELP_STRING = [[
490 Usage: texlua ]] .. arg[0] .. [[ [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
491 where OPTIONS are documented in the Lua interface section of the
492 technical Markdown package documentation.
493
494 When OUTPUT_FILE is unspecified, the result of the conversion will be
495 written to the standard output. When INPUT_FILE is also unspecified, the
496 result of the conversion will be read from the standard input.
497
498 Report bugs to: witiko@mail.muni.cz
499 Markdown package home page: <https://github.com/witiko/markdown>]]
500
```



**Figure 3: A sequence diagram of the Markdown package typesetting a markdown document using the Lua command-line interface**

```

501 VERSION_STRING = [[
502 markdown-cli.lua (Markdown) ]] .. metadata.version .. [[
503
504 Copyright (C) ]] .. table.concat(metadata.copyright,
505                                "\nCopyright (C) ") .. [[
506
507 License: ]] .. metadata.license
508
509 local function warn(s)
510   io.stderr:write("Warning: " .. s .. "\n") end
511
512 local function error(s)
513   io.stderr:write("Error: " .. s .. "\n")
514   os.exit(1) end
515
516 local process_options = true
517 local options = {}
518 local input_filename
519 local output_filename
520 for i = 1, #arg do
521   if process_options then

```

After the optional `--` argument has been specified, the remaining arguments are assumed to be input and output filenames. This argument is optional, but encouraged, because it helps resolve ambiguities when deciding whether an option or a filename has been specified.

```

522     if arg[i] == "--" then
523       process_options = false
524       goto continue

```



Unless the `--` argument has been specified before, an argument containing the equals sign (=) is assumed to be an option specification in a  $\langle key \rangle = \langle value \rangle$  format. The available options are listed in Section 2.1.2.

```
525     elseif arg[i]:match("=") then
526         key, value = arg[i]:match("(.)=(.*)")
```

The `defaultOptions` table is consulted to identify whether  $\langle value \rangle$  should be parsed as a string or as a boolean.

```
527         default_type = type(defaultOptions[key])
528         if default_type == "boolean" then
529             options[key] = (value == "true")
530         elseif default_type == "number" then
531             options[key] = tonumber(value)
532         else
533             if default_type ~= "string" then
534                 if default_type == "nil" then
535                     warn('Option "' .. key .. '" not recognized.')
536                 else
537                     warn('Option "' .. key .. '" type not recognized, please file ' ..
538                         'a report to the package maintainer.')
539                 end
540                 warn('Parsing the ' .. 'value "' .. value .. '" of option "' ..
541                     key .. '" as a string.')
542             end
543             options[key] = value
544         end
545         goto continue
```

Unless the `--` argument has been specified before, an argument `--help`, or `-h` causes a brief documentation for how to invoke the program to be printed to the standard output.

```
546     elseif arg[i] == "--help" or arg[i] == "-h" then
547         print(HELP_STRING)
548         os.exit()
```

Unless the `--` argument has been specified before, an argument `--version`, or `-v` causes the program to print information about its name, version, origin and legal status, all on standard output.

```
549     elseif arg[i] == "--version" or arg[i] == "-v" then
550         print(VERSION_STRING)
551         os.exit()
552     end
553 end
```

The first argument that matches none of the above patterns is assumed to be the input filename. The input filename should correspond to the Markdown document that is going to be converted to a TeX document.

```

554   if input_filename == nil then
555       input_filename = arg[i]

```

The first argument that matches none of the above patterns is assumed to be the output filename. The output filename should correspond to the  $\text{\TeX}$  document that will result from the conversion.

```

556   elseif output_filename == nil then
557       output_filename = arg[i]
558   else
559       error('Unexpected argument: "' .. arg[i] .. '".')
560   end
561   ::continue::
562 end

```

The command-line Lua interface is implemented by the `markdown-cli.lua` file that can be invoked from the command line as follows:

```

texlua /path/to/markdown-cli.lua cacheDir=. -- hello.md hello.tex

```

to convert the Markdown document `hello.md` to a  $\text{\TeX}$  document `hello.tex`. After the Markdown package for our  $\text{\TeX}$  format has been loaded, the converted document can be typeset as follows:

```



```

## 2.2 Plain $\text{\TeX}$ Interface

The plain  $\text{\TeX}$  interface provides macros for the typesetting of markdown input from within plain  $\text{\TeX}$ , for setting the Lua interface options (see Section 2.1.2) used during the conversion from markdown to plain  $\text{\TeX}$  and for changing the way markdown the tokens are rendered.

```

563 \def\markdownLastModified{((LASTMODIFIED))}%
564 \def\markdownVersion{((VERSION))}%

```

The plain  $\text{\TeX}$  interface is implemented by the `markdown.tex` file that can be loaded as follows:

```



```

It is expected that the special plain  $\text{\TeX}$  characters have the expected category codes, when `\inputting` the file.

### 2.2.1 Typesetting Markdown

The interface exposes the `\markdownBegin`, `\markdownEnd`, and `\markdownInput` macros.

The `\markdownBegin` macro marks the beginning of a markdown document fragment and the `\markdownEnd` macro marks its end.

```
565 \let\markdownBegin\relax
566 \let\markdownEnd\relax
```

You may prepend your own code to the `\markdownBegin` macro and redefine the `\markdownEnd` macro to produce special effects before and after the markdown block.

There are several limitations to the macros you need to be aware of. The first limitation concerns the `\markdownEnd` macro, which must be visible directly from the input line buffer (it may not be produced as a result of input expansion). Otherwise, it will not be recognized as the end of the markdown string. As a corollary, the `\markdownEnd` string may not appear anywhere inside the markdown input.

Another limitation concerns spaces at the right end of an input line. In markdown, these are used to produce a forced line break. However, any such spaces are removed before the lines enter the input buffer of T<sub>E</sub>X [4, p. 46]. As a corollary, the `\markdownBegin` macro also ignores them.

The `\markdownBegin` and `\markdownEnd` macros will also consume the rest of the lines at which they appear. In the following example plain T<sub>E</sub>X code, the characters `c`, `e`, and `f` will not appear in the output.

```
\input markdown
a
b \markdownBegin c
d
e \markdownEnd f
g
\bye
```

Note that you may also not nest the `\markdownBegin` and `\markdownEnd` macros.

The following example plain T<sub>E</sub>X code showcases the usage of the `\markdownBegin` and `\markdownEnd` macros:

```
\input markdown
\markdownBegin
_Hello_ **world** ...
\markdownEnd
\bye
```

The `\markdownInput` macro accepts a single parameter containing the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain TeX.

```
567 \let\markdownInput\relax
```

This macro is not subject to the abovelisted limitations of the `\markdownBegin` and `\markdownEnd` macros.

The following example plain TeX code showcases the usage of the `\markdownInput` macro:

```
\input markdown
\markdownInput{hello.md}
\bye
```

## 2.2.2 Options

The plain TeX options are represented by TeX commands. Some of them map directly to the options recognized by the Lua interface (see Section 2.1.2), while some of them are specific to the plain TeX interface.

**2.2.2.1 Finalizing and Freezing the Cache** The `\markdownOptionFinalizeCache` option corresponds to the Lua interface `finalizeCache` option, which creates an output file `\markdownOptionFrozenCacheFileName` (frozen cache) that contains a mapping between an enumeration of the markdown documents in the plain TeX document and their auxiliary files cached in the `cacheDir` directory.

```
568 \let\markdownOptionFinalizeCache\undefined
```

The `\markdownOptionFrozenCache` option uses the mapping previously created by the `\markdownOptionFinalizeCache` option, and uses it to typeset the plain TeX document without invoking Lua. As a result, the plain TeX document becomes more portable, but further changes in the order and the content of markdown documents will not be reflected. It defaults to `false`.

The standard usage of the above two options is as follows:

1. Remove the `cacheDir` cache directory with stale auxiliary cache files.
2. Enable the `\markdownOptionFinalizeCache` option.
4. Typeset the plain TeX document to populate and finalize the cache.
5. Enable the `\markdownOptionFrozenCache` option.
6. Publish the source code of the plain TeX document and the `cacheDir` directory.

**2.2.2.2 File and Directory Names** The `\markdownOptionHelperScriptFileName` macro sets the filename of the helper Lua script file that is created during the conversion from markdown to plain TeX in TeX engines without the `\directlua`

primitive. It defaults to `\jobname.markdown.lua`, where `\jobname` is the base name of the document being typeset.

The expansion of this macro must not contain quotation marks (") or backslash symbols (\). Mind that `TEX` engines tend to put quotation marks around `\jobname`, when it contains spaces.

```
569 \def\markdownOptionHelperScriptFileName{\jobname.markdown.lua}%
```

The `\markdownOptionInputTempFileName` macro sets the filename of the temporary input file that is created during the conversion from markdown to plain `TEX` in `\markdownMode` other than 2. It defaults to `\jobname.markdown.out`. The same limitations as in the case of the `\markdownOptionHelperScriptFileName` macro apply here.

```
570 \def\markdownOptionInputTempFileName{\jobname.markdown.in}%
```

The `\markdownOptionOutputTempFileName` macro sets the filename of the temporary output file that is created during the conversion from markdown to plain `TEX` in `\markdownMode` other than 2. It defaults to `\jobname.markdown.out`. The same limitations apply here as in the case of the `\markdownOptionHelperScriptFileName` macro.

```
571 \def\markdownOptionOutputTempFileName{\jobname.markdown.out}%
```

The `\markdownOptionErrorTempFileName` macro sets the filename of the temporary output file that is created when a Lua error is encountered during the conversion from markdown to plain `TEX` in `\markdownMode` other than 2. It defaults to `\jobname.markdown.err`. The same limitations apply here as in the case of the `\markdownOptionHelperScriptFileName` macro.

```
572 \def\markdownOptionErrorTempFileName{\jobname.markdown.err}%
```

The `\markdownOptionOutputDir` macro sets the path to the directory that will contain the auxiliary cache files produced by the Lua implementation and also the auxiliary files produced by the plain `TEX` implementation. The option defaults to `..`.

The path must be set to the same value as the `-output-directory` option of your `TEX` engine for the package to function correctly. We need this macro to make the Lua implementation aware where it should store the helper files. The same limitations apply here as in the case of the `\markdownOptionHelperScriptFileName` macro.

```
573 \def\markdownOptionOutputDir{.}%
```

The `\markdownOptionCacheDir` macro corresponds to the Lua interface `cacheDir` option that sets the path to the directory that will contain the produced cache files. The option defaults to `_markdown_\jobname`, which is a similar naming scheme to the one used by the minted `LATEX` package. The same limitations apply here as in the case of the `\markdownOptionHelperScriptFileName` macro.

```
574 \def\markdownOptionCacheDir{\markdownOptionOutputDir/_markdown_\jobname}%
```

The `\markdownOptionFrozenCacheFileName` macro corresponds to the Lua interface `frozenCacheFileName` option that sets the path to an output file (frozen

cache) that will contain a mapping between an enumeration of the markdown documents in the plain  $\text{\TeX}$  document and their auxiliary cache files. The option defaults to `frozenCache.tex`. The same limitations apply here as in the case of the `\markdownOptionHelperScriptFileName` macro.

```
575 \def\markdownOptionFrozenCacheFileName{\markdownOptionCacheDir/frozenCache.tex}
```

**2.2.2.3 Lua Interface Options** The following macros map directly to the options recognized by the Lua interface (see Section 2.1.2) and are not processed by the plain  $\text{\TeX}$  implementation, only passed along to Lua. They are undefined, which makes them fall back to the default values provided by the Lua interface.

For the macros that correspond to the non-boolean options recognized by the Lua interface, the same limitations apply here in the case of the `\markdownOptionHelperScriptFileName` macro.

```
576 \let\markdownOptionBlankBeforeBlockquote\undefined
577 \let\markdownOptionBlankBeforeCodeFence\undefined
578 \let\markdownOptionBlankBeforeHeading\undefined
579 \let\markdownOptionBreakableBlockquotes\undefined
580 \let\markdownOptionCitations\undefined
581 \let\markdownOptionCitationNbsps\undefined
582 \let\markdownOptionContentBlocks\undefined
583 \let\markdownOptionContentBlocksLanguageMap\undefined
584 \let\markdownOptionDefinitionLists\undefined
585 \let\markdownOptionEagerCache\undefined
586 \let\markdownOptionFootnotes\undefined
587 \let\markdownOptionFencedCode\undefined
588 \let\markdownOptionHardLineBreaks\undefined
589 \let\markdownOptionHashEnumerators\undefined
590 \let\markdownOptionHeaderAttributes\undefined
591 \let\markdownOptionHtml\undefined
592 \let\markdownOptionHybrid\undefined
593 \let\markdownOptionInlineFootnotes\undefined
594 \let\markdownOptionJekyllData\undefined
595 \let\markdownOptionPipeTables\undefined
596 \let\markdownOptionPreserveTabs\undefined
597 \let\markdownOptionRelativeReferences\undefined
598 \let\markdownOptionShiftHeadings\undefined
599 \let\markdownOptionSlice\undefined
600 \let\markdownOptionSmartEllipses\undefined
601 \let\markdownOptionStartNumber\undefined
602 \let\markdownOptionStripIndent\undefined
603 \let\markdownOptionTableCaptions\undefined
604 \let\markdownOptionTaskLists\undefined
605 \let\markdownOptionTexComments\undefined
606 \let\markdownOptionTightLists\undefined
```

**2.2.2.4 Miscellaneous Options** The `\markdownOptionStripPercentSigns` macro controls whether a percent sign (%) at the beginning of a line will be discarded when buffering Markdown input (see Section 3.2.5) or not. Notably, this enables the use of markdown when writing T<sub>E</sub>X package documentation using the Doc L<sup>A</sup>T<sub>E</sub>X package [5] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```
607 \def\markdownOptionStripPercentSigns{false}%
```

### 2.2.3 Token Renderers

The following T<sub>E</sub>X macros may occur inside the output of the converter functions exposed by the Lua interface (see Section 2.1.1) and represent the parsed markdown tokens. These macros are intended to be redefined by the user who is typesetting a document. By default, they point to the corresponding prototypes (see Section 2.2.4).

To enable the enumeration of token renderers, we will maintain the `\g_@@_renderers_seq` sequence.

```
608 \ExplSyntaxOn
609 \seq_new:N \g_@@_renderers_seq
```

To enable the reflection of token renderers and their parameters, we will maintain the `\g_@@_renderer_arities_prop` property list.

```
610 \prop_new:N \g_@@_renderer_arities_prop
611 \ExplSyntaxOff
```

**2.2.3.1 Tickbox Renderers** The macros named `\markdownRendererTickedBox`, `\markdownRendererHalfTickedBox`, and `\markdownRendererUntickedBox` represent ticked and unticked boxes, respectively. These macros will either be produced, when the `taskLists` option is enabled, or when the Ballot Box with X (☒, U+2612), Hourglass (⏸, U+231B) or Ballot Box (☐, U+2610) Unicode characters are encountered in the markdown input, respectively.

```
612 \def\markdownRendererTickedBox{%
613   \markdownRendererTickedBoxPrototype}%
614 \ExplSyntaxOn
615 \seq_put_right:Nn
616   \g_@@_renderers_seq
617   { tickedBox }
618 \prop_put:Nnn
619   \g_@@_renderer_arities_prop
620   { tickedBox }
621   { 0 }
622 \ExplSyntaxOff
623 \def\markdownRendererHalfTickedBox{%
624   \markdownRendererHalfTickedBoxPrototype}%
625 \ExplSyntaxOn
```

```

626 \seq_put_right:Nn
627   \g_@@_renderers_seq
628   { halfTickedBox }
629 \prop_put:Nnn
630   \g_@@_renderer_arities_prop
631   { halfTickedBox }
632   { 0 }
633 \ExplSyntaxOff
634 \def\markdownRendererUntickedBox{%
635   \markdownRendererUntickedBoxPrototype}%
636 \ExplSyntaxOn
637 \seq_put_right:Nn
638   \g_@@_renderers_seq
639   { untickedBox }
640 \prop_put:Nnn
641   \g_@@_renderer_arities_prop
642   { untickedBox }
643   { 0 }
644 \ExplSyntaxOff

```

**2.2.3.2 Markdown Document Renderers** The `\markdownRendererDocumentBegin` and `\markdownRendererDocumentEnd` macros represent the beginning and the end of a *markdown* document. The macros receive no arguments.

A  $\text{\TeX}$  document may contain any number of markdown documents. Additionally, markdown documents may appear not only in a sequence, but several markdown documents may also be *nested*. Redefinitions of the macros should take this into account.

```

645 \def\markdownRendererDocumentBegin{%
646   \markdownRendererDocumentBeginPrototype}%
647 \ExplSyntaxOn
648 \seq_put_right:Nn
649   \g_@@_renderers_seq
650   { documentBegin }
651 \prop_put:Nnn
652   \g_@@_renderer_arities_prop
653   { documentBegin }
654   { 0 }
655 \ExplSyntaxOff
656 \def\markdownRendererDocumentEnd{%
657   \markdownRendererDocumentEndPrototype}%
658 \ExplSyntaxOn
659 \seq_put_right:Nn
660   \g_@@_renderers_seq
661   { documentEnd }
662 \prop_put:Nnn

```



```

663 \g_@@_renderer_arities_prop
664 { documentEnd }
665 { 0 }
666 \ExplSyntaxOff

```

**2.2.3.3 Interblock Separator Renderer** The `\markdownRendererInterblockSeparator` macro represents a separator between two markdown block elements. The macro receives no arguments.

```

667 \def\markdownRendererInterblockSeparator{%
668 \markdownRendererInterblockSeparatorPrototype}%
669 \ExplSyntaxOn
670 \seq_put_right:Nn
671 \g_@@_renderers_seq
672 { interblockSeparator }
673 \prop_put:Nnn
674 \g_@@_renderer_arities_prop
675 { interblockSeparator }
676 { 0 }
677 \ExplSyntaxOff

```

**2.2.3.4 Line Break Renderer** The `\markdownRendererLineBreak` macro represents a forced line break. The macro receives no arguments.

```

678 \def\markdownRendererLineBreak{%
679 \markdownRendererLineBreakPrototype}%
680 \ExplSyntaxOn
681 \seq_put_right:Nn
682 \g_@@_renderers_seq
683 { lineBreak }
684 \prop_put:Nnn
685 \g_@@_renderer_arities_prop
686 { lineBreak }
687 { 0 }
688 \ExplSyntaxOff

```

**2.2.3.5 Ellipsis Renderer** The `\markdownRendererEllipsis` macro replaces any occurrence of ASCII ellipses in the input text. This macro will only be produced, when the `smartEllipses` option is enabled. The macro receives no arguments.

```

689 \def\markdownRendererEllipsis{%
690 \markdownRendererEllipsisPrototype}%
691 \ExplSyntaxOn
692 \seq_put_right:Nn
693 \g_@@_renderers_seq
694 { ellipsis }
695 \prop_put:Nnn

```

```

696 \g_@@_renderer_arities_prop
697 { ellipsis }
698 { 0 }
699 \ExplSyntaxOff

```

**2.2.3.6 Non-Breaking Space Renderer** The `\markdownRendererNbsp` macro represents a non-breaking space.

```

700 \def\markdownRendererNbsp{%
701 \markdownRendererNbspPrototype}%
702 \ExplSyntaxOn
703 \seq_put_right:Nn
704 \g_@@_renderers_seq
705 { nbsp }
706 \prop_put:Nnn
707 \g_@@_renderer_arities_prop
708 { nbsp }
709 { 0 }
710 \ExplSyntaxOff

```

**2.2.3.7 Special Character Renderers** The following macros replace any special plain  $\TeX$  characters, including the active pipe character (`|`) of `ConTeXt`, in the input text. These macros will only be produced, when the `hybrid` option is `false`.

```

711 \def\markdownRendererLeftBrace{%
712 \markdownRendererLeftBracePrototype}%
713 \ExplSyntaxOn
714 \seq_put_right:Nn
715 \g_@@_renderers_seq
716 { leftBrace }
717 \prop_put:Nnn
718 \g_@@_renderer_arities_prop
719 { leftBrace }
720 { 0 }
721 \ExplSyntaxOff
722 \def\markdownRendererRightBrace{%
723 \markdownRendererRightBracePrototype}%
724 \ExplSyntaxOn
725 \seq_put_right:Nn
726 \g_@@_renderers_seq
727 { rightBrace }
728 \prop_put:Nnn
729 \g_@@_renderer_arities_prop
730 { rightBrace }
731 { 0 }
732 \ExplSyntaxOff
733 \def\markdownRendererDollarSign{%

```

```

734 \markdownRendererDollarSignPrototype}%
735 \ExplSyntaxOn
736 \seq_put_right:Nn
737 \g_@@_renderers_seq
738 { dollarSign }
739 \prop_put:Nnn
740 \g_@@_renderer_arities_prop
741 { dollarSign }
742 { 0 }
743 \ExplSyntaxOff
744 \def\markdownRendererPercentSign{%
745 \markdownRendererPercentSignPrototype}%
746 \ExplSyntaxOn
747 \seq_put_right:Nn
748 \g_@@_renderers_seq
749 { percentSign }
750 \prop_put:Nnn
751 \g_@@_renderer_arities_prop
752 { percentSign }
753 { 0 }
754 \ExplSyntaxOff
755 \def\markdownRendererAmpersand{%
756 \markdownRendererAmpersandPrototype}%
757 \ExplSyntaxOn
758 \seq_put_right:Nn
759 \g_@@_renderers_seq
760 { ampersand }
761 \prop_put:Nnn
762 \g_@@_renderer_arities_prop
763 { ampersand }
764 { 0 }
765 \ExplSyntaxOff
766 \def\markdownRendererUnderscore{%
767 \markdownRendererUnderscorePrototype}%
768 \ExplSyntaxOn
769 \seq_put_right:Nn
770 \g_@@_renderers_seq
771 { underscore }
772 \prop_put:Nnn
773 \g_@@_renderer_arities_prop
774 { underscore }
775 { 0 }
776 \ExplSyntaxOff
777 \def\markdownRendererHash{%
778 \markdownRendererHashPrototype}%
779 \ExplSyntaxOn
780 \seq_put_right:Nn

```

```

781 \g_@@_renderers_seq
782 { hash }
783 \prop_put:Nnn
784 \g_@@_renderer_arities_prop
785 { hash }
786 { 0 }
787 \ExplSyntaxOff
788 \def\markdownRendererCircumflex{%
789 \markdownRendererCircumflexPrototype}%
790 \ExplSyntaxOn
791 \seq_put_right:Nn
792 \g_@@_renderers_seq
793 { circumflex }
794 \prop_put:Nnn
795 \g_@@_renderer_arities_prop
796 { circumflex }
797 { 0 }
798 \ExplSyntaxOff
799 \def\markdownRendererBackslash{%
800 \markdownRendererBackslashPrototype}%
801 \ExplSyntaxOn
802 \seq_put_right:Nn
803 \g_@@_renderers_seq
804 { backslash }
805 \prop_put:Nnn
806 \g_@@_renderer_arities_prop
807 { backslash }
808 { 0 }
809 \ExplSyntaxOff
810 \def\markdownRendererTilde{%
811 \markdownRendererTildePrototype}%
812 \ExplSyntaxOn
813 \seq_put_right:Nn
814 \g_@@_renderers_seq
815 { tilde }
816 \prop_put:Nnn
817 \g_@@_renderer_arities_prop
818 { tilde }
819 { 0 }
820 \ExplSyntaxOff
821 \def\markdownRendererPipe{%
822 \markdownRendererPipePrototype}%
823 \ExplSyntaxOn
824 \seq_put_right:Nn
825 \g_@@_renderers_seq
826 { pipe }
827 \prop_put:Nnn

```

```

828 \g_@@_renderer_arities_prop
829 { pipe }
830 { 0 }
831 \ExplSyntaxOff

```

**2.2.3.8 Code Span Renderer** The `\markdownRendererCodeSpan` macro represents inlined code span in the input text. It receives a single argument that corresponds to the inlined code span.

```

832 \def\markdownRendererCodeSpan{%
833 \markdownRendererCodeSpanPrototype}%
834 \ExplSyntaxOn
835 \seq_put_right:Nn
836 \g_@@_renderers_seq
837 { codeSpan }
838 \prop_put:Nnn
839 \g_@@_renderer_arities_prop
840 { codeSpan }
841 { 1 }
842 \ExplSyntaxOff

```

**2.2.3.9 Link Renderer** The `\markdownRendererLink` macro represents a hyperlink. It receives four arguments: the label, the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the link.

```

843 \def\markdownRendererLink{%
844 \markdownRendererLinkPrototype}%
845 \ExplSyntaxOn
846 \seq_put_right:Nn
847 \g_@@_renderers_seq
848 { link }
849 \prop_put:Nnn
850 \g_@@_renderer_arities_prop
851 { link }
852 { 4 }
853 \ExplSyntaxOff

```

**2.2.3.10 Image Renderer** The `\markdownRendererImage` macro represents an image. It receives four arguments: the label, the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the link.

```

854 \def\markdownRendererImage{%
855 \markdownRendererImagePrototype}%
856 \ExplSyntaxOn
857 \seq_put_right:Nn
858 \g_@@_renderers_seq

```

```

859 { image }
860 \prop_put:Nnn
861 \g_@@_renderer_arities_prop
862 { image }
863 { 4 }
864 \ExplSyntaxOff

```

**2.2.3.11 Content Block Rendere** The `\markdownRendererContentBlock` macro represents an iA Writer content block. It receives four arguments: the local file or online image filename extension cast to the lower case, the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the content block.

```

865 \def\markdownRendererContentBlock{%
866 \markdownRendererContentBlockPrototype}%
867 \ExplSyntaxOn
868 \seq_put_right:Nn
869 \g_@@_renderers_seq
870 { contentBlock }
871 \prop_put:Nnn
872 \g_@@_renderer_arities_prop
873 { contentBlock }
874 { 4 }
875 \ExplSyntaxOff

```

The `\markdownRendererContentBlockOnlineImage` macro represents an iA Writer online image content block. The macro receives the same arguments as `\markdownRendererContentBlock`.

```

876 \def\markdownRendererContentBlockOnlineImage{%
877 \markdownRendererContentBlockOnlineImagePrototype}%
878 \ExplSyntaxOn
879 \seq_put_right:Nn
880 \g_@@_renderers_seq
881 { contentBlockOnlineImage }
882 \prop_put:Nnn
883 \g_@@_renderer_arities_prop
884 { contentBlockOnlineImage }
885 { 4 }
886 \ExplSyntaxOff

```

The `\markdownRendererContentBlockCode` macro represents an iA Writer content block that was recognized as a file in a known programming language by its filename extension  $s$ . If any `markdown-languages.json` file found by `kpathsea`<sup>7</sup> contains a record  $(k, v)$ , then a non-online-image content block with the filename

---

<sup>7</sup>Local files take precedence. Filenames other than `markdown-languages.json` may be specified using the `contentBlocksLanguageMap` Lua option.

extension  $s$ ,  $s:\text{lower}() = k$  is considered to be in a known programming language  $v$ . The macro receives five arguments: the local file name extension  $s$  cast to the lower case, the language  $v$ , the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the content block.

Note that you will need to place a `markdown-languages.json` file inside your working directory or inside your local  $\text{T}_{\text{E}}\text{X}$  directory structure. In this file, you will define a mapping between filename extensions and the language names recognized by your favorite syntax highlighter; there may exist other creative uses beside syntax highlighting. The `Languages.json` file provided by Sotkov [3] is a good starting point.

```

887 \def\markdownRendererContentBlockCode{%
888   \markdownRendererContentBlockCodePrototype}%
889 \ExplSyntaxOn
890 \seq_put_right:Nn
891   \g_@@_renderers_seq
892   { contentBlockCode }
893 \prop_put:Nnn
894   \g_@@_renderer_arities_prop
895   { contentBlockCode }
896   { 5 }
897 \ExplSyntaxOff

```

**2.2.3.12 Bullet List Renderers** The `\markdownRendererUlBegin` macro represents the beginning of a bulleted list that contains an item with several paragraphs of text (the list is not tight). The macro receives no arguments.

```

898 \def\markdownRendererUlBegin{%
899   \markdownRendererUlBeginPrototype}%
900 \ExplSyntaxOn
901 \seq_put_right:Nn
902   \g_@@_renderers_seq
903   { ulBegin }
904 \prop_put:Nnn
905   \g_@@_renderer_arities_prop
906   { ulBegin }
907   { 0 }
908 \ExplSyntaxOff

```

The `\markdownRendererUlBeginTight` macro represents the beginning of a bulleted list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `tightLists` option is `false`. The macro receives no arguments.

```

909 \def\markdownRendererUlBeginTight{%
910   \markdownRendererUlBeginTightPrototype}%
911 \ExplSyntaxOn

```

```

912 \seq_put_right:Nn
913   \g_@@_renderers_seq
914   { ulBeginTight }
915 \prop_put:Nnn
916   \g_@@_renderer_arities_prop
917   { ulBeginTight }
918   { 0 }
919 \ExplSyntaxOff

```

The `\markdownRendererUListItem` macro represents an item in a bulleted list. The macro receives no arguments.

```

920 \def\markdownRendererUListItem{%
921   \markdownRendererUListItemPrototype}%
922 \ExplSyntaxOn
923 \seq_put_right:Nn
924   \g_@@_renderers_seq
925   { ulItem }
926 \prop_put:Nnn
927   \g_@@_renderer_arities_prop
928   { ulItem }
929   { 0 }
930 \ExplSyntaxOff

```

The `\markdownRendererUListItemEnd` macro represents the end of an item in a bulleted list. The macro receives no arguments.

```

931 \def\markdownRendererUListItemEnd{%
932   \markdownRendererUListItemEndPrototype}%
933 \ExplSyntaxOn
934 \seq_put_right:Nn
935   \g_@@_renderers_seq
936   { ulItemEnd }
937 \prop_put:Nnn
938   \g_@@_renderer_arities_prop
939   { ulItemEnd }
940   { 0 }
941 \ExplSyntaxOff

```

The `\markdownRendererUListEnd` macro represents the end of a bulleted list that contains an item with several paragraphs of text (the list is not tight). The macro receives no arguments.

```

942 \def\markdownRendererUListEnd{%
943   \markdownRendererUListEndPrototype}%
944 \ExplSyntaxOn
945 \seq_put_right:Nn
946   \g_@@_renderers_seq
947   { ulEnd }

```



```

948 \prop_put:Nnn
949   \g_@@_renderer_arities_prop
950   { ulEnd }
951   { 0 }
952 \ExplSyntaxOff

```

The `\markdownRendererU1EndTight` macro represents the end of a bulleted list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `tightLists` option is `false`. The macro receives no arguments.

```

953 \def\markdownRendererU1EndTight{%
954   \markdownRendererU1EndTightPrototype}%
955 \ExplSyntaxOn
956 \seq_put_right:Nn
957   \g_@@_renderers_seq
958   { ulEndTight }
959 \prop_put:Nnn
960   \g_@@_renderer_arities_prop
961   { ulEndTight }
962   { 0 }
963 \ExplSyntaxOff

```

**2.2.3.13 Ordered List Renderers** The `\markdownRendererO1Begin` macro represents the beginning of an ordered list that contains an item with several paragraphs of text (the list is not tight). The macro receives no arguments.

```

964 \def\markdownRendererO1Begin{%
965   \markdownRendererO1BeginPrototype}%
966 \ExplSyntaxOn
967 \seq_put_right:Nn
968   \g_@@_renderers_seq
969   { olBegin }
970 \prop_put:Nnn
971   \g_@@_renderer_arities_prop
972   { olBegin }
973   { 0 }
974 \ExplSyntaxOff

```

The `\markdownRendererO1BeginTight` macro represents the beginning of an ordered list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `tightLists` option is `false`. The macro receives no arguments.

```

975 \def\markdownRendererO1BeginTight{%
976   \markdownRendererO1BeginTightPrototype}%
977 \ExplSyntaxOn
978 \seq_put_right:Nn

```

```

979 \g_@@_renderers_seq
980 { olBeginTight }
981 \prop_put:Nnn
982 \g_@@_renderer_arities_prop
983 { olBeginTight }
984 { 0 }
985 \ExplSyntaxOff

```

The `\markdownRendererOItem` macro represents an item in an ordered list. This macro will only be produced, when the `startNumber` option is `false`. The macro receives no arguments.

```

986 \def\markdownRendererOItem{%
987 \markdownRendererOItemPrototype}%
988 \ExplSyntaxOn
989 \seq_put_right:Nn
990 \g_@@_renderers_seq
991 { olItem }
992 \prop_put:Nnn
993 \g_@@_renderer_arities_prop
994 { olItem }
995 { 0 }
996 \ExplSyntaxOff

```

The `\markdownRendererOItemEnd` macro represents the end of an item in an ordered list. The macro receives no arguments.

```

997 \def\markdownRendererOItemEnd{%
998 \markdownRendererOItemEndPrototype}%
999 \ExplSyntaxOn
1000 \seq_put_right:Nn
1001 \g_@@_renderers_seq
1002 { olItemEnd }
1003 \prop_put:Nnn
1004 \g_@@_rendered_arities_prop
1005 { olItemEnd }
1006 { 0 }
1007 \ExplSyntaxOff

```

The `\markdownRendererOItemWithNumber` macro represents an item in an ordered list. This macro will only be produced, when the `startNumber` option is enabled. The macro receives a single numeric argument that corresponds to the item number.

```

1008 \def\markdownRendererOItemWithNumber{%
1009 \markdownRendererOItemWithNumberPrototype}%
1010 \ExplSyntaxOn
1011 \seq_put_right:Nn
1012 \g_@@_renderers_seq

```

```

1013 { olItemWithNumber }
1014 \prop_put:Nnn
1015 \g_@@_renderer_arities_prop
1016 { olItemWithNumber }
1017 { 1 }
1018 \ExplSyntaxOff

```

The `\markdownRendererOlEnd` macro represents the end of an ordered list that contains an item with several paragraphs of text (the list is not tight). The macro receives no arguments.

```

1019 \def\markdownRendererOlEnd{%
1020 \markdownRendererOlEndPrototype}%
1021 \ExplSyntaxOn
1022 \seq_put_right:Nn
1023 \g_@@_renderers_seq
1024 { olEnd }
1025 \prop_put:Nnn
1026 \g_@@_renderer_arities_prop
1027 { olEnd }
1028 { 0 }
1029 \ExplSyntaxOff

```

The `\markdownRendererOlEndTight` macro represents the end of an ordered list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `tightLists` option is `false`. The macro receives no arguments.

```

1030 \def\markdownRendererOlEndTight{%
1031 \markdownRendererOlEndTightPrototype}%
1032 \ExplSyntaxOn
1033 \seq_put_right:Nn
1034 \g_@@_renderers_seq
1035 { olEndTight }
1036 \prop_put:Nnn
1037 \g_@@_renderer_arities_prop
1038 { olEndTight }
1039 { 0 }
1040 \ExplSyntaxOff

```

**2.2.3.14 Definition List Renderers** The following macros are only produced, when the `definitionLists` option is enabled.

The `\markdownRendererDlBegin` macro represents the beginning of a definition list that contains an item with several paragraphs of text (the list is not tight). The macro receives no arguments.

```

1041 \def\markdownRendererDlBegin{%
1042 \markdownRendererDlBeginPrototype}%

```

```

1043 \ExplSyntaxOn
1044 \seq_put_right:Nn
1045   \g_@@_renderers_seq
1046   { dlBegin }
1047 \prop_put:Nnn
1048   \g_@@_renderer_arities_prop
1049   { dlBegin }
1050   { 0 }
1051 \ExplSyntaxOff

```

The `\markdownRendererDlBeginTight` macro represents the beginning of a definition list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the `tightLists` option is `false`. The macro receives no arguments.

```

1052 \def\markdownRendererDlBeginTight{%
1053   \markdownRendererDlBeginTightPrototype}%
1054 \ExplSyntaxOn
1055 \seq_put_right:Nn
1056   \g_@@_renderers_seq
1057   { dlBeginTight }
1058 \prop_put:Nnn
1059   \g_@@_renderer_arities_prop
1060   { dlBeginTight }
1061   { 0 }
1062 \ExplSyntaxOff

```

The `\markdownRendererDlItem` macro represents a term in a definition list. The macro receives a single argument that corresponds to the term being defined.

```

1063 \def\markdownRendererDlItem{%
1064   \markdownRendererDlItemPrototype}%
1065 \ExplSyntaxOn
1066 \seq_put_right:Nn
1067   \g_@@_renderers_seq
1068   { dlItem }
1069 \prop_put:Nnn
1070   \g_@@_renderer_arities_prop
1071   { dlItem }
1072   { 1 }
1073 \ExplSyntaxOff

```

The `\markdownRendererDlItemEnd` macro represents the end of a list of definitions for a single term.

```

1074 \def\markdownRendererDlItemEnd{%
1075   \markdownRendererDlItemEndPrototype}%
1076 \ExplSyntaxOn
1077 \seq_put_right:Nn

```

```

1078 \g_@@_renderers_seq
1079 { dlItemEnd }
1080 \prop_put:Nnn
1081 \g_@@_renderer_arities_prop
1082 { dlItemEnd }
1083 { 0 }
1084 \ExplSyntaxOff

```

The `\markdownRendererDlDefinitionBegin` macro represents the beginning of a definition in a definition list. There can be several definitions for a single term.

```

1085 \def\markdownRendererDlDefinitionBegin{%
1086 \markdownRendererDlDefinitionBeginPrototype}%
1087 \ExplSyntaxOn
1088 \seq_put_right:Nn
1089 \g_@@_renderers_seq
1090 { dlDefinitionBegin }
1091 \prop_put:Nnn
1092 \g_@@_renderer_arities_prop
1093 { dlDefinitionBegin }
1094 { 0 }
1095 \ExplSyntaxOff

```

The `\markdownRendererDlDefinitionEnd` macro represents the end of a definition in a definition list. There can be several definitions for a single term.

```

1096 \def\markdownRendererDlDefinitionEnd{%
1097 \markdownRendererDlDefinitionEndPrototype}%
1098 \ExplSyntaxOn
1099 \seq_put_right:Nn
1100 \g_@@_renderers_seq
1101 { dlDefinitionEnd }
1102 \prop_put:Nnn
1103 \g_@@_renderer_arities_prop
1104 { dlDefinitionEnd }
1105 { 0 }
1106 \ExplSyntaxOff

```

The `\markdownRendererDlEnd` macro represents the end of a definition list that contains an item with several paragraphs of text (the list is not tight). The macro receives no arguments.

```

1107 \def\markdownRendererDlEnd{%
1108 \markdownRendererDlEndPrototype}%
1109 \ExplSyntaxOn
1110 \seq_put_right:Nn
1111 \g_@@_renderers_seq
1112 { dlEnd }
1113 \prop_put:Nnn

```

```

1114 \g_@@_renderer_arities_prop
1115 { dlEnd }
1116 { 0 }
1117 \ExplSyntaxOff

```

The `\markdownRendererDlEndTight` macro represents the end of a definition list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `tightLists` option is `false`. The macro receives no arguments.

```

1118 \def\markdownRendererDlEndTight{%
1119 \markdownRendererDlEndTightPrototype}%
1120 \ExplSyntaxOn
1121 \seq_put_right:Nn
1122 \g_@@_renderers_seq
1123 { dlEndTight }
1124 \prop_put:Nnn
1125 \g_@@_renderer_arities_prop
1126 { dlEndTight }
1127 { 0 }
1128 \ExplSyntaxOff

```

**2.2.3.15 Emphasis Renderers** The `\markdownRendererEmphasis` macro represents an emphasized span of text. The macro receives a single argument that corresponds to the emphasized span of text.

```

1129 \def\markdownRendererEmphasis{%
1130 \markdownRendererEmphasisPrototype}%
1131 \ExplSyntaxOn
1132 \seq_put_right:Nn
1133 \g_@@_renderers_seq
1134 { emphasis }
1135 \prop_put:Nnn
1136 \g_@@_renderer_arities_prop
1137 { emphasis }
1138 { 1 }
1139 \ExplSyntaxOff

```

The `\markdownRendererStrongEmphasis` macro represents a strongly emphasized span of text. The macro receives a single argument that corresponds to the emphasized span of text.

```

1140 \def\markdownRendererStrongEmphasis{%
1141 \markdownRendererStrongEmphasisPrototype}%
1142 \ExplSyntaxOn
1143 \seq_put_right:Nn
1144 \g_@@_renderers_seq
1145 { strongEmphasis }

```

```

1146 \prop_put:Nnn
1147   \g_@@_renderer_arities_prop
1148   { strongEmphasis }
1149   { 1 }
1150 \ExplSyntaxOff

```

**2.2.3.16 Block Quote Renderers** The `\markdownRendererBlockQuoteBegin` macro represents the beginning of a block quote. The macro receives no arguments.

```

1151 \def\markdownRendererBlockQuoteBegin{%
1152   \markdownRendererBlockQuoteBeginPrototype}%
1153 \ExplSyntaxOn
1154 \seq_put_right:Nn
1155   \g_@@_renderers_seq
1156   { blockQuoteBegin }
1157 \prop_put:Nnn
1158   \g_@@_renderer_arities_prop
1159   { blockQuoteBegin }
1160   { 0 }
1161 \ExplSyntaxOff

```

The `\markdownRendererBlockQuoteEnd` macro represents the end of a block quote. The macro receives no arguments.

```

1162 \def\markdownRendererBlockQuoteEnd{%
1163   \markdownRendererBlockQuoteEndPrototype}%
1164 \ExplSyntaxOn
1165 \seq_put_right:Nn
1166   \g_@@_renderers_seq
1167   { blockQuoteEnd }
1168 \prop_put:Nnn
1169   \g_@@_renderer_arities_prop
1170   { blockQuoteEnd }
1171   { 0 }
1172 \ExplSyntaxOff

```

**2.2.3.17 Code Block Renderers** The `\markdownRendererInputVerbatim` macro represents a code block. The macro receives a single argument that corresponds to the filename of a file containing the code block contents.

```

1173 \def\markdownRendererInputVerbatim{%
1174   \markdownRendererInputVerbatimPrototype}%
1175 \ExplSyntaxOn
1176 \seq_put_right:Nn
1177   \g_@@_renderers_seq
1178   { inputVerbatim }
1179 \prop_put:Nnn
1180   \g_@@_renderer_arities_prop

```

```

1181 { inputVerbatim }
1182 { 1 }
1183 \ExplSyntaxOff

```

The `\markdownRendererInputFencedCode` macro represents a fenced code block. This macro will only be produced, when the `fencedCode` option is enabled. The macro receives two arguments that correspond to the filename of a file containing the code block contents and to the code fence infostring.

```

1184 \def\markdownRendererInputFencedCode{%
1185   \markdownRendererInputFencedCodePrototype}%
1186 \ExplSyntaxOn
1187 \seq_put_right:Nn
1188   \g_@@_renderers_seq
1189   { inputFencedCode }
1190 \prop_put:Nnn
1191   \g_@@_renderer_arities_prop
1192   { inputFencedCode }
1193   { 2 }
1194 \ExplSyntaxOff

```

**2.2.3.18 YAML Metadata Renderers** The `\markdownRendererJekyllDataBegin` macro represents the beginning of a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives no arguments.

```

1195 \def\markdownRendererJekyllDataBegin{%
1196   \markdownRendererJekyllDataBeginPrototype}%
1197 \ExplSyntaxOn
1198 \seq_put_right:Nn
1199   \g_@@_renderers_seq
1200   { jekyllDataBegin }
1201 \prop_put:Nnn
1202   \g_@@_renderer_arities_prop
1203   { jekyllDataBegin }
1204   { 0 }
1205 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataEnd` macro represents the end of a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives no arguments.

```

1206 \def\markdownRendererJekyllDataEnd{%
1207   \markdownRendererJekyllDataEndPrototype}%
1208 \ExplSyntaxOn
1209 \seq_put_right:Nn
1210   \g_@@_renderers_seq
1211   { jekyllDataEnd }
1212 \prop_put:Nnn

```



```

1213 \g_@@_renderer_arities_prop
1214 { jekyllDataEnd }
1215 { 0 }
1216 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataMappingBegin` macro represents the beginning of a mapping in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives two arguments: the scalar key in the parent structure, cast to a string following YAML serialization rules, and the number of items in the mapping.

```

1217 \def\markdownRendererJekyllDataMappingBegin{%
1218 \markdownRendererJekyllDataMappingBeginPrototype}%
1219 \ExplSyntaxOn
1220 \seq_put_right:Nn
1221 \g_@@_renderers_seq
1222 { jekyllDataMappingBegin }
1223 \prop_put:Nnn
1224 \g_@@_renderer_arities_prop
1225 { jekyllDataMappingBegin }
1226 { 2 }
1227 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataMappingEnd` macro represents the end of a mapping in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives no arguments.

```

1228 \def\markdownRendererJekyllDataMappingEnd{%
1229 \markdownRendererJekyllDataMappingEndPrototype}%
1230 \ExplSyntaxOn
1231 \seq_put_right:Nn
1232 \g_@@_renderers_seq
1233 { jekyllDataMappingEnd }
1234 \prop_put:Nnn
1235 \g_@@_renderer_arities_prop
1236 { jekyllDataMappingEnd }
1237 { 0 }
1238 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataSequenceBegin` macro represents the beginning of a sequence in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives two arguments: the scalar key in the parent structure, cast to a string following YAML serialization rules, and the number of items in the sequence.

```

1239 \def\markdownRendererJekyllDataSequenceBegin{%
1240 \markdownRendererJekyllDataSequenceBeginPrototype}%
1241 \ExplSyntaxOn
1242 \seq_put_right:Nn

```

```

1243 \g_@@_renderers_seq
1244 { jekyllDataSequenceBegin }
1245 \prop_put:Nnn
1246 \g_@@_renderers_aritys_prop
1247 { jekyllDataSequenceBegin }
1248 { 2 }
1249 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataSequenceEnd` macro represents the end of a sequence in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives no arguments.

```

1250 \def\markdownRendererJekyllDataSequenceEnd{%
1251 \markdownRendererJekyllDataSequenceEndPrototype}%
1252 \ExplSyntaxOn
1253 \seq_put_right:Nn
1254 \g_@@_renderers_seq
1255 { jekyllDataSequenceEnd }
1256 \prop_put:Nnn
1257 \g_@@_renderers_aritys_prop
1258 { jekyllDataSequenceEnd }
1259 { 0 }
1260 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataBoolean` macro represents a boolean scalar value in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives two arguments: the scalar key in the parent structure, and the scalar value, both cast to a string following YAML serialization rules.

```

1261 \def\markdownRendererJekyllDataBoolean{%
1262 \markdownRendererJekyllDataBooleanPrototype}%
1263 \ExplSyntaxOn
1264 \seq_put_right:Nn
1265 \g_@@_renderers_seq
1266 { jekyllDataBoolean }
1267 \prop_put:Nnn
1268 \g_@@_renderers_aritys_prop
1269 { jekyllDataBoolean }
1270 { 2 }
1271 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataNumber` macro represents a numeric scalar value in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives two arguments: the scalar key in the parent structure, and the scalar value, both cast to a string following YAML serialization rules.

```

1272 \def\markdownRendererJekyllDataNumber{%

```

```

1273 \markdownRendererJekyllDataNumberPrototype}%
1274 \ExplSyntaxOn
1275 \seq_put_right:Nn
1276 \g_@@_renderers_seq
1277 { jekyllDataNumber }
1278 \prop_put:Nnn
1279 \g_@@_renderer_arities_prop
1280 { jekyllDataNumber }
1281 { 2 }
1282 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataString` macro represents a string scalar value in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives two arguments: the scalar key in the parent structure, cast to a string following YAML serialization rules, and the scalar value.

```

1283 \def\markdownRendererJekyllDataString{%
1284 \markdownRendererJekyllDataStringPrototype}%
1285 \ExplSyntaxOn
1286 \seq_put_right:Nn
1287 \g_@@_renderers_seq
1288 { jekyllDataString }
1289 \prop_put:Nnn
1290 \g_@@_renderer_arities_prop
1291 { jekyllDataString }
1292 { 2 }
1293 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataEmpty` macro represents an empty scalar value in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives one argument: the scalar key in the parent structure, cast to a string following YAML serialization rules.

See also Section 2.2.4.1 for the description of the high-level `expl3` interface that you can also use to react to YAML metadata.

```

1294 \def\markdownRendererJekyllDataEmpty{%
1295 \markdownRendererJekyllDataEmptyPrototype}%
1296 \ExplSyntaxOn
1297 \seq_put_right:Nn
1298 \g_@@_renderers_seq
1299 { jekyllDataEmpty }
1300 \prop_put:Nnn
1301 \g_@@_renderer_arities_prop
1302 { jekyllDataEmpty }
1303 { 1 }
1304 \ExplSyntaxOff

```

**2.2.3.19 Heading Renderers** The `\markdownRendererHeadingOne` macro represents a first level heading. The macro receives a single argument that corresponds to the heading text.

```

1305 \def\markdownRendererHeadingOne{%
1306   \markdownRendererHeadingOnePrototype}%
1307 \ExplSyntaxOn
1308 \seq_put_right:Nn
1309   \g_@@_renderers_seq
1310   { headingOne }
1311 \prop_put:Nnn
1312   \g_@@_renderer_arities_prop
1313   { headingOne }
1314   { 1 }
1315 \ExplSyntaxOff

```

The `\markdownRendererHeadingTwo` macro represents a second level heading. The macro receives a single argument that corresponds to the heading text.

```

1316 \def\markdownRendererHeadingTwo{%
1317   \markdownRendererHeadingTwoPrototype}%
1318 \ExplSyntaxOn
1319 \seq_put_right:Nn
1320   \g_@@_renderers_seq
1321   { headingTwo }
1322 \prop_put:Nnn
1323   \g_@@_renderer_arities_prop
1324   { headingTwo }
1325   { 1 }
1326 \ExplSyntaxOff

```

The `\markdownRendererHeadingThree` macro represents a third level heading. The macro receives a single argument that corresponds to the heading text.

```

1327 \def\markdownRendererHeadingThree{%
1328   \markdownRendererHeadingThreePrototype}%
1329 \ExplSyntaxOn
1330 \seq_put_right:Nn
1331   \g_@@_renderers_seq
1332   { headingThree }
1333 \prop_put:Nnn
1334   \g_@@_renderer_arities_prop
1335   { headingThree }
1336   { 1 }
1337 \ExplSyntaxOff

```

The `\markdownRendererHeadingFour` macro represents a fourth level heading. The macro receives a single argument that corresponds to the heading text.

```

1338 \def\markdownRendererHeadingFour{%

```

```

1339 \markdownRendererHeadingFourPrototype}%
1340 \ExplSyntaxOn
1341 \seq_put_right:Nn
1342 \g_@@_renderers_seq
1343 { headingFour }
1344 \prop_put:Nnn
1345 \g_@@_renderer_arities_prop
1346 { headingFour }
1347 { 1 }
1348 \ExplSyntaxOff

```

The `\markdownRendererHeadingFive` macro represents a fifth level heading. The macro receives a single argument that corresponds to the heading text.

```

1349 \def\markdownRendererHeadingFive{%
1350 \markdownRendererHeadingFivePrototype}%
1351 \ExplSyntaxOn
1352 \seq_put_right:Nn
1353 \g_@@_renderers_seq
1354 { headingFive }
1355 \prop_put:Nnn
1356 \g_@@_renderer_arities_prop
1357 { headingFive }
1358 { 1 }
1359 \ExplSyntaxOff

```

The `\markdownRendererHeadingSix` macro represents a sixth level heading. The macro receives a single argument that corresponds to the heading text.

```

1360 \def\markdownRendererHeadingSix{%
1361 \markdownRendererHeadingSixPrototype}%
1362 \ExplSyntaxOn
1363 \seq_put_right:Nn
1364 \g_@@_renderers_seq
1365 { headingSix }
1366 \prop_put:Nnn
1367 \g_@@_renderer_arities_prop
1368 { headingSix }
1369 { 1 }
1370 \ExplSyntaxOff

```

**2.2.3.20 Horizontal Rule Renderer** The `\markdownRendererHorizontalRule` macro represents a horizontal rule. The macro receives no arguments.

```

1371 \def\markdownRendererHorizontalRule{%
1372 \markdownRendererHorizontalRulePrototype}%
1373 \ExplSyntaxOn
1374 \seq_put_right:Nn
1375 \g_@@_renderers_seq

```

```

1376 { horizontalRule }
1377 \prop_put:Nnn
1378 \g_@@_renderer_arities_prop
1379 { horizontalRule }
1380 { 0 }
1381 \ExplSyntaxOff

```

**2.2.3.21 Footnote Renderer** The `\markdownRendererFootnote` macro represents a footnote. This macro will only be produced, when the `footnotes` option is enabled. The macro receives a single argument that corresponds to the footnote text.

```

1382 \def\markdownRendererFootnote{%
1383 \markdownRendererFootnotePrototype}%
1384 \ExplSyntaxOn
1385 \seq_put_right:Nn
1386 \g_@@_renderers_seq
1387 { footnote }
1388 \prop_put:Nnn
1389 \g_@@_renderer_arities_prop
1390 { footnote }
1391 { 1 }
1392 \ExplSyntaxOff

```

**2.2.3.22 Parenthesized Citations Renderer** The `\markdownRendererCite` macro represents a string of one or more parenthetical citations. This macro will only be produced, when the `citations` option is enabled. The macro receives the parameter `{<number of citations>}` followed by `<suppress author>` `{<prenote>}{<postnote>}{<name>}` repeated `<number of citations>` times. The `<suppress author>` parameter is either the token `-`, when the author's name is to be suppressed, or `+` otherwise.

```

1393 \def\markdownRendererCite{%
1394 \markdownRendererCitePrototype}%
1395 \ExplSyntaxOn
1396 \seq_put_right:Nn
1397 \g_@@_renderers_seq
1398 { cite }
1399 \prop_put:Nnn
1400 \g_@@_renderer_arities_prop
1401 { cite }
1402 { 1 }
1403 \ExplSyntaxOff

```

**2.2.3.23 Text Citations Renderer** The `\markdownRendererTextCite` macro represents a string of one or more text citations. This macro will only be produced,

when the `citations` option is enabled. The macro receives parameters in the same format as the `\markdownRendererCite` macro.

```

1404 \def\markdownRendererTextCite{%
1405   \markdownRendererTextCitePrototype}%
1406 \ExplSyntaxOn
1407 \seq_put_right:Nn
1408   \g_@@_renderers_seq
1409   { textCite }
1410 \prop_put:Nnn
1411   \g_@@_renderer_arities_prop
1412   { textCite }
1413   { 1 }
1414 \ExplSyntaxOff

```

**2.2.3.24 Table Renderer** The `\markdownRendererTable` macro represents a table. This macro will only be produced, when the `pipeTables` option is enabled. The macro receives the parameters `{<caption>}{<number of rows>}{<number of columns>}` followed by `{<alignments>}` and then by `{<row>}` repeated `<number of rows>` times, where `<row>` is `{<column>}` repeated `<number of columns>` times, `<alignments>` is `<alignment>` repeated `<number of columns>` times, and `<alignment>` is one of the following:

- `d` – The corresponding column has an unspecified (default) alignment.
- `l` – The corresponding column is left-aligned.
- `c` – The corresponding column is centered.
- `r` – The corresponding column is right-aligned.

```

1415 \def\markdownRendererTable{%
1416   \markdownRendererTablePrototype}%
1417 \ExplSyntaxOn
1418 \seq_put_right:Nn
1419   \g_@@_renderers_seq
1420   { table }
1421 \prop_put:Nnn
1422   \g_@@_renderer_arities_prop
1423   { table }
1424   { 3 }
1425 \ExplSyntaxOff

```

**2.2.3.25 HTML Comment Renderers** The `\markdownRendererInlineHtmlComment` macro represents the contents of an inline HTML comment. This macro will only be produced, when the `html` option is enabled. The macro receives a single argument that corresponds to the contents of the HTML comment.

The `\markdownRendererBlockHtmlCommentBegin` and `\markdownRendererBlockHtmlCommentEnd` macros represent the beginning and the end of a block HTML comment. The macros receive no arguments.

```

1426 \def\markdownRendererInlineHtmlComment{%
1427   \markdownRendererInlineHtmlCommentPrototype}%
1428 \ExplSyntaxOn
1429 \seq_put_right:Nn
1430   \g_@@_renderers_seq
1431   { inlineHtmlComment }
1432 \prop_put:Nnn
1433   \g_@@_renderer_arities_prop
1434   { inlineHtmlComment }
1435   { 1 }
1436 \ExplSyntaxOff
1437 \def\markdownRendererBlockHtmlCommentBegin{%
1438   \markdownRendererBlockHtmlCommentBeginPrototype}%
1439 \ExplSyntaxOn
1440 \seq_put_right:Nn
1441   \g_@@_renderers_seq
1442   { blockHtmlCommentBegin }
1443 \prop_put:Nnn
1444   \g_@@_renderer_arities_prop
1445   { blockHtmlCommentBegin }
1446   { 0 }
1447 \ExplSyntaxOff
1448 \def\markdownRendererBlockHtmlCommentEnd{%
1449   \markdownRendererBlockHtmlCommentEndPrototype}%
1450 \ExplSyntaxOn
1451 \seq_put_right:Nn
1452   \g_@@_renderers_seq
1453   { blockHtmlCommentEnd }
1454 \prop_put:Nnn
1455   \g_@@_renderer_arities_prop
1456   { blockHtmlCommentEnd }
1457   { 0 }
1458 \ExplSyntaxOff

```

**2.2.3.26 HTML Tag and Element Renderers** The `\markdownRendererInlineHtmlTag` macro represents an opening, closing, or empty inline HTML tag. This macro will only be produced, when the `html` option is enabled. The macro receives a single argument that corresponds to the contents of the HTML tag.

The `\markdownRendererInputBlockHtmlElement` macro represents a block HTML element. This macro will only be produced, when the `html` option is enabled. The macro receives a single argument that filename of a file containing the contents of the HTML element.



```

1459 \def\markdownRendererInlineHtmlTag{%
1460   \markdownRendererInlineHtmlTagPrototype}%
1461 \ExplSyntaxOn
1462 \seq_put_right:Nn
1463   \g_@@_renderers_seq
1464   { inlineHtmlTag }
1465 \prop_put:Nnn
1466   \g_@@_renderer_arities_prop
1467   { inlineHtmlTag }
1468   { 1 }
1469 \ExplSyntaxOff
1470 \def\markdownRendererInputBlockHtmlElement{%
1471   \markdownRendererInputBlockHtmlElementPrototype}%
1472 \ExplSyntaxOn
1473 \seq_put_right:Nn
1474   \g_@@_renderers_seq
1475   { inputBlockHtmlElement }
1476 \prop_put:Nnn
1477   \g_@@_renderer_arities_prop
1478   { inputBlockHtmlElement }
1479   { 1 }
1480 \ExplSyntaxOff

```

**2.2.3.27 Attribute Renderers** The following macros are only produced, when the `headerAttributes` option is enabled.

`\markdownRendererAttributeIdentifier` represents the  $\langle identifier \rangle$  of a markdown element (`id="⟨identifier⟩"` in HTML and `#⟨identifier⟩` in Markdown's `headerAttributes` syntax extension). The macro receives a single attribute that corresponds to the  $\langle identifier \rangle$ .

`\markdownRendererAttributeName` represents the  $\langle class name \rangle$  of a markdown element (`class="⟨class name⟩ ..."` in HTML and `.⟨class name⟩` in Markdown's `headerAttributes` syntax extension). The macro receives a single attribute that corresponds to the  $\langle class name \rangle$ .

`\markdownRendererAttributeKeyValue` represents a HTML attribute in the form  $\langle key \rangle = \langle value \rangle$  that is neither an identifier nor a class name. The macro receives two attributes that correspond to the  $\langle key \rangle$  and the  $\langle value \rangle$ , respectively.

```

1481 \def\markdownRendererAttributeIdentifier{%
1482   \markdownRendererAttributeIdentifierPrototype}%
1483 \ExplSyntaxOn
1484 \seq_put_right:Nn
1485   \g_@@_renderers_seq
1486   { attributeIdentifier }
1487 \prop_put:Nnn
1488   \g_@@_renderer_arities_prop
1489   { attributeIdentifier }

```

```

1490 { 1 }
1491 \ExplSyntaxOff
1492 \def\markdownRendererAttributeClassName{%
1493 \markdownRendererAttributeClassNamePrototype}%
1494 \ExplSyntaxOn
1495 \seq_put_right:Nn
1496 \g_@@_renderers_seq
1497 { attributeClassName }
1498 \prop_put:Nnn
1499 \g_@@_renderer_arities_prop
1500 { attributeClassName }
1501 { 1 }
1502 \ExplSyntaxOff
1503 \def\markdownRendererAttributeKeyValue{%
1504 \markdownRendererAttributeKeyValuePrototype}%
1505 \ExplSyntaxOn
1506 \seq_put_right:Nn
1507 \g_@@_renderers_seq
1508 { attributeKeyValue }
1509 \prop_put:Nnn
1510 \g_@@_renderer_arities_prop
1511 { attributeKeyValue }
1512 { 2 }
1513 \ExplSyntaxOff

```

**2.2.3.28 Header Attribute Context Renderers** The following macros are only produced, when the `headerAttributes` option is enabled.

The `\markdownRendererHeaderAttributeContextBegin` and `\markdownRendererHeaderAttributeContextEnd` macros represent the beginning and the end of a section in which the attributes of a heading apply. The macros receive no arguments.

```

1514 \def\markdownRendererHeaderAttributeContextBegin{%
1515 \markdownRendererHeaderAttributeContextBeginPrototype}%
1516 \ExplSyntaxOn
1517 \seq_put_right:Nn
1518 \g_@@_renderers_seq
1519 { headerAttributeContextBegin }
1520 \prop_put:Nnn
1521 \g_@@_renderer_arities_prop
1522 { headerAttributeContextBegin }
1523 { 0 }
1524 \ExplSyntaxOff
1525 \def\markdownRendererHeaderAttributeContextEnd{%
1526 \markdownRendererHeaderAttributeContextEndPrototype}%
1527 \ExplSyntaxOn
1528 \seq_put_right:Nn
1529 \g_@@_renderers_seq

```

```

1530 { headerAttributeContextEnd }
1531 \prop_put:Nnn
1532 \g_@@_renderer_arities_prop
1533 { headerAttributeContextEnd }
1534 { 0 }
1535 \ExplSyntaxOff

```

## 2.2.4 Token Renderer Prototypes

**2.2.4.1 YAML Metadata Renderer Prototypes** By default, the renderer prototypes for YAML metadata provide a high-level interface that can be programmed using the `markdown/jekyllData` key-values from the `l3keys` module of the  $\LaTeX$ 3 kernel.

```

1536 \ExplSyntaxOn
1537 \keys_define:nn
1538 { markdown/jekyllData }
1539 { }
1540 \ExplSyntaxOff

```

The following  $\TeX$  macros provide definitions for the token renderers (see Section 2.2.3) that have not been redefined by the user. These macros are intended to be redefined by macro package authors who wish to provide sensible default token renderers. They are also redefined by the  $\LaTeX$  and  $\ConTeXt$  implementations (see sections 3.3 and 3.4).

```

1541 \def\markdownRendererAttributeIdentifierPrototype#1{}%
1542 \def\markdownRendererAttributeClassNamePrototype#1{}%
1543 \def\markdownRendererAttributeKeyValuePrototype#1#2{}%
1544 \def\markdownRendererDocumentBeginPrototype{}%
1545 \def\markdownRendererDocumentEndPrototype{}%
1546 \def\markdownRendererInterblockSeparatorPrototype{}%
1547 \def\markdownRendererLineBreakPrototype{}%
1548 \def\markdownRendererEllipsisPrototype{}%
1549 \def\markdownRendererHeaderAttributeContextBeginPrototype{}%
1550 \def\markdownRendererHeaderAttributeContextEndPrototype{}%
1551 \def\markdownRendererNbspPrototype{}%
1552 \def\markdownRendererLeftBracePrototype{}%
1553 \def\markdownRendererRightBracePrototype{}%
1554 \def\markdownRendererDollarSignPrototype{}%
1555 \def\markdownRendererPercentSignPrototype{}%
1556 \def\markdownRendererAmpersandPrototype{}%
1557 \def\markdownRendererUnderscorePrototype{}%
1558 \def\markdownRendererHashPrototype{}%
1559 \def\markdownRendererCircumflexPrototype{}%
1560 \def\markdownRendererBackslashPrototype{}%
1561 \def\markdownRendererTildePrototype{}%
1562 \def\markdownRendererPipePrototype{}%
1563 \def\markdownRendererCodeSpanPrototype#1{}%

```

```

1564 \def\markdownRendererLinkPrototype#1#2#3#4{}%
1565 \def\markdownRendererImagePrototype#1#2#3#4{}%
1566 \def\markdownRendererContentBlockPrototype#1#2#3#4{}%
1567 \def\markdownRendererContentBlockOnlineImagePrototype#1#2#3#4{}%
1568 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{}%
1569 \def\markdownRendererUlBeginPrototype{}%
1570 \def\markdownRendererUlBeginTightPrototype{}%
1571 \def\markdownRendererUlItemPrototype{}%
1572 \def\markdownRendererUlItemEndPrototype{}%
1573 \def\markdownRendererUlEndPrototype{}%
1574 \def\markdownRendererUlEndTightPrototype{}%
1575 \def\markdownRendererOlBeginPrototype{}%
1576 \def\markdownRendererOlBeginTightPrototype{}%
1577 \def\markdownRendererOlItemPrototype{}%
1578 \def\markdownRendererOlItemWithNumberPrototype#1{}%
1579 \def\markdownRendererOlItemEndPrototype{}%
1580 \def\markdownRendererOlEndPrototype{}%
1581 \def\markdownRendererOlEndTightPrototype{}%
1582 \def\markdownRendererDlBeginPrototype{}%
1583 \def\markdownRendererDlBeginTightPrototype{}%
1584 \def\markdownRendererDlItemPrototype#1{}%
1585 \def\markdownRendererDlItemEndPrototype{}%
1586 \def\markdownRendererDlDefinitionBeginPrototype{}%
1587 \def\markdownRendererDlDefinitionEndPrototype{}%
1588 \def\markdownRendererDlEndPrototype{}%
1589 \def\markdownRendererDlEndTightPrototype{}%
1590 \def\markdownRendererEmphasisPrototype#1{}%
1591 \def\markdownRendererStrongEmphasisPrototype#1{}%
1592 \def\markdownRendererBlockQuoteBeginPrototype{}%
1593 \def\markdownRendererBlockQuoteEndPrototype{}%
1594 \def\markdownRendererInputVerbatimPrototype#1{}%
1595 \def\markdownRendererInputFencedCodePrototype#1#2{}%
1596 \def\markdownRendererJekyllDataBeginPrototype{}%
1597 \def\markdownRendererJekyllDataEndPrototype{}%
1598 \def\markdownRendererHeadingOnePrototype#1{}%
1599 \def\markdownRendererHeadingTwoPrototype#1{}%
1600 \def\markdownRendererHeadingThreePrototype#1{}%
1601 \def\markdownRendererHeadingFourPrototype#1{}%
1602 \def\markdownRendererHeadingFivePrototype#1{}%
1603 \def\markdownRendererHeadingSixPrototype#1{}%
1604 \def\markdownRendererHorizontalRulePrototype{}%
1605 \def\markdownRendererFootnotePrototype#1{}%
1606 \def\markdownRendererCitePrototype#1{}%
1607 \def\markdownRendererTextCitePrototype#1{}%
1608 \def\markdownRendererTablePrototype#1#2#3{}%
1609 \def\markdownRendererInlineHtmlCommentPrototype#1{}%
1610 \let\markdownRendererBlockHtmlCommentBeginPrototype=\iffalse

```

```

1611 \let\markdownRendererBlockHtmlCommentBegin=\iffalse
1612 \let\markdownRendererBlockHtmlCommentEndPrototype=\fi
1613 \let\markdownRendererBlockHtmlCommentEnd=\fi
1614 \def\markdownRendererInlineHtmlTagPrototype#1{}%
1615 \def\markdownRendererInputBlockHtmlElementPrototype#1{}%
1616 \def\markdownRendererTickedBoxPrototype{}%
1617 \def\markdownRendererHalfTickedBoxPrototype{}%
1618 \def\markdownRendererUntickedBoxPrototype{}%

```

### 2.2.5 Logging Facilities

The `\markdownInfo`, `\markdownWarning`, and `\markdownError` macros perform logging for the Markdown package. Their first argument specifies the text of the info, warning, or error message. The `\markdownError` macro receives a second argument that provides a help text. You may redefine these macros to redirect and process the info, warning, and error messages.

### 2.2.6 Miscellanea

The `\markdownMakeOther` macro is used by the package, when a  $\TeX$  engine that does not support direct Lua access is starting to buffer a text. The plain  $\TeX$  implementation changes the category code of plain  $\TeX$  special characters to *other*, but there may be other active characters that may break the output. This macro should temporarily change the category of these to *other*.

```
1619 \let\markdownMakeOther\relax
```

The `\markdownReadAndConvert` macro implements the `\markdownBegin` macro. The first argument specifies the token sequence that will terminate the markdown input (`\markdownEnd` in the instance of the `\markdownBegin` macro) when the plain  $\TeX$  special characters have had their category changed to *other*. The second argument specifies the token sequence that will actually be inserted into the document, when the ending token sequence has been found.

```
1620 \let\markdownReadAndConvert\relax
```

```
1621 \begingroup
```

Locally swap the category code of the backslash symbol (`\`) with the pipe symbol (`|`). This is required in order that all the special symbols in the first argument of the `markdownReadAndConvert` macro have the category code *other*.

```

1622 \catcode`\|=0\catcode`\=12%
1623 |gdef|markdownBegin{%
1624   |markdownReadAndConvert{\markdownEnd}%
1625   { |markdownEnd}}%
1626 |endgroup

```

The macro is exposed in the interface, so that the user can create their own markdown environments. Due to the way the arguments are passed to Lua (see Section 3.2.7),

the first argument may not contain the string `]]` (regardless of the category code of the bracket symbol (`]`)).

The `\markdownMode` macro specifies how the plain  $\TeX$  implementation interfaces with the Lua interface. The valid values and their meaning are as follows:

- `0` – Shell escape via the 18 output file stream
- `1` – Shell escape via the Lua `os.execute` method
- `2` – Direct Lua access

By defining the macro, the user can coerce the package to use a specific mode. If the user does not define the macro prior to loading the plain  $\TeX$  implementation, the correct value will be automatically detected. The outcome of changing the value of `\markdownMode` after the implementation has been loaded is undefined.

```
1627 \ifx\markdownMode\undefined
1628   \ifx\directlua\undefined
1629     \def\markdownMode{0}%
1630   \else
1631     \def\markdownMode{2}%
1632   \fi
1633 \fi
```

The following macros are no longer a part of the plain  $\TeX$  interface and are only defined for backwards compatibility:

```
1634 \def\markdownLuaRegisterIBCcallback#1{\relax}%
1635 \def\markdownLuaUnregisterIBCcallback#1{\relax}%
```

## 2.3 $\LaTeX$ Interface

The  $\LaTeX$  interface provides  $\LaTeX$  environments for the typesetting of markdown input from within  $\LaTeX$ , facilities for setting Lua interface options (see Section 2.1.2) used during the conversion from markdown to plain  $\TeX$ , and facilities for changing the way markdown tokens are rendered. The rest of the interface is inherited from the plain  $\TeX$  interface (see Section 2.2).

The  $\LaTeX$  interface is implemented by the `markdown.sty` file, which can be loaded from the  $\LaTeX$  document preamble as follows:

```
\usepackage[<options>]{markdown}
```

where `<options>` are the  $\LaTeX$  interface options (see Section 2.3.2). Note that `<options>` inside the `\usepackage` macro may not set the `markdownRenderers` (see Section 2.3.2.5) and `markdownRendererPrototypes` (see Section 2.3.2.6) keys. This limitation is due to the way  $\LaTeX 2_{\epsilon}$  parses package options.

### 2.3.1 Typesetting Markdown

The interface exposes the `markdown` and `markdown*`  $\LaTeX$  environments, and redefines the `\markdownInput` command.

The `markdown` and `markdown*` L<sup>A</sup>T<sub>E</sub>X environments are used to typeset markdown document fragments. The starred version of the `markdown` environment accepts L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.2) as its only argument. These options will only influence this markdown document fragment.

```
1636 \newenvironment{markdown}\relax\relax
1637 \newenvironment{markdown*}[1]\relax\relax
```

You may prepend your own code to the `\markdown` macro and append your own code to the `\endmarkdown` macro to produce special effects before and after the `markdown` L<sup>A</sup>T<sub>E</sub>X environment (and likewise for the starred version).

Note that the `markdown` and `markdown*` L<sup>A</sup>T<sub>E</sub>X environments are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros exposed by the plain T<sub>E</sub>X interface.

The following example L<sup>A</sup>T<sub>E</sub>X code showcases the usage of the `markdown` and `markdown*` environments:

<pre>\documentclass{article} \usepackage{markdown} \begin{document} % ... \begin{markdown} _Hello_ <b>world</b> ... \end{markdown} % ... \end{document}</pre>	<pre>\documentclass{article} \usepackage{markdown} \begin{document} % ... \begin{markdown*}{smartEllipses} _Hello_ <b>world</b> ... \end{markdown*} % ... \end{document}</pre>
---	--

The `\markdownInput` macro accepts a single mandatory parameter containing the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain T<sub>E</sub>X. Unlike the `\markdownInput` macro provided by the plain T<sub>E</sub>X interface, this macro also accepts L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.2) as its optional argument. These options will only influence this markdown document.

The following example L<sup>A</sup>T<sub>E</sub>X code showcases the usage of the `\markdownInput` macro:

<pre>\documentclass{article} \usepackage{markdown} \begin{document} \markdownInput[smartEllipses]{hello.md} \end{document}</pre>
--

## 2.3.2 Options

The  $\LaTeX$  options are represented by a comma-delimited list of  $\langle key \rangle = \langle value \rangle$  pairs. For boolean options, the  $= \langle value \rangle$  part is optional, and  $\langle key \rangle$  will be interpreted as  $\langle key \rangle = \text{true}$  if the  $= \langle value \rangle$  part has been omitted.

Except for the `plain` option described in Section 2.3.2.1, and the  $\LaTeX$  themes described in Section 2.3.2.2, and the  $\LaTeX$  setup snippets described in Section 2.3.2.3,  $\LaTeX$  options map directly to the options recognized by the plain  $\TeX$  interface (see Section 2.2.2) and to the markdown token renderers and their prototypes recognized by the plain  $\TeX$  interface (see Sections 2.2.3 and 2.2.4).

The  $\LaTeX$  options may be specified when loading the  $\LaTeX$  package, when using the `markdown*`  $\LaTeX$  environment or the `\markdownInput` macro (see Section 2.3), or via the `\markdownSetup` macro. The `\markdownSetup` macro receives the options to set up as its only argument:

```
1638 \newcommand\markdownSetup[1]{%
1639   \setkeys{markdownOptions}{#1}}%
```

We may also store  $\LaTeX$  options as *setup snippets* and invoke them later using the `\markdownSetupSnippet` macro. The `\markdownSetupSnippet` macro receives two arguments: the name of the setup snippet and the options to store:

```
1640 \newcommand\markdownSetupSnippet[2]{%
1641   \markdownIfSnippetExists{#1}%
1642   {%
1643     \markdownWarning
1644     {Redefined setup snippet \markdownLaTeXThemeName#1}%
1645     \csname markdownLaTeXSetupSnippet%
1646       \markdownLaTeXThemeName#1\endcsname={#2}%
1647   }{%
1648     \newtoks\next
1649     \next={#2}%
1650     \expandafter\let\csname markdownLaTeXSetupSnippet%
1651       \markdownLaTeXThemeName#1\endcsname=\next
1652   }}%
```

To decide whether a setup snippet exists, we can use the `\markdownIfSnippetExists` macro:

```
1653 \newcommand\markdownIfSnippetExists[3]{%
1654   \@ifundefined
1655     {markdownLaTeXSetupSnippet\markdownLaTeXThemeName#1}%
1656     {#3}{#2}}%
```

See Section 2.3.2.2 for information on interactions between setup snippets and  $\LaTeX$  themes. See Section 2.3.2.3 for information about invoking the stored setup snippets.

**2.3.2.1 No default token renderer prototypes** Default token renderer prototypes require  $\LaTeX$  packages that may clash with other packages used in a document.



Additionally, if we redefine token renderers and renderer prototypes ourselves, the default definitions will bring no benefit to us. Using the `plain` package option, we can keep the default definitions from the plain  $\TeX$  implementation (see Section 3.2.3) and prevent the soft  $\LaTeX$  prerequisites in Section 1.1.3 from being loaded:

```
\usepackage[plain]{markdown}
```

```
1657 \newif\ifmarkdownLaTeXPlain
1658   \markdownLaTeXPlainfalse
1659 \define@key{markdownOptions}{plain}[true]{%
1660   \ifmarkdownLaTeXLoaded
1661     \markdownWarning
1662     {The plain option must be specified when loading the package}%
1663   \else
1664     \markdownLaTeXPlaintrue
1665   \fi}
```

**2.3.2.2  $\LaTeX$  themes** User-contributed  $\LaTeX$  themes for the Markdown package provide a domain-specific interpretation of some Markdown tokens. Similarly to  $\LaTeX$  packages, themes allow the authors to achieve a specific look and other high-level goals without low-level programming.

The  $\LaTeX$  option with key `theme` loads a  $\LaTeX$  package (further referred to as *a theme*) named `markdowntheme<munged theme name>.sty`, where the *munged theme name* is the *theme name* after a substitution of all forward slashes (/) for an underscore (\_), the theme name is a value that is *qualified* and contains no underscores, and a value is qualified if and only if it contains at least one forward slash. Themes are inspired by the Beamer  $\LaTeX$  package, which provides similar functionality with its `\usetheme` macro [6, Section 15.1].

Theme names must be qualified to minimize naming conflicts between different themes intended for a single  $\LaTeX$  document class or for a single  $\LaTeX$  package. The preferred format of a theme name is `<theme author>/<target  $\LaTeX$  document class or package>/<private naming scheme>`, where the *private naming scheme* may contain additional forward slashes. For example, a theme by a user `witiko` for the MU theme of the Beamer document class may have the name `witiko/beamer/MU`.

Theme names are munged, because  $\LaTeX$  packages are identified only by their filenames, not by their pathnames. [7] Therefore, we can't store the qualified theme names directly using directories, but we must encode the individual segments of the qualified theme in the filename. For example, loading a theme named `witiko/beamer/MU` would load a  $\LaTeX$  package named `markdownthemewitiko_beamer_MU.sty`.

If the  $\LaTeX$  option with key `theme` is (repeatedly) specified in the `\usepackage` macro, the loading of the theme(s) will be postponed in first-in-first-out order until

after the Markdown  $\LaTeX$  package has been loaded. Otherwise, the theme(s) will be loaded immediately. For example, there is a theme named `witiko/dot`, which typesets fenced code blocks with the `dot` infostring as images of directed graphs rendered by the Graphviz tools. The following code would first load the Markdown package, then the `markdownthemewitiko_beamer_MU.sty`  $\LaTeX$  package, and finally the `markdownthemewitiko_dot.sty`  $\LaTeX$  package:

```
\usepackage[
  theme = witiko/beamer/MU,
  theme = witiko/dot,
]{markdown}
```

```
1666 \newif\ifmarkdownLaTeXLoaded
1667 \markdownLaTeXLoadedfalse
1668 \AtEndOfPackage{\markdownLaTeXLoadedtrue}
1669 \define@key{markdownOptions}{theme}{%
1670 \IfSubStr{#1}{/}{%}{%
1671 \markdownError
1672 {Won't load theme with unqualified name #1}%
1673 {Theme names must contain at least one forward slash}}%
1674 \StrSubstitute{#1}{/}{_}[\markdownLaTeXThemePackageName]%
1675 \edef\markdownLaTeXThemePackageName{%
1676 markdowntheme\markdownLaTeXThemePackageName}%
1677 \expandafter\markdownLaTeXThemeLoad\expandafter{%
1678 \markdownLaTeXThemePackageName}{#1/}}%
```

The  $\LaTeX$  themes have a useful synergy with the setup snippets (see Section 2.3.2): To make it less likely that different themes will define setup snippets with the same name, we will prepend  $\langle$ *theme name* $\rangle/$  before the snippet name and use the result as the snippet name. For example, if the `witiko/dot` theme defines the `product` setup snippet, the setup snippet will be available under the name `witiko/dot/product`. Due to limitations of  $\LaTeX$ , themes may not be loaded after the beginning of a  $\LaTeX$  document.

```
1679 \@onlypreamble\KV@markdownOptions@theme
```

Example themes provided with the Markdown package include:

**witiko/dot** A theme that typesets fenced code blocks with the `dot ...` infostring as images of directed graphs rendered by the Graphviz tools. The right tail of the infostring is used as the image title.

```
\documentclass{article}
\usepackage[theme=witiko/dot]{markdown}
\setkeys{Gin}{
  width = \columnwidth,
```

```

    height = 0.65\paperheight,
    keepaspectratio}
\begin{document}
\begin{markdown}
... dot Various formats of mathematical formulae
digraph tree {
    margin = 0;
    rankdir = "LR";

    latex -> pmml;
    latex -> cmml;
    pmml -> slt;
    cmml -> opt;
    cmml -> prefix;
    cmml -> infix;
    pmml -> mterms [style=dashed];
    cmml -> mterms;

    latex [label = "LaTeX"];
    pmml [label = "Presentation MathML"];
    cmml [label = "Content MathML"];
    slt [label = "Symbol Layout Tree"];
    opt [label = "Operator Tree"];
    prefix [label = "Prefix"];
    infix [label = "Infix"];
    mterms [label = "M-Terms"];
}
...
\end{markdown}
\end{document}

```

Typesetting the above document produces the output shown in Figure 4.

The theme requires a Unix-like operating system with GNU Diffutils and Graphviz installed. The theme also requires shell access unless the `\markdownOptionFrozenCache` plain T<sub>E</sub>X option is enabled.

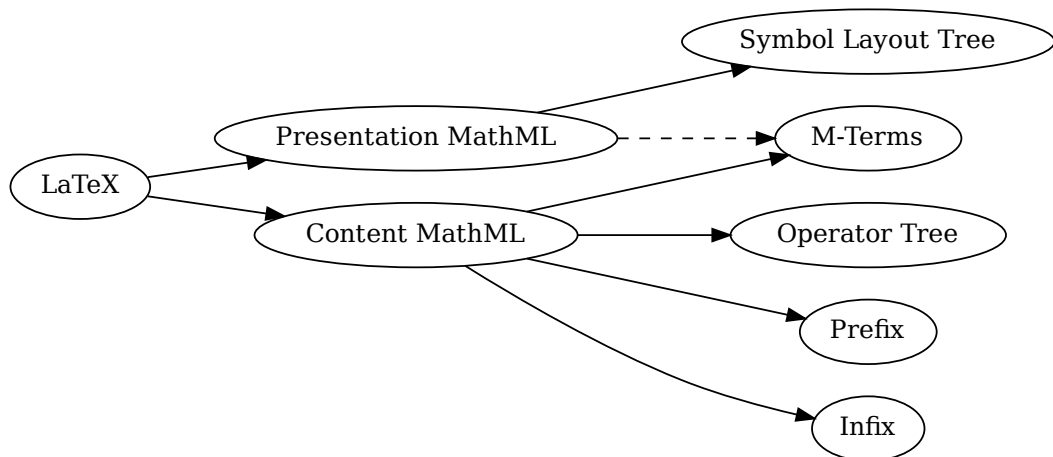
1680 \ProvidesPackage{markdownthemewitiko\_dot}[2021/03/09]%

**witiko/graphicx/http** A theme that adds support for downloading images whose URL has the http or https protocol.

```

\documentclass{article}
\usepackage[theme=witiko/graphicx/http]{markdown}

```



**Figure 4: Various formats of mathematical formulae**

```

\begin{document}
\begin{markdown}
! [img] (https://github.com/witiko/markdown/raw/main/markdown.png
      "The banner of the Markdown package")
\end{markdown}
\end{document}
  
```

Typesetting the above document produces the output shown in Figure 5. The theme requires the catchfile  $\LaTeX$  package and a Unix-like operating system with GNU Coreutils `md5sum` and either GNU Wget or `cURL` installed. The theme also requires shell access unless the `\markdownOptionFrozenCache` plain  $\TeX$  option is enabled.

```
1681 \ProvidesPackage{markdownthemewitiko_graphicx_http}[2021/03/22]%
```

**witiko/tilde** A theme that makes tilde (`~`) always typeset the non-breaking space even when the `hybrid` Lua option is `false`.

```

\documentclass{article}
\usepackage[theme=witiko/tilde]{markdown}
\begin{document}
\begin{markdown}
Bartel~Leendert van~der~Waerden
\end{markdown}
\end{document}
  
```

```

\documentclass{book}
\usepackage{markdown}
\markdownSetup{pipeTables,tableCaptions}
\begin{document}
\begin{markdown}
Introduction
=====
## Section
### Subsection
Hello *Markdown*!

| Right | Left | Default | Center |
|-----:|:-----|-----:|:-----:|
| 12    | 12   | 12      | 12     |
| 123   | 123  | 123     | 123    |
| 1     | 1    | 1       | 1      |

: Table
\end{markdown}
\end{document}

```



# Chapter 1

## Introduction

1.1 Section  
1.1.1 Subsection  
Hello *Markdown!*

Right	Left	Default	Center
12	12	12	12
123	123	123	123
1	1	1	1

Table 1.1: Table

**Figure 5: The banner of the Markdown package**

Typesetting the above document produces the following text: “Bartel Leendert van der Waerden”.

```
1682 \ProvidesPackage{markdownthemewitiko_tilde}[2021/03/22]%
```

Please, see Section 3.3.2.1 for implementation details of the example themes.

**2.3.2.3 L<sup>A</sup>T<sub>E</sub>X setup snippets** The L<sup>A</sup>T<sub>E</sub>X option with key `snippet` invokes a snippet named `<value>`:

```

1683 \define@key{markdownOptions}{snippet}{%
1684   \markdownIfSnippetExists{#1}%
1685   {%
1686     \expandafter\markdownSetup\expandafter{%
1687       \the\csname markdownLaTeXSetupSnippet%
1688         \markdownLaTeXThemeName#1\endcsname}%
1689   }{%
1690     \markdownError
1691     {Can't invoke setup snippet #1}%
1692     {The setup snippet is undefined}%
1693   }%
1694 }%

```

Here is how we can use setup snippets to store options and invoke them later:

```

\markdownSetupSnippet{romanNumerals}{
  renderers = {
    olItemWithNumber = {\item[\romannumeral#1\relax.]},
  },
}
\begin{markdown}

```

The following ordered list will be preceded by arabic numerals:

1. wahid
2. aithnayn

```

\end{markdown}
\begin{markdown*}{snippet=romanNumerals}

```

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```

\end{markdown*}

```

**2.3.2.4 Plain T<sub>E</sub>X Interface Options** The following options map directly to the option macros exposed by the plain T<sub>E</sub>X interface (see Section 2.2.2).

```

1695 \define@key{markdownOptions}{helperScriptFileName}{%
1696   \def\markdownOptionHelperScriptFileName{#1}}%
1697 \define@key{markdownOptions}{inputTempFileName}{%
1698   \def\markdownOptionInputTempFileName{#1}}%
1699 \define@key{markdownOptions}{outputTempFileName}{%
1700   \def\markdownOptionOutputTempFileName{#1}}%
1701 \define@key{markdownOptions}{errorTempFileName}{%
1702   \def\markdownOptionErrorTempFileName{#1}}%
1703 \define@key{markdownOptions}{cacheDir}{%
1704   \def\markdownOptionCacheDir{#1}}%
1705 \define@key{markdownOptions}{outputDir}{%
1706   \def\markdownOptionOutputDir{#1}}%
1707 \define@key{markdownOptions}{blankBeforeBlockquote}[true]{%
1708   \def\markdownOptionBlankBeforeBlockquote{#1}}%
1709 \define@key{markdownOptions}{blankBeforeCodeFence}[true]{%
1710   \def\markdownOptionBlankBeforeCodeFence{#1}}%
1711 \define@key{markdownOptions}{blankBeforeHeading}[true]{%
1712   \def\markdownOptionBlankBeforeHeading{#1}}%

```

```

1713 \define@key{markdownOptions}{breakableBlockquotes}[true]{%
1714   \def\markdownOptionBreakableBlockquotes{#1}}%
1715 \define@key{markdownOptions}{citations}[true]{%
1716   \def\markdownOptionCitations{#1}}%
1717 \define@key{markdownOptions}{citationNbsps}[true]{%
1718   \def\markdownOptionCitationNbsps{#1}}%
1719 \define@key{markdownOptions}{contentBlocks}[true]{%
1720   \def\markdownOptionContentBlocks{#1}}%
1721 \define@key{markdownOptions}{codeSpans}[true]{%
1722   \def\markdownOptionCodeSpans{#1}}%
1723 \define@key{markdownOptions}{contentBlocksLanguageMap}{%
1724   \def\markdownOptionContentBlocksLanguageMap{#1}}%
1725 \define@key{markdownOptions}{definitionLists}[true]{%
1726   \def\markdownOptionDefinitionLists{#1}}%
1727 \define@key{markdownOptions}{eagerCache}[true]{%
1728   \def\markdownOptionEagerCache{#1}}%
1729 \define@key{markdownOptions}{expectJekyllData}[true]{%
1730   \def\markdownOptionExpectJekyllData{#1}}%
1731 \define@key{markdownOptions}{footnotes}[true]{%
1732   \def\markdownOptionFootnotes{#1}}%
1733 \define@key{markdownOptions}{fencedCode}[true]{%
1734   \def\markdownOptionFencedCode{#1}}%
1735 \define@key{markdownOptions}{jekyllData}[true]{%
1736   \def\markdownOptionJekyllData{#1}}%
1737 \define@key{markdownOptions}{hardLineBreaks}[true]{%
1738   \def\markdownOptionHardLineBreaks{#1}}%
1739 \define@key{markdownOptions}{hashEnumerators}[true]{%
1740   \def\markdownOptionHashEnumerators{#1}}%
1741 \define@key{markdownOptions}{headerAttributes}[true]{%
1742   \def\markdownOptionHeaderAttributes{#1}}%
1743 \define@key{markdownOptions}{html}[true]{%
1744   \def\markdownOptionHtml{#1}}%
1745 \define@key{markdownOptions}{hybrid}[true]{%
1746   \def\markdownOptionHybrid{#1}}%
1747 \define@key{markdownOptions}{inlineFootnotes}[true]{%
1748   \def\markdownOptionInlineFootnotes{#1}}%
1749 \define@key{markdownOptions}{pipeTables}[true]{%
1750   \def\markdownOptionPipeTables{#1}}%
1751 \define@key{markdownOptions}{preserveTabs}[true]{%
1752   \def\markdownOptionPreserveTabs{#1}}%
1753 \define@key{markdownOptions}{relativeReferences}[true]{%
1754   \def\markdownOptionRelativeReferences{#1}}%
1755 \define@key{markdownOptions}{smartEllipses}[true]{%
1756   \def\markdownOptionSmartEllipses{#1}}%
1757 \define@key{markdownOptions}{shiftHeadings}{%
1758   \def\markdownOptionShiftHeadings{#1}}%
1759 \define@key{markdownOptions}{slice}{%

```

```

1760 \def\markdownOptionSlice{#1}}%
1761 \define@key{markdownOptions}{startNumber}[true]{%
1762 \def\markdownOptionStartNumber{#1}}%
1763 \define@key{markdownOptions}{stripIndent}[true]{%
1764 \def\markdownOptionStripIndent{#1}}%
1765 \define@key{markdownOptions}{tableCaptions}[true]{%
1766 \def\markdownOptionTableCaptions{#1}}%
1767 \define@key{markdownOptions}{taskLists}[true]{%
1768 \def\markdownOptionTaskLists{#1}}%
1769 \define@key{markdownOptions}{texComments}[true]{%
1770 \def\markdownOptionTexComments{#1}}%
1771 \define@key{markdownOptions}{tightLists}[true]{%
1772 \def\markdownOptionTightLists{#1}}%
1773 \define@key{markdownOptions}{underscores}[true]{%
1774 \def\markdownOptionUnderscores{#1}}%
1775 \define@key{markdownOptions}{stripPercentSigns}[true]{%
1776 \def\markdownOptionStripPercentSigns{#1}}%

```

The `\markdownOptionFinalizeCache` and `\markdownOptionFrozenCache` plain TeX options are exposed through L<sup>A</sup>T<sub>E</sub>X options with keys `finalizeCache` and `frozenCache`.

To ensure compatibility with the `minted` package [8, Section 5.1], which supports the `finalizcache` and `frozenscache` package options with similar semantics, the Markdown package also recognizes these as aliases and recognizes them as document class options. By passing `finalizcache` and `frozenscache` as document class options, you may conveniently control the behavior of both packages at once:

```

\documentclass[frozenscache]{article}
\usepackage{markdown,minted}
\begin{document}
\end{document}

```

We hope that other packages will support the `finalizcache` and `frozenscache` package options in the future, so that they can become a standard interface for preparing L<sup>A</sup>T<sub>E</sub>X document sources for distribution.

```

1777 \define@key{markdownOptions}{finalizeCache}[true]{%
1778 \def\markdownOptionFinalizeCache{#1}}%
1779 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
1780 \define@key{markdownOptions}{frozenCache}[true]{%
1781 \def\markdownOptionFrozenCache{#1}}%
1782 \DeclareOption{frozenscache}{\markdownSetup{frozenCache}}
1783 \define@key{markdownOptions}{frozenCacheFileName}{%
1784 \def\markdownOptionFrozenCacheFileName{#1}}%

```



The following example  $\LaTeX$  code showcases a possible configuration of plain  $\TeX$  interface options `\markdownOptionHybrid`, `\markdownOptionSmartEllipses`, and `\markdownOptionCacheDir`.

```
\markdownSetup{
  hybrid,
  smartEllipses,
  cacheDir = /tmp,
}
```

**2.3.2.5 Plain  $\TeX$  Markdown Token Renderers** The  $\LaTeX$  interface recognizes an option with the `renderers` key, whose value must be a list of options that map directly to the markdown token renderer macros exposed by the plain  $\TeX$  interface (see Section 2.2.3).

```
1785 \ExplSyntaxOn
1786 \cs_new:Nn \@@_latex_define_renderers:
1787   {
1788     \seq_map_function:NN
1789       \g_@@_renderers_seq
1790       \@@_latex_define_renderer:n
1791   }
1792 \cs_new:Nn \@@_latex_define_renderer:n
1793   {
1794     \tl_set:Nn
1795       \l_tmpb_tl
1796     %   TODO: Replace with \str_uppercase:n in TeX Live 2020.
1797       { \str_upper_case:n { #1 } }
1798     \tl_set:Nx
1799       \l_tmpa_tl
1800       {
1801         markdownRenderer
1802         \tl_head:f { \l_tmpb_tl }
1803         \tl_tail:n { #1 }
1804       }
1805     \prop_get:NnN
1806       \g_@@_renderer_arities_prop
1807       { #1 }
1808       \l_tmpb_tl
1809     \@@_latex_define_renderer:ncV
1810       { #1 }
1811       { \l_tmpa_tl }
1812       \l_tmpb_tl
1813   }
1814 \cs_new:Nn \@@_latex_define_renderer:nNn
1815   {
```

```

1816 \define@key
1817   { markdownRenderers }
1818   { #1 }
1819   {
1820     \cs_generate_from_arg_count:NNnn
1821     #2
1822     \cs_set:Npn
1823     { #3 }
1824     { ##1 }
1825   }
1826 }
1827 \cs_generate_variant:Nn
1828 \@@_latex_define_renderer:nNn
1829 { ncV }
1830 \ExplSyntaxOff

```

The following example  $\LaTeX$  code showcases a possible configuration of the `\markdownRendererLink` and `\markdownRendererEmphasis` markdown token renderers.

```

\markdownSetup{
  renderers = {
    link = {#4},           % Render links as the link title.
    emphasis = {\emph{#1}}, % Render emphasized text via \emph`.
  }
}

```

**2.3.2.6 Plain  $\TeX$  Markdown Token Renderer Prototypes** The  $\LaTeX$  interface recognizes an option with the `rendererPrototypes` key, whose value must be a list of options that map directly to the markdown token renderer prototype macros exposed by the plain  $\TeX$  interface (see Section 2.2.4).

```

1831 \ExplSyntaxOn
1832 \cs_new:Nn \@@_latex_define_renderer_prototypes:
1833   {
1834     \seq_map_function:NN
1835     \g_@@_renderers_seq
1836     \@@_latex_define_renderer_prototype:n
1837   }
1838 \cs_new:Nn \@@_latex_define_renderer_prototype:n
1839   {
1840     \tl_set:Nn
1841     \l_tmpb_tl
1842     % TODO: Replace with \str_uppercase:n in TeX Live 2020.
1843     { \str_upper_case:n { #1 } }
1844     \tl_set:Nx

```

```

1845     \l_tmpa_tl
1846     {
1847         markdownRenderer
1848         \tl_head:f { \l_tmpb_tl }
1849         \tl_tail:n { #1 }
1850         Prototype
1851     }
1852 \prop_get:NnN
1853 \g_@@_renderer_arities_prop
1854 { #1 }
1855 \l_tmpb_tl
1856 \@@_latex_define_renderer_prototype:ncV
1857 { #1 }
1858 { \l_tmpa_tl }
1859 \l_tmpb_tl
1860 }
1861 \cs_new:Nn \@@_latex_define_renderer_prototype:nNn
1862 {
1863     \define@key
1864     { markdownRendererPrototypes }
1865     { #1 }
1866     {
1867         \cs_generate_from_arg_count:NNnn
1868         #2
1869         \cs_set:Npn
1870         { #3 }
1871         { ##1 }
1872     }
1873 }
1874 \cs_generate_variant:Nn
1875 \@@_latex_define_renderer_prototype:nNn
1876 { ncV }
1877 \ExplSyntaxOff

```

The following example L<sup>A</sup>T<sub>E</sub>X code showcases a possible configuration of the `\markdownRendererImagePrototype` and `\markdownRendererCodeSpanPrototype` markdown token renderer prototypes.

```

\markdownSetup{
  rendererPrototypes = {
    image = {\includegraphics{#2}},
    codeSpan = {\texttt{#1}}, % Render inline code via \texttt.
  }
}

```

## 2.4 ConT<sub>E</sub>Xt Interface

The ConT<sub>E</sub>Xt interface provides a start-stop macro pair for the typesetting of mark-down input from within ConT<sub>E</sub>Xt. The rest of the interface is inherited from the plain T<sub>E</sub>X interface (see Section 2.2).

```
1878 \writestatus{loading}{ConTeXt User Module / markdown}%
1879 \startmodule[markdown]
1880 \unprotect
```

The ConT<sub>E</sub>Xt interface is implemented by the `t-markdown.tex` ConT<sub>E</sub>Xt module file that can be loaded as follows:

```
\usemodule[t][markdown]
```

It is expected that the special plain T<sub>E</sub>X characters have the expected category codes, when `\inputting` the file.

### 2.4.1 Typesetting Markdown

The interface exposes the `\startmarkdown` and `\stopmarkdown` macro pair for the typesetting of a markdown document fragment.

```
1881 \let\startmarkdown\relax
1882 \let\stopmarkdown\relax
```

You may prepend your own code to the `\startmarkdown` macro and redefine the `\stopmarkdown` macro to produce special effects before and after the markdown block.

Note that the `\startmarkdown` and `\stopmarkdown` macros are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros exposed by the plain T<sub>E</sub>X interface.

The following example ConT<sub>E</sub>Xt code showcases the usage of the `\startmarkdown` and `\stopmarkdown` macros:

```
\usemodule[t][markdown]
\starttext
\startmarkdown
_Hello_ world ...
\stopmarkdown
\stoptext
```

## 3 Implementation

This part of the documentation describes the implementation of the interfaces exposed by the package (see Section 2) and is aimed at the developers of the package, as well as the curious users.

Figure 1 shows the high-level structure of the Markdown package: The translation from markdown to  $\text{T}_{\text{E}}\text{X}$  *token renderers* is performed by the Lua layer. The plain  $\text{T}_{\text{E}}\text{X}$  layer provides default definitions for the token renderers. The  $\text{\LaTeX}$  and  $\text{ConT}_{\text{E}}\text{Xt}$  layers correct idiosyncrasies of the respective  $\text{T}_{\text{E}}\text{X}$  formats, and provide format-specific default definitions for the token renderers.

### 3.1 Lua Implementation

The Lua implementation implements `writer` and `reader` objects that provide the conversion from markdown to plain  $\text{T}_{\text{E}}\text{X}$ .

The Lunamark Lua module implements writers for the conversion to various other formats, such as DocBook, Groff, or HTML. These were stripped from the module and the remaining markdown reader and plain  $\text{T}_{\text{E}}\text{X}$  writer were hidden behind the converter functions exposed by the Lua interface (see Section 2.1).

```
1883 local upper, gsub, format, length =
1884   string.upper, string.gsub, string.format, string.len
1885 local concat = table.concat
1886 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, any =
1887   lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
1888   lpeg.Cmt, lpeg.Cc, lpeg.Ct, lpeg.B, lpeg.Cs, lpeg.P(1)
```

#### 3.1.1 Utility Functions

This section documents the utility functions used by the plain  $\text{T}_{\text{E}}\text{X}$  writer and the markdown reader. These functions are encapsulated in the `util` object. The functions were originally located in the `lunamark/util.lua` file in the Lunamark Lua module.

```
1889 local util = {}
```

The `util.err` method prints an error message `msg` and exits. If `exit_code` is provided, it specifies the exit code. Otherwise, the exit code will be 1.

```
1890 function util.err(msg, exit_code)
1891   io.stderr:write("markdown.lua: " .. msg .. "\n")
1892   os.exit(exit_code or 1)
1893 end
```

The `util.cache` method computes the digest of `string` and `salt`, adds the `suffix` and looks into the directory `dir`, whether a file with such a name exists. If it does not, it gets created with `transform(string)` as its content. The filename is then returned.

```
1894 function util.cache(dir, string, salt, transform, suffix)
1895   local digest = md5.sumhexa(string .. (salt or ""))
1896   local name = util.pathname(dir, digest .. suffix)
1897   local file = io.open(name, "r")
1898   if file == nil then -- If no cache entry exists, then create a new one.
```

```

1899     local file = assert(io.open(name, "w"),
1900         [[could not open file "]] .. name .. [[ for writing]])
1901     local result = string
1902     if transform ~= nil then
1903         result = transform(result)
1904     end
1905     assert(file:write(result))
1906     assert(file:close())
1907 end
1908 return name
1909 end

```

The `util.table_copy` method creates a shallow copy of a table `t` and its metatable.

```

1910 function util.table_copy(t)
1911     local u = { }
1912     for k, v in pairs(t) do u[k] = v end
1913     return setmetatable(u, getmetatable(t))
1914 end

```

The `util.expand_tabs_in_line` expands tabs in string `s`. If `tabstop` is specified, it is used as the tab stop width. Otherwise, the tab stop width of 4 characters is used. The method is a copy of the tab expansion algorithm from Ierusalimsky [9, Chapter 21].

```

1915 function util.expand_tabs_in_line(s, tabstop)
1916     local tab = tabstop or 4
1917     local corr = 0
1918     return (s:gsub("\t", function(p)
1919         local sp = tab - (p - 1 + corr) % tab
1920         corr = corr - 1 + sp
1921         return string.rep(" ", sp)
1922     end))
1923 end

```

The `util.walk` method walks a rope `t`, applying a function `f` to each leaf element in order. A rope is an array whose elements may be ropes, strings, numbers, or functions. If a leaf element is a function, call it and get the return value before proceeding.

```

1924 function util.walk(t, f)
1925     local typ = type(t)
1926     if typ == "string" then
1927         f(t)
1928     elseif typ == "table" then
1929         local i = 1
1930         local n
1931         n = t[i]
1932         while n do
1933             util.walk(n, f)
1934             i = i + 1

```

```

1935     n = t[i]
1936     end
1937 elseif typ == "function" then
1938     local ok, val = pcall(t)
1939     if ok then
1940         util.walk(val,f)
1941     end
1942 else
1943     f(tostring(t))
1944 end
1945 end

```

The `util.flatten` method flattens an array `ary` that does not contain cycles and returns the result.

```

1946 function util.flatten(ary)
1947     local new = {}
1948     for _,v in ipairs(ary) do
1949         if type(v) == "table" then
1950             for _,w in ipairs(util.flatten(v)) do
1951                 new[#new + 1] = w
1952             end
1953         else
1954             new[#new + 1] = v
1955         end
1956     end
1957     return new
1958 end

```

The `util.rope_to_string` method converts a rope `rope` to a string and returns it. For the definition of a rope, see the definition of the `util.walk` method.

```

1959 function util.rope_to_string(rope)
1960     local buffer = {}
1961     util.walk(rope, function(x) buffer[#buffer + 1] = x end)
1962     return table.concat(buffer)
1963 end

```

The `util.rope_last` method retrieves the last item in a rope. For the definition of a rope, see the definition of the `util.walk` method.

```

1964 function util.rope_last(rope)
1965     if #rope == 0 then
1966         return nil
1967     else
1968         local l = rope[#rope]
1969         if type(l) == "table" then
1970             return util.rope_last(l)
1971         else
1972             return l
1973         end

```

```

1974 end
1975 end

```

Given an array `ary` and a string `x`, the `util.intersperse` method returns an array `new`, such that `ary[i] == new[2*(i-1)+1]` and `new[2*i] == x` for all  $1 \leq i \leq \#ary$ .

```

1976 function util.intersperse(ary, x)
1977   local new = {}
1978   local l = #ary
1979   for i,v in ipairs(ary) do
1980     local n = #new
1981     new[n + 1] = v
1982     if i ~= l then
1983       new[n + 2] = x
1984     end
1985   end
1986   return new
1987 end

```

Given an array `ary` and a function `f`, the `util.map` method returns an array `new`, such that `new[i] == f(ary[i])` for all  $1 \leq i \leq \#ary$ .

```

1988 function util.map(ary, f)
1989   local new = {}
1990   for i,v in ipairs(ary) do
1991     new[i] = f(v)
1992   end
1993   return new
1994 end

```

Given a table `char_escapes` mapping escapable characters to escaped strings and optionally a table `string_escapes` mapping escapable strings to escaped strings, the `util.escaper` method returns an escaper function that escapes all occurrences of escapable strings and characters (in this order).

The method uses LPeg, which is faster than the Lua `string.gsub` built-in method.

```

1995 function util.escaper(char_escapes, string_escapes)

```

Build a string of escapable characters.

```

1996   local char_escapes_list = ""
1997   for i,_ in pairs(char_escapes) do
1998     char_escapes_list = char_escapes_list .. i
1999   end

```

Create an LPeg capture `escapable` that produces the escaped string corresponding to the matched escapable character.

```

2000   local escapable = S(char_escapes_list) / char_escapes

```

If `string_escapes` is provided, turn `escapable` into the

$$\sum_{(k,v) \in \text{string\_escapes}} P(k) / v + \text{escapable}$$



capture that replaces any occurrence of the string `k` with the string `v` for each  $(k, v) \in \text{string\_escapes}$ . Note that the pattern summation is not commutative and its operands are inspected in the summation order during the matching. As a corollary, the strings always take precedence over the characters.

```
2001  if string_escapes then
2002      for k,v in pairs(string_escapes) do
2003          escapable = P(k) / v + escapable
2004      end
2005  end
```

Create an LPeg capture `escape_string` that captures anything `escapable` does and matches any other unmatched characters.

```
2006  local escape_string = Cs((escapable + any)^0)
```

Return a function that matches the input string `s` against the `escape_string` capture.

```
2007  return function(s)
2008      return lpeg.match(escape_string, s)
2009  end
2010 end
```

The `util.pathname` method produces a pathname out of a directory name `dir` and a filename `file` and returns it.

```
2011 function util.pathname(dir, file)
2012     if #dir == 0 then
2013         return file
2014     else
2015         return dir .. "/" .. file
2016     end
2017 end
```

### 3.1.2 HTML Entities

This section documents the HTML entities recognized by the markdown reader. These functions are encapsulated in the `entities` object. The functions were originally located in the `lunamark/entities.lua` file in the Lunamark Lua module.

```
2018 local entities = {}
2019
2020 local character_entities = {
2021     ["Tab"] = 9,
2022     ["NewLine"] = 10,
2023     ["excl"] = 33,
2024     ["quot"] = 34,
2025     ["QUOT"] = 34,
2026     ["num"] = 35,
2027     ["dollar"] = 36,
2028     ["percent"] = 37,
```

2029 ["amp"] = 38,  
2030 ["AMP"] = 38,  
2031 ["apos"] = 39,  
2032 ["lpar"] = 40,  
2033 ["rpar"] = 41,  
2034 ["ast"] = 42,  
2035 ["midast"] = 42,  
2036 ["plus"] = 43,  
2037 ["comma"] = 44,  
2038 ["period"] = 46,  
2039 ["sol"] = 47,  
2040 ["colon"] = 58,  
2041 ["semi"] = 59,  
2042 ["lt"] = 60,  
2043 ["LT"] = 60,  
2044 ["equals"] = 61,  
2045 ["gt"] = 62,  
2046 ["GT"] = 62,  
2047 ["quest"] = 63,  
2048 ["commat"] = 64,  
2049 ["lsqb"] = 91,  
2050 ["lbrack"] = 91,  
2051 ["bsol"] = 92,  
2052 ["rsqb"] = 93,  
2053 ["rbrack"] = 93,  
2054 ["Hat"] = 94,  
2055 ["lowbar"] = 95,  
2056 ["grave"] = 96,  
2057 ["DiacriticalGrave"] = 96,  
2058 ["lcub"] = 123,  
2059 ["lbrace"] = 123,  
2060 ["verbar"] = 124,  
2061 ["vert"] = 124,  
2062 ["VerticalLine"] = 124,  
2063 ["rcub"] = 125,  
2064 ["rbrace"] = 125,  
2065 ["nbsp"] = 160,  
2066 ["NonBreakingSpace"] = 160,  
2067 ["iexcl"] = 161,  
2068 ["cent"] = 162,  
2069 ["pound"] = 163,  
2070 ["curren"] = 164,  
2071 ["yen"] = 165,  
2072 ["brvbar"] = 166,  
2073 ["sect"] = 167,  
2074 ["Dot"] = 168,  
2075 ["die"] = 168,

2076 ["DoubleDot"] = 168,  
2077 ["uml"] = 168,  
2078 ["copy"] = 169,  
2079 ["COPY"] = 169,  
2080 ["ordf"] = 170,  
2081 ["laquo"] = 171,  
2082 ["not"] = 172,  
2083 ["shy"] = 173,  
2084 ["reg"] = 174,  
2085 ["circledR"] = 174,  
2086 ["REG"] = 174,  
2087 ["macr"] = 175,  
2088 ["OverBar"] = 175,  
2089 ["strns"] = 175,  
2090 ["deg"] = 176,  
2091 ["plusmn"] = 177,  
2092 ["pm"] = 177,  
2093 ["PlusMinus"] = 177,  
2094 ["sup2"] = 178,  
2095 ["sup3"] = 179,  
2096 ["acute"] = 180,  
2097 ["DiacriticalAcute"] = 180,  
2098 ["micro"] = 181,  
2099 ["para"] = 182,  
2100 ["middot"] = 183,  
2101 ["centerdot"] = 183,  
2102 ["CenterDot"] = 183,  
2103 ["cedil"] = 184,  
2104 ["Cedilla"] = 184,  
2105 ["sup1"] = 185,  
2106 ["ordm"] = 186,  
2107 ["raquo"] = 187,  
2108 ["frac14"] = 188,  
2109 ["frac12"] = 189,  
2110 ["half"] = 189,  
2111 ["frac34"] = 190,  
2112 ["iquest"] = 191,  
2113 ["Agrave"] = 192,  
2114 ["Aacute"] = 193,  
2115 ["Acirc"] = 194,  
2116 ["Atilde"] = 195,  
2117 ["Auml"] = 196,  
2118 ["Aring"] = 197,  
2119 ["AElig"] = 198,  
2120 ["Ccedil"] = 199,  
2121 ["Egrave"] = 200,  
2122 ["Eacute"] = 201,

2123 ["Ecirc"] = 202,  
2124 ["Euml"] = 203,  
2125 ["Igrave"] = 204,  
2126 ["Iacute"] = 205,  
2127 ["Icirc"] = 206,  
2128 ["Iuml"] = 207,  
2129 ["ETH"] = 208,  
2130 ["Ntilde"] = 209,  
2131 ["Ograve"] = 210,  
2132 ["Oacute"] = 211,  
2133 ["Ocirc"] = 212,  
2134 ["Otilde"] = 213,  
2135 ["Ouml"] = 214,  
2136 ["times"] = 215,  
2137 ["Oslash"] = 216,  
2138 ["Ugrave"] = 217,  
2139 ["Uacute"] = 218,  
2140 ["Ucirc"] = 219,  
2141 ["Uuml"] = 220,  
2142 ["Yacute"] = 221,  
2143 ["THORN"] = 222,  
2144 ["szlig"] = 223,  
2145 ["agrave"] = 224,  
2146 ["aacute"] = 225,  
2147 ["acirc"] = 226,  
2148 ["atilde"] = 227,  
2149 ["auml"] = 228,  
2150 ["aring"] = 229,  
2151 ["aelig"] = 230,  
2152 ["ccedil"] = 231,  
2153 ["egrave"] = 232,  
2154 ["eacute"] = 233,  
2155 ["ecirc"] = 234,  
2156 ["euml"] = 235,  
2157 ["igrave"] = 236,  
2158 ["iacute"] = 237,  
2159 ["icirc"] = 238,  
2160 ["iuml"] = 239,  
2161 ["eth"] = 240,  
2162 ["ntilde"] = 241,  
2163 ["ograve"] = 242,  
2164 ["oacute"] = 243,  
2165 ["ocirc"] = 244,  
2166 ["otilde"] = 245,  
2167 ["ouml"] = 246,  
2168 ["divide"] = 247,  
2169 ["div"] = 247,

2170 ["oslash"] = 248,  
2171 ["ugrave"] = 249,  
2172 ["uacute"] = 250,  
2173 ["ucirc"] = 251,  
2174 ["uuml"] = 252,  
2175 ["yacute"] = 253,  
2176 ["thorn"] = 254,  
2177 ["yuml"] = 255,  
2178 ["Amacr"] = 256,  
2179 ["amacr"] = 257,  
2180 ["Abreve"] = 258,  
2181 ["abreve"] = 259,  
2182 ["Aogon"] = 260,  
2183 ["aogon"] = 261,  
2184 ["Cacute"] = 262,  
2185 ["cacute"] = 263,  
2186 ["Ccirc"] = 264,  
2187 ["ccirc"] = 265,  
2188 ["Cdot"] = 266,  
2189 ["cdot"] = 267,  
2190 ["Ccaron"] = 268,  
2191 ["ccaron"] = 269,  
2192 ["Dcaron"] = 270,  
2193 ["dcaron"] = 271,  
2194 ["Dstrok"] = 272,  
2195 ["dstrok"] = 273,  
2196 ["Emacr"] = 274,  
2197 ["emacr"] = 275,  
2198 ["Edot"] = 278,  
2199 ["edot"] = 279,  
2200 ["Eogon"] = 280,  
2201 ["eogon"] = 281,  
2202 ["Ecaron"] = 282,  
2203 ["ecaron"] = 283,  
2204 ["Gcirc"] = 284,  
2205 ["gcirc"] = 285,  
2206 ["Gbreve"] = 286,  
2207 ["gbreve"] = 287,  
2208 ["Gdot"] = 288,  
2209 ["gdot"] = 289,  
2210 ["Gcedil"] = 290,  
2211 ["Hcirc"] = 292,  
2212 ["hcirc"] = 293,  
2213 ["Hstrok"] = 294,  
2214 ["hstrok"] = 295,  
2215 ["Itilde"] = 296,  
2216 ["itilde"] = 297,

2217 ["Imacr"] = 298,  
2218 ["imacr"] = 299,  
2219 ["Iogon"] = 302,  
2220 ["iogon"] = 303,  
2221 ["Idot"] = 304,  
2222 ["imath"] = 305,  
2223 ["inodot"] = 305,  
2224 ["IJlig"] = 306,  
2225 ["ijlig"] = 307,  
2226 ["Jcirc"] = 308,  
2227 ["jcirc"] = 309,  
2228 ["Kcedil"] = 310,  
2229 ["kcedil"] = 311,  
2230 ["kgreen"] = 312,  
2231 ["Lacute"] = 313,  
2232 ["lacute"] = 314,  
2233 ["Lcedil"] = 315,  
2234 ["lcedil"] = 316,  
2235 ["Lcaron"] = 317,  
2236 ["lcaron"] = 318,  
2237 ["Lmidot"] = 319,  
2238 ["lmidot"] = 320,  
2239 ["Lstrok"] = 321,  
2240 ["lstrok"] = 322,  
2241 ["Nacute"] = 323,  
2242 ["nacute"] = 324,  
2243 ["Ncedil"] = 325,  
2244 ["ncedil"] = 326,  
2245 ["Ncaron"] = 327,  
2246 ["ncaron"] = 328,  
2247 ["napos"] = 329,  
2248 ["ENG"] = 330,  
2249 ["eng"] = 331,  
2250 ["Omacr"] = 332,  
2251 ["omacr"] = 333,  
2252 ["Odblac"] = 336,  
2253 ["odblac"] = 337,  
2254 ["OElig"] = 338,  
2255 ["oelig"] = 339,  
2256 ["Racute"] = 340,  
2257 ["racute"] = 341,  
2258 ["Rcedil"] = 342,  
2259 ["rcedil"] = 343,  
2260 ["Rcaron"] = 344,  
2261 ["rcaron"] = 345,  
2262 ["Sacute"] = 346,  
2263 ["sacute"] = 347,

2264 ["Scirc"] = 348,  
2265 ["scirc"] = 349,  
2266 ["Scedil"] = 350,  
2267 ["scedil"] = 351,  
2268 ["Scaron"] = 352,  
2269 ["scaron"] = 353,  
2270 ["Tcedil"] = 354,  
2271 ["tcedil"] = 355,  
2272 ["Tcaron"] = 356,  
2273 ["tcaron"] = 357,  
2274 ["Tstrok"] = 358,  
2275 ["tstrok"] = 359,  
2276 ["Utilde"] = 360,  
2277 ["utilde"] = 361,  
2278 ["Umacr"] = 362,  
2279 ["umacr"] = 363,  
2280 ["Ubreve"] = 364,  
2281 ["ubreve"] = 365,  
2282 ["Uring"] = 366,  
2283 ["uring"] = 367,  
2284 ["Udblac"] = 368,  
2285 ["udblac"] = 369,  
2286 ["Uogon"] = 370,  
2287 ["uogon"] = 371,  
2288 ["Wcirc"] = 372,  
2289 ["wcirc"] = 373,  
2290 ["Ycirc"] = 374,  
2291 ["ycirc"] = 375,  
2292 ["Yuml"] = 376,  
2293 ["Zacute"] = 377,  
2294 ["zacute"] = 378,  
2295 ["Zdot"] = 379,  
2296 ["zdot"] = 380,  
2297 ["Zcaron"] = 381,  
2298 ["zcaron"] = 382,  
2299 ["fnof"] = 402,  
2300 ["imped"] = 437,  
2301 ["gacute"] = 501,  
2302 ["jmath"] = 567,  
2303 ["circ"] = 710,  
2304 ["caron"] = 711,  
2305 ["Hacek"] = 711,  
2306 ["breve"] = 728,  
2307 ["Breve"] = 728,  
2308 ["dot"] = 729,  
2309 ["DiacriticalDot"] = 729,  
2310 ["ring"] = 730,

2311 ["ogon"] = 731,  
2312 ["tilde"] = 732,  
2313 ["DiacriticalTilde"] = 732,  
2314 ["dblac"] = 733,  
2315 ["DiacriticalDoubleAcute"] = 733,  
2316 ["DownBreve"] = 785,  
2317 ["UnderBar"] = 818,  
2318 ["Alpha"] = 913,  
2319 ["Beta"] = 914,  
2320 ["Gamma"] = 915,  
2321 ["Delta"] = 916,  
2322 ["Epsilon"] = 917,  
2323 ["Zeta"] = 918,  
2324 ["Eta"] = 919,  
2325 ["Theta"] = 920,  
2326 ["Iota"] = 921,  
2327 ["Kappa"] = 922,  
2328 ["Lambda"] = 923,  
2329 ["Mu"] = 924,  
2330 ["Nu"] = 925,  
2331 ["Xi"] = 926,  
2332 ["Omicron"] = 927,  
2333 ["Pi"] = 928,  
2334 ["Rho"] = 929,  
2335 ["Sigma"] = 931,  
2336 ["Tau"] = 932,  
2337 ["Upsilon"] = 933,  
2338 ["Phi"] = 934,  
2339 ["Chi"] = 935,  
2340 ["Psi"] = 936,  
2341 ["Omega"] = 937,  
2342 ["alpha"] = 945,  
2343 ["beta"] = 946,  
2344 ["gamma"] = 947,  
2345 ["delta"] = 948,  
2346 ["epsiv"] = 949,  
2347 ["varepsilon"] = 949,  
2348 ["epsilon"] = 949,  
2349 ["zeta"] = 950,  
2350 ["eta"] = 951,  
2351 ["theta"] = 952,  
2352 ["iota"] = 953,  
2353 ["kappa"] = 954,  
2354 ["lambda"] = 955,  
2355 ["mu"] = 956,  
2356 ["nu"] = 957,  
2357 ["xi"] = 958,



2358 ["omicron"] = 959,  
2359 ["pi"] = 960,  
2360 ["rho"] = 961,  
2361 ["sigmav"] = 962,  
2362 ["varsigma"] = 962,  
2363 ["sigmaf"] = 962,  
2364 ["sigma"] = 963,  
2365 ["tau"] = 964,  
2366 ["upsilon"] = 965,  
2367 ["upsilon"] = 965,  
2368 ["phi"] = 966,  
2369 ["phiv"] = 966,  
2370 ["varphi"] = 966,  
2371 ["chi"] = 967,  
2372 ["psi"] = 968,  
2373 ["omega"] = 969,  
2374 ["thetav"] = 977,  
2375 ["vartheta"] = 977,  
2376 ["thetasym"] = 977,  
2377 ["Upsilon"] = 978,  
2378 ["upsih"] = 978,  
2379 ["straightphi"] = 981,  
2380 ["piv"] = 982,  
2381 ["varpi"] = 982,  
2382 ["Gammad"] = 988,  
2383 ["gammad"] = 989,  
2384 ["digamma"] = 989,  
2385 ["kappav"] = 1008,  
2386 ["varkappa"] = 1008,  
2387 ["rhov"] = 1009,  
2388 ["varrho"] = 1009,  
2389 ["epsi"] = 1013,  
2390 ["straightepsilon"] = 1013,  
2391 ["bepsi"] = 1014,  
2392 ["backepsilon"] = 1014,  
2393 ["IOcy"] = 1025,  
2394 ["DJcy"] = 1026,  
2395 ["GJcy"] = 1027,  
2396 ["Jukcy"] = 1028,  
2397 ["DScy"] = 1029,  
2398 ["Iukcy"] = 1030,  
2399 ["YIcy"] = 1031,  
2400 ["Jsercy"] = 1032,  
2401 ["LJcy"] = 1033,  
2402 ["NJcy"] = 1034,  
2403 ["TSHcy"] = 1035,  
2404 ["KJcy"] = 1036,

2405 ["Ubrcy"] = 1038,  
2406 ["DZcy"] = 1039,  
2407 ["Acy"] = 1040,  
2408 ["Bcy"] = 1041,  
2409 ["Vcy"] = 1042,  
2410 ["Gcy"] = 1043,  
2411 ["Dcy"] = 1044,  
2412 ["IEcy"] = 1045,  
2413 ["ZHcy"] = 1046,  
2414 ["Zcy"] = 1047,  
2415 ["Icy"] = 1048,  
2416 ["Jcy"] = 1049,  
2417 ["Kcy"] = 1050,  
2418 ["Lcy"] = 1051,  
2419 ["Mcy"] = 1052,  
2420 ["Ncy"] = 1053,  
2421 ["Ocy"] = 1054,  
2422 ["Pcy"] = 1055,  
2423 ["Rcy"] = 1056,  
2424 ["Scy"] = 1057,  
2425 ["Tcy"] = 1058,  
2426 ["Ucy"] = 1059,  
2427 ["Fcy"] = 1060,  
2428 ["KHcy"] = 1061,  
2429 ["TScy"] = 1062,  
2430 ["CHcy"] = 1063,  
2431 ["SHcy"] = 1064,  
2432 ["SHCHcy"] = 1065,  
2433 ["HARDcy"] = 1066,  
2434 ["Ycy"] = 1067,  
2435 ["SOFTcy"] = 1068,  
2436 ["Ecy"] = 1069,  
2437 ["YUcy"] = 1070,  
2438 ["YAcy"] = 1071,  
2439 ["acy"] = 1072,  
2440 ["bcy"] = 1073,  
2441 ["vcy"] = 1074,  
2442 ["gcy"] = 1075,  
2443 ["dcy"] = 1076,  
2444 ["iecy"] = 1077,  
2445 ["zhcy"] = 1078,  
2446 ["zcy"] = 1079,  
2447 ["icy"] = 1080,  
2448 ["jcy"] = 1081,  
2449 ["kcy"] = 1082,  
2450 ["lcy"] = 1083,  
2451 ["mcy"] = 1084,

2452 ["ncy"] = 1085,  
2453 ["ocy"] = 1086,  
2454 ["pcy"] = 1087,  
2455 ["rcy"] = 1088,  
2456 ["scy"] = 1089,  
2457 ["tcy"] = 1090,  
2458 ["ucy"] = 1091,  
2459 ["fcy"] = 1092,  
2460 ["khcy"] = 1093,  
2461 ["tscy"] = 1094,  
2462 ["chcy"] = 1095,  
2463 ["shcy"] = 1096,  
2464 ["shchcy"] = 1097,  
2465 ["hardcy"] = 1098,  
2466 ["ycy"] = 1099,  
2467 ["softcy"] = 1100,  
2468 ["ecy"] = 1101,  
2469 ["yucy"] = 1102,  
2470 ["yacy"] = 1103,  
2471 ["iocy"] = 1105,  
2472 ["djcy"] = 1106,  
2473 ["gjcy"] = 1107,  
2474 ["jukcy"] = 1108,  
2475 ["dscy"] = 1109,  
2476 ["iukcy"] = 1110,  
2477 ["yicy"] = 1111,  
2478 ["jsercy"] = 1112,  
2479 ["ljcy"] = 1113,  
2480 ["njcy"] = 1114,  
2481 ["tshcy"] = 1115,  
2482 ["kjcy"] = 1116,  
2483 ["ubrscy"] = 1118,  
2484 ["dzcy"] = 1119,  
2485 ["ensp"] = 8194,  
2486 ["emsp"] = 8195,  
2487 ["emsp13"] = 8196,  
2488 ["emsp14"] = 8197,  
2489 ["numsp"] = 8199,  
2490 ["puncsp"] = 8200,  
2491 ["thinsp"] = 8201,  
2492 ["ThinSpace"] = 8201,  
2493 ["hairsp"] = 8202,  
2494 ["VeryThinSpace"] = 8202,  
2495 ["ZeroWidthSpace"] = 8203,  
2496 ["NegativeVeryThinSpace"] = 8203,  
2497 ["NegativeThinSpace"] = 8203,  
2498 ["NegativeMediumSpace"] = 8203,

2499 ["NegativeThickSpace"] = 8203,  
2500 ["zwnj"] = 8204,  
2501 ["zwj"] = 8205,  
2502 ["lrm"] = 8206,  
2503 ["rlm"] = 8207,  
2504 ["hyphen"] = 8208,  
2505 ["dash"] = 8208,  
2506 ["ndash"] = 8211,  
2507 ["mdash"] = 8212,  
2508 ["horbar"] = 8213,  
2509 ["Verbar"] = 8214,  
2510 ["Vert"] = 8214,  
2511 ["lsquo"] = 8216,  
2512 ["OpenCurlyQuote"] = 8216,  
2513 ["rsquo"] = 8217,  
2514 ["rsquor"] = 8217,  
2515 ["CloseCurlyQuote"] = 8217,  
2516 ["lsquor"] = 8218,  
2517 ["sbquo"] = 8218,  
2518 ["ldquo"] = 8220,  
2519 ["OpenCurlyDoubleQuote"] = 8220,  
2520 ["rdquo"] = 8221,  
2521 ["rdquor"] = 8221,  
2522 ["CloseCurlyDoubleQuote"] = 8221,  
2523 ["ldquor"] = 8222,  
2524 ["bdquo"] = 8222,  
2525 ["dagger"] = 8224,  
2526 ["Dagger"] = 8225,  
2527 ["ddagger"] = 8225,  
2528 ["bull"] = 8226,  
2529 ["bullet"] = 8226,  
2530 ["nldr"] = 8229,  
2531 ["hellip"] = 8230,  
2532 ["mldr"] = 8230,  
2533 ["permil"] = 8240,  
2534 ["pertenk"] = 8241,  
2535 ["prime"] = 8242,  
2536 ["Prime"] = 8243,  
2537 ["tprime"] = 8244,  
2538 ["bprime"] = 8245,  
2539 ["backprime"] = 8245,  
2540 ["lsaquo"] = 8249,  
2541 ["rsaquo"] = 8250,  
2542 ["oline"] = 8254,  
2543 ["caret"] = 8257,  
2544 ["hybull"] = 8259,  
2545 ["frasl"] = 8260,

2546 ["bsemi"] = 8271,  
2547 ["qprime"] = 8279,  
2548 ["MediumSpace"] = 8287,  
2549 ["NoBreak"] = 8288,  
2550 ["ApplyFunction"] = 8289,  
2551 ["af"] = 8289,  
2552 ["InvisibleTimes"] = 8290,  
2553 ["it"] = 8290,  
2554 ["InvisibleComma"] = 8291,  
2555 ["ic"] = 8291,  
2556 ["euro"] = 8364,  
2557 ["tdot"] = 8411,  
2558 ["TripleDot"] = 8411,  
2559 ["DotDot"] = 8412,  
2560 ["Copf"] = 8450,  
2561 ["complexes"] = 8450,  
2562 ["incare"] = 8453,  
2563 ["gscr"] = 8458,  
2564 ["hamilt"] = 8459,  
2565 ["HilbertSpace"] = 8459,  
2566 ["Hscr"] = 8459,  
2567 ["Hfr"] = 8460,  
2568 ["Poincareplane"] = 8460,  
2569 ["quaternions"] = 8461,  
2570 ["Hopf"] = 8461,  
2571 ["planckh"] = 8462,  
2572 ["planck"] = 8463,  
2573 ["hbar"] = 8463,  
2574 ["plankv"] = 8463,  
2575 ["hslash"] = 8463,  
2576 ["Iscr"] = 8464,  
2577 ["imagline"] = 8464,  
2578 ["image"] = 8465,  
2579 ["Im"] = 8465,  
2580 ["imagpart"] = 8465,  
2581 ["Ifr"] = 8465,  
2582 ["Lscr"] = 8466,  
2583 ["lagran"] = 8466,  
2584 ["Laplacetrif"] = 8466,  
2585 ["ell"] = 8467,  
2586 ["Nopf"] = 8469,  
2587 ["naturals"] = 8469,  
2588 ["numero"] = 8470,  
2589 ["copysr"] = 8471,  
2590 ["weierp"] = 8472,  
2591 ["wp"] = 8472,  
2592 ["Popf"] = 8473,

2593 ["primes"] = 8473,  
2594 ["rationals"] = 8474,  
2595 ["Qopf"] = 8474,  
2596 ["Rscr"] = 8475,  
2597 ["realine"] = 8475,  
2598 ["real"] = 8476,  
2599 ["Re"] = 8476,  
2600 ["realpart"] = 8476,  
2601 ["Rfr"] = 8476,  
2602 ["reals"] = 8477,  
2603 ["Ropf"] = 8477,  
2604 ["rx"] = 8478,  
2605 ["trade"] = 8482,  
2606 ["TRADE"] = 8482,  
2607 ["integers"] = 8484,  
2608 ["Zopf"] = 8484,  
2609 ["ohm"] = 8486,  
2610 ["mho"] = 8487,  
2611 ["Zfr"] = 8488,  
2612 ["zeetrf"] = 8488,  
2613 ["iiota"] = 8489,  
2614 ["angst"] = 8491,  
2615 ["bernou"] = 8492,  
2616 ["Bernoullis"] = 8492,  
2617 ["Bscr"] = 8492,  
2618 ["Cfr"] = 8493,  
2619 ["Cayleys"] = 8493,  
2620 ["escr"] = 8495,  
2621 ["Escr"] = 8496,  
2622 ["expectation"] = 8496,  
2623 ["Fscr"] = 8497,  
2624 ["Fouriertrf"] = 8497,  
2625 ["phmmat"] = 8499,  
2626 ["Mellintrf"] = 8499,  
2627 ["Mscr"] = 8499,  
2628 ["order"] = 8500,  
2629 ["orderof"] = 8500,  
2630 ["oscr"] = 8500,  
2631 ["alefsym"] = 8501,  
2632 ["aleph"] = 8501,  
2633 ["beth"] = 8502,  
2634 ["gimel"] = 8503,  
2635 ["daleth"] = 8504,  
2636 ["CapitalDifferentialD"] = 8517,  
2637 ["DD"] = 8517,  
2638 ["DifferentialD"] = 8518,  
2639 ["dd"] = 8518,

2640 ["ExponentialE"] = 8519,  
 2641 ["exponentiale"] = 8519,  
 2642 ["ee"] = 8519,  
 2643 ["ImaginaryI"] = 8520,  
 2644 ["ii"] = 8520,  
 2645 ["frac13"] = 8531,  
 2646 ["frac23"] = 8532,  
 2647 ["frac15"] = 8533,  
 2648 ["frac25"] = 8534,  
 2649 ["frac35"] = 8535,  
 2650 ["frac45"] = 8536,  
 2651 ["frac16"] = 8537,  
 2652 ["frac56"] = 8538,  
 2653 ["frac18"] = 8539,  
 2654 ["frac38"] = 8540,  
 2655 ["frac58"] = 8541,  
 2656 ["frac78"] = 8542,  
 2657 ["larr"] = 8592,  
 2658 ["leftarrow"] = 8592,  
 2659 ["LeftArrow"] = 8592,  
 2660 ["slarr"] = 8592,  
 2661 ["ShortLeftArrow"] = 8592,  
 2662 ["uarr"] = 8593,  
 2663 ["uparrow"] = 8593,  
 2664 ["UpArrow"] = 8593,  
 2665 ["ShortUpArrow"] = 8593,  
 2666 ["rarr"] = 8594,  
 2667 ["rightarrow"] = 8594,  
 2668 ["RightArrow"] = 8594,  
 2669 ["srarr"] = 8594,  
 2670 ["ShortRightArrow"] = 8594,  
 2671 ["darr"] = 8595,  
 2672 ["downarrow"] = 8595,  
 2673 ["DownArrow"] = 8595,  
 2674 ["ShortDownArrow"] = 8595,  
 2675 ["harr"] = 8596,  
 2676 ["leftrightarrow"] = 8596,  
 2677 ["LeftRightArrow"] = 8596,  
 2678 ["varr"] = 8597,  
 2679 ["updownarrow"] = 8597,  
 2680 ["UpDownArrow"] = 8597,  
 2681 ["nwarr"] = 8598,  
 2682 ["UpperLeftArrow"] = 8598,  
 2683 ["nwarrow"] = 8598,  
 2684 ["nearr"] = 8599,  
 2685 ["UpperRightArrow"] = 8599,  
 2686 ["nearrow"] = 8599,

2687 ["searr"] = 8600,  
 2688 ["searrow"] = 8600,  
 2689 ["LowerRightArrow"] = 8600,  
 2690 ["swarr"] = 8601,  
 2691 ["swarrow"] = 8601,  
 2692 ["LowerLeftArrow"] = 8601,  
 2693 ["nlarr"] = 8602,  
 2694 ["nleftarrow"] = 8602,  
 2695 ["nrarr"] = 8603,  
 2696 ["nrightarrow"] = 8603,  
 2697 ["rarrw"] = 8605,  
 2698 ["rightsquigarrow"] = 8605,  
 2699 ["Larr"] = 8606,  
 2700 ["twoheadleftarrow"] = 8606,  
 2701 ["Uarr"] = 8607,  
 2702 ["Rarr"] = 8608,  
 2703 ["twoheadrightarrow"] = 8608,  
 2704 ["Darr"] = 8609,  
 2705 ["larrtl"] = 8610,  
 2706 ["leftarrowtail"] = 8610,  
 2707 ["rarrtl"] = 8611,  
 2708 ["rightarrowtail"] = 8611,  
 2709 ["LeftTeeArrow"] = 8612,  
 2710 ["mapstoleft"] = 8612,  
 2711 ["UpTeeArrow"] = 8613,  
 2712 ["mapstoup"] = 8613,  
 2713 ["map"] = 8614,  
 2714 ["RightTeeArrow"] = 8614,  
 2715 ["mapsto"] = 8614,  
 2716 ["DownTeeArrow"] = 8615,  
 2717 ["mapstodown"] = 8615,  
 2718 ["larrhk"] = 8617,  
 2719 ["hookleftarrow"] = 8617,  
 2720 ["rarrhk"] = 8618,  
 2721 ["hookrightarrow"] = 8618,  
 2722 ["larrlp"] = 8619,  
 2723 ["looparrowleft"] = 8619,  
 2724 ["rarrlp"] = 8620,  
 2725 ["looparrowright"] = 8620,  
 2726 ["harrw"] = 8621,  
 2727 ["leftrightsquigarrow"] = 8621,  
 2728 ["nharr"] = 8622,  
 2729 ["nleftrightarrow"] = 8622,  
 2730 ["lsh"] = 8624,  
 2731 ["Lsh"] = 8624,  
 2732 ["rsh"] = 8625,  
 2733 ["Rsh"] = 8625,



2734 ["ldsh"] = 8626,  
 2735 ["rdsh"] = 8627,  
 2736 ["crarr"] = 8629,  
 2737 ["cularr"] = 8630,  
 2738 ["curvearrowleft"] = 8630,  
 2739 ["curarr"] = 8631,  
 2740 ["curvearrowright"] = 8631,  
 2741 ["olarr"] = 8634,  
 2742 ["circlearrowleft"] = 8634,  
 2743 ["orarr"] = 8635,  
 2744 ["circlearrowright"] = 8635,  
 2745 ["lharu"] = 8636,  
 2746 ["LeftVector"] = 8636,  
 2747 ["leftharpoonup"] = 8636,  
 2748 ["lhard"] = 8637,  
 2749 ["leftharpoondown"] = 8637,  
 2750 ["DownLeftVector"] = 8637,  
 2751 ["uharr"] = 8638,  
 2752 ["upharpoonright"] = 8638,  
 2753 ["RightUpVector"] = 8638,  
 2754 ["uharl"] = 8639,  
 2755 ["upharpoonleft"] = 8639,  
 2756 ["LeftUpVector"] = 8639,  
 2757 ["rharu"] = 8640,  
 2758 ["RightVector"] = 8640,  
 2759 ["rightharpoonup"] = 8640,  
 2760 ["rhard"] = 8641,  
 2761 ["rightharpoondown"] = 8641,  
 2762 ["DownRightVector"] = 8641,  
 2763 ["dharr"] = 8642,  
 2764 ["RightDownVector"] = 8642,  
 2765 ["downharpoonright"] = 8642,  
 2766 ["dharl"] = 8643,  
 2767 ["LeftDownVector"] = 8643,  
 2768 ["downharpoonleft"] = 8643,  
 2769 ["rlarr"] = 8644,  
 2770 ["rightleftarrows"] = 8644,  
 2771 ["RightArrowLeftArrow"] = 8644,  
 2772 ["udarr"] = 8645,  
 2773 ["UpArrowDownArrow"] = 8645,  
 2774 ["lrarr"] = 8646,  
 2775 ["leftrightarrows"] = 8646,  
 2776 ["LeftArrowRightArrow"] = 8646,  
 2777 ["llarr"] = 8647,  
 2778 ["leftleftarrows"] = 8647,  
 2779 ["uuarr"] = 8648,  
 2780 ["upuparrows"] = 8648,

2781 ["rrarr"] = 8649,  
 2782 ["rightrightarrows"] = 8649,  
 2783 ["ddarr"] = 8650,  
 2784 ["downdownarrows"] = 8650,  
 2785 ["lrhar"] = 8651,  
 2786 ["ReverseEquilibrium"] = 8651,  
 2787 ["leftrightharpoons"] = 8651,  
 2788 ["rlhar"] = 8652,  
 2789 ["rightleftharpoons"] = 8652,  
 2790 ["Equilibrium"] = 8652,  
 2791 ["nlArr"] = 8653,  
 2792 ["nLeftarrow"] = 8653,  
 2793 ["nhArr"] = 8654,  
 2794 ["nLeftrightarrow"] = 8654,  
 2795 ["nrArr"] = 8655,  
 2796 ["nrightarrow"] = 8655,  
 2797 ["lArr"] = 8656,  
 2798 ["Leftarrow"] = 8656,  
 2799 ["DoubleLeftArrow"] = 8656,  
 2800 ["uArr"] = 8657,  
 2801 ["Uparrow"] = 8657,  
 2802 ["DoubleUpArrow"] = 8657,  
 2803 ["rArr"] = 8658,  
 2804 ["Rightarrow"] = 8658,  
 2805 ["Implies"] = 8658,  
 2806 ["DoubleRightArrow"] = 8658,  
 2807 ["dArr"] = 8659,  
 2808 ["Downarrow"] = 8659,  
 2809 ["DoubleDownArrow"] = 8659,  
 2810 ["hArr"] = 8660,  
 2811 ["Leftrightarrow"] = 8660,  
 2812 ["DoubleLeftRightArrow"] = 8660,  
 2813 ["iff"] = 8660,  
 2814 ["vArr"] = 8661,  
 2815 ["Updownarrow"] = 8661,  
 2816 ["DoubleUpDownArrow"] = 8661,  
 2817 ["nwArr"] = 8662,  
 2818 ["neArr"] = 8663,  
 2819 ["seArr"] = 8664,  
 2820 ["swArr"] = 8665,  
 2821 ["lAarr"] = 8666,  
 2822 ["Lleftarrow"] = 8666,  
 2823 ["rAarr"] = 8667,  
 2824 ["Rrightarrow"] = 8667,  
 2825 ["zigrarr"] = 8669,  
 2826 ["larrb"] = 8676,  
 2827 ["LeftArrowBar"] = 8676,

2828 ["rarrb"] = 8677,  
 2829 ["RightArrowBar"] = 8677,  
 2830 ["duarr"] = 8693,  
 2831 ["DownArrowUpArrow"] = 8693,  
 2832 ["loarr"] = 8701,  
 2833 ["roarr"] = 8702,  
 2834 ["hoarr"] = 8703,  
 2835 ["forall"] = 8704,  
 2836 ["ForAll"] = 8704,  
 2837 ["comp"] = 8705,  
 2838 ["complement"] = 8705,  
 2839 ["part"] = 8706,  
 2840 ["PartialD"] = 8706,  
 2841 ["exist"] = 8707,  
 2842 ["Exists"] = 8707,  
 2843 ["nexist"] = 8708,  
 2844 ["NotExists"] = 8708,  
 2845 ["nexists"] = 8708,  
 2846 ["empty"] = 8709,  
 2847 ["emptyset"] = 8709,  
 2848 ["emptyv"] = 8709,  
 2849 ["varnothing"] = 8709,  
 2850 ["nabla"] = 8711,  
 2851 ["Del"] = 8711,  
 2852 ["isin"] = 8712,  
 2853 ["isinv"] = 8712,  
 2854 ["Element"] = 8712,  
 2855 ["in"] = 8712,  
 2856 ["notin"] = 8713,  
 2857 ["NotElement"] = 8713,  
 2858 ["notinva"] = 8713,  
 2859 ["niv"] = 8715,  
 2860 ["ReverseElement"] = 8715,  
 2861 ["ni"] = 8715,  
 2862 ["SuchThat"] = 8715,  
 2863 ["notni"] = 8716,  
 2864 ["notniva"] = 8716,  
 2865 ["NotReverseElement"] = 8716,  
 2866 ["prod"] = 8719,  
 2867 ["Product"] = 8719,  
 2868 ["coprod"] = 8720,  
 2869 ["Coproduct"] = 8720,  
 2870 ["sum"] = 8721,  
 2871 ["Sum"] = 8721,  
 2872 ["minus"] = 8722,  
 2873 ["mnplus"] = 8723,  
 2874 ["mp"] = 8723,

2875 ["MinusPlus"] = 8723,  
 2876 ["plusdo"] = 8724,  
 2877 ["dotplus"] = 8724,  
 2878 ["setmn"] = 8726,  
 2879 ["setminus"] = 8726,  
 2880 ["Backslash"] = 8726,  
 2881 ["ssetmn"] = 8726,  
 2882 ["smallsetminus"] = 8726,  
 2883 ["lowast"] = 8727,  
 2884 ["compfn"] = 8728,  
 2885 ["SmallCircle"] = 8728,  
 2886 ["radic"] = 8730,  
 2887 ["Sqrt"] = 8730,  
 2888 ["prop"] = 8733,  
 2889 ["propto"] = 8733,  
 2890 ["Proportional"] = 8733,  
 2891 ["vprop"] = 8733,  
 2892 ["varpropto"] = 8733,  
 2893 ["infin"] = 8734,  
 2894 ["angrt"] = 8735,  
 2895 ["ang"] = 8736,  
 2896 ["angle"] = 8736,  
 2897 ["angmsd"] = 8737,  
 2898 ["measuredangle"] = 8737,  
 2899 ["angsph"] = 8738,  
 2900 ["mid"] = 8739,  
 2901 ["VerticalBar"] = 8739,  
 2902 ["smid"] = 8739,  
 2903 ["shortmid"] = 8739,  
 2904 ["nmid"] = 8740,  
 2905 ["NotVerticalBar"] = 8740,  
 2906 ["nsmid"] = 8740,  
 2907 ["nshortmid"] = 8740,  
 2908 ["par"] = 8741,  
 2909 ["parallel"] = 8741,  
 2910 ["DoubleVerticalBar"] = 8741,  
 2911 ["spar"] = 8741,  
 2912 ["shortparallel"] = 8741,  
 2913 ["npar"] = 8742,  
 2914 ["nparallel"] = 8742,  
 2915 ["NotDoubleVerticalBar"] = 8742,  
 2916 ["nspar"] = 8742,  
 2917 ["nshortparallel"] = 8742,  
 2918 ["and"] = 8743,  
 2919 ["wedge"] = 8743,  
 2920 ["or"] = 8744,  
 2921 ["vee"] = 8744,

2922 ["cap"] = 8745,  
 2923 ["cup"] = 8746,  
 2924 ["int"] = 8747,  
 2925 ["Integral"] = 8747,  
 2926 ["Int"] = 8748,  
 2927 ["tint"] = 8749,  
 2928 ["iiint"] = 8749,  
 2929 ["conint"] = 8750,  
 2930 ["oint"] = 8750,  
 2931 ["ContourIntegral"] = 8750,  
 2932 ["Conint"] = 8751,  
 2933 ["DoubleContourIntegral"] = 8751,  
 2934 ["Cconint"] = 8752,  
 2935 ["cwint"] = 8753,  
 2936 ["cwconint"] = 8754,  
 2937 ["ClockwiseContourIntegral"] = 8754,  
 2938 ["awconint"] = 8755,  
 2939 ["CounterClockwiseContourIntegral"] = 8755,  
 2940 ["there4"] = 8756,  
 2941 ["therefore"] = 8756,  
 2942 ["Therefore"] = 8756,  
 2943 ["because"] = 8757,  
 2944 ["because"] = 8757,  
 2945 ["Because"] = 8757,  
 2946 ["ratio"] = 8758,  
 2947 ["Colon"] = 8759,  
 2948 ["Proportion"] = 8759,  
 2949 ["minusd"] = 8760,  
 2950 ["dotminus"] = 8760,  
 2951 ["mDDot"] = 8762,  
 2952 ["homtht"] = 8763,  
 2953 ["sim"] = 8764,  
 2954 ["Tilde"] = 8764,  
 2955 ["thksim"] = 8764,  
 2956 ["thicksim"] = 8764,  
 2957 ["bsim"] = 8765,  
 2958 ["backsim"] = 8765,  
 2959 ["ac"] = 8766,  
 2960 ["mstpos"] = 8766,  
 2961 ["acd"] = 8767,  
 2962 ["wreath"] = 8768,  
 2963 ["VerticalTilde"] = 8768,  
 2964 ["wr"] = 8768,  
 2965 ["nsim"] = 8769,  
 2966 ["NotTilde"] = 8769,  
 2967 ["esim"] = 8770,  
 2968 ["EqualTilde"] = 8770,

2969 ["eqsim"] = 8770,  
 2970 ["sime"] = 8771,  
 2971 ["TildeEqual"] = 8771,  
 2972 ["simeq"] = 8771,  
 2973 ["nsime"] = 8772,  
 2974 ["nsimeq"] = 8772,  
 2975 ["NotTildeEqual"] = 8772,  
 2976 ["cong"] = 8773,  
 2977 ["TildeFullEqual"] = 8773,  
 2978 ["simne"] = 8774,  
 2979 ["ncong"] = 8775,  
 2980 ["NotTildeFullEqual"] = 8775,  
 2981 ["asymp"] = 8776,  
 2982 ["ap"] = 8776,  
 2983 ["TildeTilde"] = 8776,  
 2984 ["approx"] = 8776,  
 2985 ["thkap"] = 8776,  
 2986 ["thickapprox"] = 8776,  
 2987 ["nap"] = 8777,  
 2988 ["NotTildeTilde"] = 8777,  
 2989 ["naprox"] = 8777,  
 2990 ["ape"] = 8778,  
 2991 ["approxpeq"] = 8778,  
 2992 ["apid"] = 8779,  
 2993 ["bcong"] = 8780,  
 2994 ["backcong"] = 8780,  
 2995 ["asympeq"] = 8781,  
 2996 ["CupCap"] = 8781,  
 2997 ["bump"] = 8782,  
 2998 ["HumpDownHump"] = 8782,  
 2999 ["Bumpeq"] = 8782,  
 3000 ["bumpe"] = 8783,  
 3001 ["HumpEqual"] = 8783,  
 3002 ["bumpeq"] = 8783,  
 3003 ["esdot"] = 8784,  
 3004 ["DotEqual"] = 8784,  
 3005 ["doteq"] = 8784,  
 3006 ["eDot"] = 8785,  
 3007 ["doteqdot"] = 8785,  
 3008 ["efDot"] = 8786,  
 3009 ["fallingdotseq"] = 8786,  
 3010 ["erDot"] = 8787,  
 3011 ["risingdotseq"] = 8787,  
 3012 ["colone"] = 8788,  
 3013 ["coloneq"] = 8788,  
 3014 ["Assign"] = 8788,  
 3015 ["ecolon"] = 8789,

3016 ["eqcolon"] = 8789,  
3017 ["ecir"] = 8790,  
3018 ["eqcirc"] = 8790,  
3019 ["cire"] = 8791,  
3020 ["circeq"] = 8791,  
3021 ["wedgeq"] = 8793,  
3022 ["veeeq"] = 8794,  
3023 ["trie"] = 8796,  
3024 ["triangleq"] = 8796,  
3025 ["equest"] = 8799,  
3026 ["questeq"] = 8799,  
3027 ["ne"] = 8800,  
3028 ["NotEqual"] = 8800,  
3029 ["equiv"] = 8801,  
3030 ["Congruent"] = 8801,  
3031 ["nequiv"] = 8802,  
3032 ["NotCongruent"] = 8802,  
3033 ["le"] = 8804,  
3034 ["leq"] = 8804,  
3035 ["ge"] = 8805,  
3036 ["GreaterEqual"] = 8805,  
3037 ["geq"] = 8805,  
3038 ["lE"] = 8806,  
3039 ["LessFullEqual"] = 8806,  
3040 ["leqq"] = 8806,  
3041 ["gE"] = 8807,  
3042 ["GreaterFullEqual"] = 8807,  
3043 ["geqq"] = 8807,  
3044 ["lnE"] = 8808,  
3045 ["lneqq"] = 8808,  
3046 ["gnE"] = 8809,  
3047 ["gneqq"] = 8809,  
3048 ["Lt"] = 8810,  
3049 ["NestedLessLess"] = 8810,  
3050 ["ll"] = 8810,  
3051 ["Gt"] = 8811,  
3052 ["NestedGreaterGreater"] = 8811,  
3053 ["gg"] = 8811,  
3054 ["twixt"] = 8812,  
3055 ["between"] = 8812,  
3056 ["NotCupCap"] = 8813,  
3057 ["nlt"] = 8814,  
3058 ["NotLess"] = 8814,  
3059 ["nless"] = 8814,  
3060 ["ngt"] = 8815,  
3061 ["NotGreater"] = 8815,  
3062 ["ngtr"] = 8815,

3063 ["nle"] = 8816,  
 3064 ["NotLessEqual"] = 8816,  
 3065 ["nleq"] = 8816,  
 3066 ["nge"] = 8817,  
 3067 ["NotGreaterEqual"] = 8817,  
 3068 ["ngeq"] = 8817,  
 3069 ["lsim"] = 8818,  
 3070 ["LessTilde"] = 8818,  
 3071 ["lesssim"] = 8818,  
 3072 ["gsim"] = 8819,  
 3073 ["gtrsim"] = 8819,  
 3074 ["GreaterTilde"] = 8819,  
 3075 ["nlsim"] = 8820,  
 3076 ["NotLessTilde"] = 8820,  
 3077 ["ngsim"] = 8821,  
 3078 ["NotGreaterTilde"] = 8821,  
 3079 ["lg"] = 8822,  
 3080 ["lessgtr"] = 8822,  
 3081 ["LessGreater"] = 8822,  
 3082 ["gl"] = 8823,  
 3083 ["gtrless"] = 8823,  
 3084 ["GreaterLess"] = 8823,  
 3085 ["ntlg"] = 8824,  
 3086 ["NotLessGreater"] = 8824,  
 3087 ["ntgl"] = 8825,  
 3088 ["NotGreaterLess"] = 8825,  
 3089 ["pr"] = 8826,  
 3090 ["Precedes"] = 8826,  
 3091 ["prec"] = 8826,  
 3092 ["sc"] = 8827,  
 3093 ["Succeeds"] = 8827,  
 3094 ["succ"] = 8827,  
 3095 ["prcue"] = 8828,  
 3096 ["PrecedesSlantEqual"] = 8828,  
 3097 ["preccurlyeq"] = 8828,  
 3098 ["sccue"] = 8829,  
 3099 ["SucceedsSlantEqual"] = 8829,  
 3100 ["succcurlyeq"] = 8829,  
 3101 ["prsim"] = 8830,  
 3102 ["precsim"] = 8830,  
 3103 ["PrecedesTilde"] = 8830,  
 3104 ["scsim"] = 8831,  
 3105 ["succsim"] = 8831,  
 3106 ["SucceedsTilde"] = 8831,  
 3107 ["npr"] = 8832,  
 3108 ["nprec"] = 8832,  
 3109 ["NotPrecedes"] = 8832,



3110 ["nsc"] = 8833,  
 3111 ["nsucc"] = 8833,  
 3112 ["NotSucceeds"] = 8833,  
 3113 ["sub"] = 8834,  
 3114 ["subset"] = 8834,  
 3115 ["sup"] = 8835,  
 3116 ["supset"] = 8835,  
 3117 ["Superset"] = 8835,  
 3118 ["nsub"] = 8836,  
 3119 ["nsup"] = 8837,  
 3120 ["sube"] = 8838,  
 3121 ["SubsetEqual"] = 8838,  
 3122 ["subseteq"] = 8838,  
 3123 ["supe"] = 8839,  
 3124 ["supseteq"] = 8839,  
 3125 ["SupersetEqual"] = 8839,  
 3126 ["nsube"] = 8840,  
 3127 ["nsubseteq"] = 8840,  
 3128 ["NotSubsetEqual"] = 8840,  
 3129 ["nsupe"] = 8841,  
 3130 ["nsupseteq"] = 8841,  
 3131 ["NotSupersetEqual"] = 8841,  
 3132 ["subne"] = 8842,  
 3133 ["subsetneq"] = 8842,  
 3134 ["supne"] = 8843,  
 3135 ["supsetneq"] = 8843,  
 3136 ["cupdot"] = 8845,  
 3137 ["uplus"] = 8846,  
 3138 ["UnionPlus"] = 8846,  
 3139 ["sqsub"] = 8847,  
 3140 ["SquareSubset"] = 8847,  
 3141 ["sqsubset"] = 8847,  
 3142 ["sqsup"] = 8848,  
 3143 ["SquareSuperset"] = 8848,  
 3144 ["sqsupset"] = 8848,  
 3145 ["sqsube"] = 8849,  
 3146 ["SquareSubsetEqual"] = 8849,  
 3147 ["sqsubseteq"] = 8849,  
 3148 ["sqsupe"] = 8850,  
 3149 ["SquareSupersetEqual"] = 8850,  
 3150 ["sqsupseteq"] = 8850,  
 3151 ["sqcap"] = 8851,  
 3152 ["SquareIntersection"] = 8851,  
 3153 ["sqcup"] = 8852,  
 3154 ["SquareUnion"] = 8852,  
 3155 ["oplus"] = 8853,  
 3156 ["CirclePlus"] = 8853,

3157 ["ominus"] = 8854,  
 3158 ["CircleMinus"] = 8854,  
 3159 ["otimes"] = 8855,  
 3160 ["CircleTimes"] = 8855,  
 3161 ["osol"] = 8856,  
 3162 ["odot"] = 8857,  
 3163 ["CircleDot"] = 8857,  
 3164 ["ocir"] = 8858,  
 3165 ["circledcirc"] = 8858,  
 3166 ["oast"] = 8859,  
 3167 ["circledast"] = 8859,  
 3168 ["odash"] = 8861,  
 3169 ["circleddash"] = 8861,  
 3170 ["plusb"] = 8862,  
 3171 ["boxplus"] = 8862,  
 3172 ["minusb"] = 8863,  
 3173 ["boxminus"] = 8863,  
 3174 ["timesb"] = 8864,  
 3175 ["boxtimes"] = 8864,  
 3176 ["sdotb"] = 8865,  
 3177 ["dotsquare"] = 8865,  
 3178 ["vdash"] = 8866,  
 3179 ["RightTee"] = 8866,  
 3180 ["dashv"] = 8867,  
 3181 ["LeftTee"] = 8867,  
 3182 ["top"] = 8868,  
 3183 ["DownTee"] = 8868,  
 3184 ["bottom"] = 8869,  
 3185 ["bot"] = 8869,  
 3186 ["perp"] = 8869,  
 3187 ["UpTee"] = 8869,  
 3188 ["models"] = 8871,  
 3189 ["vDash"] = 8872,  
 3190 ["DoubleRightTee"] = 8872,  
 3191 ["Vdash"] = 8873,  
 3192 ["Vvdash"] = 8874,  
 3193 ["VDash"] = 8875,  
 3194 ["nvdash"] = 8876,  
 3195 ["nvDash"] = 8877,  
 3196 ["nVdash"] = 8878,  
 3197 ["nVDash"] = 8879,  
 3198 ["prurel"] = 8880,  
 3199 ["vltri"] = 8882,  
 3200 ["vartriangleleft"] = 8882,  
 3201 ["LeftTriangle"] = 8882,  
 3202 ["vrtri"] = 8883,  
 3203 ["vartriangleright"] = 8883,

3204 ["RightTriangle"] = 8883,  
3205 ["ltrie"] = 8884,  
3206 ["trianglelefteq"] = 8884,  
3207 ["LeftTriangleEqual"] = 8884,  
3208 ["rtrie"] = 8885,  
3209 ["trianglerighteq"] = 8885,  
3210 ["RightTriangleEqual"] = 8885,  
3211 ["origof"] = 8886,  
3212 ["imof"] = 8887,  
3213 ["mumap"] = 8888,  
3214 ["multimap"] = 8888,  
3215 ["hercon"] = 8889,  
3216 ["intcal"] = 8890,  
3217 ["intercal"] = 8890,  
3218 ["veebar"] = 8891,  
3219 ["barvee"] = 8893,  
3220 ["angrtvb"] = 8894,  
3221 ["lrtri"] = 8895,  
3222 ["xwedge"] = 8896,  
3223 ["Wedge"] = 8896,  
3224 ["bigwedge"] = 8896,  
3225 ["xvee"] = 8897,  
3226 ["Vee"] = 8897,  
3227 ["bigvee"] = 8897,  
3228 ["xcap"] = 8898,  
3229 ["Intersection"] = 8898,  
3230 ["bigcap"] = 8898,  
3231 ["xcup"] = 8899,  
3232 ["Union"] = 8899,  
3233 ["bigcup"] = 8899,  
3234 ["diam"] = 8900,  
3235 ["diamond"] = 8900,  
3236 ["Diamond"] = 8900,  
3237 ["sdot"] = 8901,  
3238 ["sstarf"] = 8902,  
3239 ["Star"] = 8902,  
3240 ["divonx"] = 8903,  
3241 ["divideontimes"] = 8903,  
3242 ["bowtie"] = 8904,  
3243 ["ltimes"] = 8905,  
3244 ["rtimes"] = 8906,  
3245 ["lthree"] = 8907,  
3246 ["leftthreetimes"] = 8907,  
3247 ["rthree"] = 8908,  
3248 ["rightthreetimes"] = 8908,  
3249 ["bsime"] = 8909,  
3250 ["backsimeq"] = 8909,

3251 ["cuvee"] = 8910,  
 3252 ["curlyvee"] = 8910,  
 3253 ["cuwed"] = 8911,  
 3254 ["curlywedge"] = 8911,  
 3255 ["Sub"] = 8912,  
 3256 ["Subset"] = 8912,  
 3257 ["Sup"] = 8913,  
 3258 ["Supset"] = 8913,  
 3259 ["Cap"] = 8914,  
 3260 ["Cup"] = 8915,  
 3261 ["fork"] = 8916,  
 3262 ["pitchfork"] = 8916,  
 3263 ["epar"] = 8917,  
 3264 ["ltdot"] = 8918,  
 3265 ["lessdot"] = 8918,  
 3266 ["gtdot"] = 8919,  
 3267 ["gtrdot"] = 8919,  
 3268 ["Ll"] = 8920,  
 3269 ["Gg"] = 8921,  
 3270 ["ggg"] = 8921,  
 3271 ["leg"] = 8922,  
 3272 ["LessEqualGreater"] = 8922,  
 3273 ["lesseqgtr"] = 8922,  
 3274 ["gel"] = 8923,  
 3275 ["gtreqless"] = 8923,  
 3276 ["GreaterEqualLess"] = 8923,  
 3277 ["cuepr"] = 8926,  
 3278 ["curlyeqprec"] = 8926,  
 3279 ["cuesc"] = 8927,  
 3280 ["curlyeqsucc"] = 8927,  
 3281 ["nprcue"] = 8928,  
 3282 ["NotPrecedesSlantEqual"] = 8928,  
 3283 ["nsccue"] = 8929,  
 3284 ["NotSucceedsSlantEqual"] = 8929,  
 3285 ["nqsube"] = 8930,  
 3286 ["NotSquareSubsetEqual"] = 8930,  
 3287 ["nqsupe"] = 8931,  
 3288 ["NotSquareSupersetEqual"] = 8931,  
 3289 ["lnsim"] = 8934,  
 3290 ["gnsim"] = 8935,  
 3291 ["prnsim"] = 8936,  
 3292 ["precnsim"] = 8936,  
 3293 ["scnsim"] = 8937,  
 3294 ["succnsim"] = 8937,  
 3295 ["nltri"] = 8938,  
 3296 ["ntriangleleft"] = 8938,  
 3297 ["NotLeftTriangle"] = 8938,

3298 ["nrtri"] = 8939,  
 3299 ["ntriangleright"] = 8939,  
 3300 ["NotRightTriangle"] = 8939,  
 3301 ["nltrie"] = 8940,  
 3302 ["ntrianglelefteq"] = 8940,  
 3303 ["NotLeftTriangleEqual"] = 8940,  
 3304 ["nrtrie"] = 8941,  
 3305 ["ntrianglerighteq"] = 8941,  
 3306 ["NotRightTriangleEqual"] = 8941,  
 3307 ["vellip"] = 8942,  
 3308 ["ctdot"] = 8943,  
 3309 ["utdot"] = 8944,  
 3310 ["dtdot"] = 8945,  
 3311 ["disin"] = 8946,  
 3312 ["isinsv"] = 8947,  
 3313 ["isins"] = 8948,  
 3314 ["isindot"] = 8949,  
 3315 ["notinvc"] = 8950,  
 3316 ["notinvb"] = 8951,  
 3317 ["isinE"] = 8953,  
 3318 ["nisd"] = 8954,  
 3319 ["xnis"] = 8955,  
 3320 ["nis"] = 8956,  
 3321 ["notnivc"] = 8957,  
 3322 ["notnivb"] = 8958,  
 3323 ["barwed"] = 8965,  
 3324 ["barwedge"] = 8965,  
 3325 ["Barwed"] = 8966,  
 3326 ["doublebarwedge"] = 8966,  
 3327 ["lceil"] = 8968,  
 3328 ["LeftCeiling"] = 8968,  
 3329 ["rceil"] = 8969,  
 3330 ["RightCeiling"] = 8969,  
 3331 ["lfloor"] = 8970,  
 3332 ["LeftFloor"] = 8970,  
 3333 ["rfloor"] = 8971,  
 3334 ["RightFloor"] = 8971,  
 3335 ["drcrop"] = 8972,  
 3336 ["dlcrop"] = 8973,  
 3337 ["urcrop"] = 8974,  
 3338 ["ulcrop"] = 8975,  
 3339 ["bnot"] = 8976,  
 3340 ["proflin"] = 8978,  
 3341 ["profsurf"] = 8979,  
 3342 ["telrec"] = 8981,  
 3343 ["target"] = 8982,  
 3344 ["ulcorn"] = 8988,

3345 ["ulcorner"] = 8988,  
3346 ["urcorn"] = 8989,  
3347 ["urcorner"] = 8989,  
3348 ["dlcorn"] = 8990,  
3349 ["llcorner"] = 8990,  
3350 ["drcorn"] = 8991,  
3351 ["lrcorner"] = 8991,  
3352 ["frown"] = 8994,  
3353 ["sfrown"] = 8994,  
3354 ["smile"] = 8995,  
3355 ["ssmile"] = 8995,  
3356 ["cylcty"] = 9005,  
3357 ["profalar"] = 9006,  
3358 ["topbot"] = 9014,  
3359 ["ovbar"] = 9021,  
3360 ["solbar"] = 9023,  
3361 ["angzarr"] = 9084,  
3362 ["lmoust"] = 9136,  
3363 ["lmoustache"] = 9136,  
3364 ["rmoust"] = 9137,  
3365 ["rmoustache"] = 9137,  
3366 ["tbrk"] = 9140,  
3367 ["OverBracket"] = 9140,  
3368 ["bbrk"] = 9141,  
3369 ["UnderBracket"] = 9141,  
3370 ["bbrktbrk"] = 9142,  
3371 ["OverParenthesis"] = 9180,  
3372 ["UnderParenthesis"] = 9181,  
3373 ["OverBrace"] = 9182,  
3374 ["UnderBrace"] = 9183,  
3375 ["trpezium"] = 9186,  
3376 ["elinters"] = 9191,  
3377 ["blank"] = 9251,  
3378 ["oS"] = 9416,  
3379 ["circledS"] = 9416,  
3380 ["boxh"] = 9472,  
3381 ["HorizontalLine"] = 9472,  
3382 ["boxv"] = 9474,  
3383 ["boxdr"] = 9484,  
3384 ["boxdl"] = 9488,  
3385 ["boxur"] = 9492,  
3386 ["boxul"] = 9496,  
3387 ["boxvr"] = 9500,  
3388 ["boxvl"] = 9508,  
3389 ["boxhd"] = 9516,  
3390 ["boxhu"] = 9524,  
3391 ["boxvh"] = 9532,

3392 ["boxH"] = 9552,  
3393 ["boxV"] = 9553,  
3394 ["boxdR"] = 9554,  
3395 ["boxDr"] = 9555,  
3396 ["boxDR"] = 9556,  
3397 ["boxdL"] = 9557,  
3398 ["boxDl"] = 9558,  
3399 ["boxDL"] = 9559,  
3400 ["boxuR"] = 9560,  
3401 ["boxUr"] = 9561,  
3402 ["boxUR"] = 9562,  
3403 ["boxuL"] = 9563,  
3404 ["boxUl"] = 9564,  
3405 ["boxUL"] = 9565,  
3406 ["boxvR"] = 9566,  
3407 ["boxVr"] = 9567,  
3408 ["boxVR"] = 9568,  
3409 ["boxvL"] = 9569,  
3410 ["boxVl"] = 9570,  
3411 ["boxVL"] = 9571,  
3412 ["boxHd"] = 9572,  
3413 ["boxhD"] = 9573,  
3414 ["boxHD"] = 9574,  
3415 ["boxHu"] = 9575,  
3416 ["boxhU"] = 9576,  
3417 ["boxHU"] = 9577,  
3418 ["boxvH"] = 9578,  
3419 ["boxVh"] = 9579,  
3420 ["boxVH"] = 9580,  
3421 ["uhblk"] = 9600,  
3422 ["lhblk"] = 9604,  
3423 ["block"] = 9608,  
3424 ["blk14"] = 9617,  
3425 ["blk12"] = 9618,  
3426 ["blk34"] = 9619,  
3427 ["squ"] = 9633,  
3428 ["square"] = 9633,  
3429 ["Square"] = 9633,  
3430 ["squf"] = 9642,  
3431 ["squarf"] = 9642,  
3432 ["blacksquare"] = 9642,  
3433 ["FilledVerySmallSquare"] = 9642,  
3434 ["EmptyVerySmallSquare"] = 9643,  
3435 ["rect"] = 9645,  
3436 ["marker"] = 9646,  
3437 ["fltns"] = 9649,  
3438 ["xutri"] = 9651,

3439 ["bigtriangleup"] = 9651,  
3440 ["utrif"] = 9652,  
3441 ["blacktriangle"] = 9652,  
3442 ["utri"] = 9653,  
3443 ["triangle"] = 9653,  
3444 ["rtrif"] = 9656,  
3445 ["blacktriangleright"] = 9656,  
3446 ["rtri"] = 9657,  
3447 ["triangleright"] = 9657,  
3448 ["xdtri"] = 9661,  
3449 ["bigtriangledown"] = 9661,  
3450 ["dtrif"] = 9662,  
3451 ["blacktriangledown"] = 9662,  
3452 ["dtri"] = 9663,  
3453 ["triangledown"] = 9663,  
3454 ["ltrif"] = 9666,  
3455 ["blacktriangleleft"] = 9666,  
3456 ["ltri"] = 9667,  
3457 ["triangleleft"] = 9667,  
3458 ["loz"] = 9674,  
3459 ["lozenge"] = 9674,  
3460 ["cir"] = 9675,  
3461 ["tridot"] = 9708,  
3462 ["xcirc"] = 9711,  
3463 ["bigcirc"] = 9711,  
3464 ["ultri"] = 9720,  
3465 ["urtri"] = 9721,  
3466 ["lltri"] = 9722,  
3467 ["EmptySmallSquare"] = 9723,  
3468 ["FilledSmallSquare"] = 9724,  
3469 ["starf"] = 9733,  
3470 ["bigstar"] = 9733,  
3471 ["star"] = 9734,  
3472 ["phone"] = 9742,  
3473 ["female"] = 9792,  
3474 ["male"] = 9794,  
3475 ["spades"] = 9824,  
3476 ["spadesuit"] = 9824,  
3477 ["clubs"] = 9827,  
3478 ["clubsuit"] = 9827,  
3479 ["hearts"] = 9829,  
3480 ["heartsuit"] = 9829,  
3481 ["diams"] = 9830,  
3482 ["diamondsuit"] = 9830,  
3483 ["sung"] = 9834,  
3484 ["flat"] = 9837,  
3485 ["natur"] = 9838,



3486 ["natural"] = 9838,  
 3487 ["sharp"] = 9839,  
 3488 ["check"] = 10003,  
 3489 ["checkmark"] = 10003,  
 3490 ["cross"] = 10007,  
 3491 ["malt"] = 10016,  
 3492 ["maltese"] = 10016,  
 3493 ["sext"] = 10038,  
 3494 ["VerticalSeparator"] = 10072,  
 3495 ["lbrk"] = 10098,  
 3496 ["rbrk"] = 10099,  
 3497 ["lobrk"] = 10214,  
 3498 ["LeftDoubleBracket"] = 10214,  
 3499 ["robrk"] = 10215,  
 3500 ["RightDoubleBracket"] = 10215,  
 3501 ["lang"] = 10216,  
 3502 ["LeftAngleBracket"] = 10216,  
 3503 ["langle"] = 10216,  
 3504 ["rang"] = 10217,  
 3505 ["RightAngleBracket"] = 10217,  
 3506 ["rangle"] = 10217,  
 3507 ["Lang"] = 10218,  
 3508 ["Rang"] = 10219,  
 3509 ["loang"] = 10220,  
 3510 ["roang"] = 10221,  
 3511 ["xlarr"] = 10229,  
 3512 ["longleftarrow"] = 10229,  
 3513 ["LongLeftArrow"] = 10229,  
 3514 ["xrarr"] = 10230,  
 3515 ["longrightarrow"] = 10230,  
 3516 ["LongRightArrow"] = 10230,  
 3517 ["xharr"] = 10231,  
 3518 ["longleftrightarrow"] = 10231,  
 3519 ["LongLeftRightArrow"] = 10231,  
 3520 ["xlArr"] = 10232,  
 3521 ["Longleftarrow"] = 10232,  
 3522 ["DoubleLongLeftArrow"] = 10232,  
 3523 ["xrArr"] = 10233,  
 3524 ["Longrightarrow"] = 10233,  
 3525 ["DoubleLongRightArrow"] = 10233,  
 3526 ["xhArr"] = 10234,  
 3527 ["Longleftrightarrow"] = 10234,  
 3528 ["DoubleLongLeftRightArrow"] = 10234,  
 3529 ["xmap"] = 10236,  
 3530 ["longmapsto"] = 10236,  
 3531 ["dzigrarr"] = 10239,  
 3532 ["nvlArr"] = 10498,

3533 ["nvrArr"] = 10499,  
3534 ["nvHarr"] = 10500,  
3535 ["Map"] = 10501,  
3536 ["lbarr"] = 10508,  
3537 ["rbarr"] = 10509,  
3538 ["bkarow"] = 10509,  
3539 ["lBarr"] = 10510,  
3540 ["rBarr"] = 10511,  
3541 ["dbkarow"] = 10511,  
3542 ["RBarr"] = 10512,  
3543 ["drbkarow"] = 10512,  
3544 ["DDotrahd"] = 10513,  
3545 ["UpArrowBar"] = 10514,  
3546 ["DownArrowBar"] = 10515,  
3547 ["Rarrtl"] = 10518,  
3548 ["latail"] = 10521,  
3549 ["ratail"] = 10522,  
3550 ["lAtail"] = 10523,  
3551 ["rAtail"] = 10524,  
3552 ["larrfs"] = 10525,  
3553 ["rarrfs"] = 10526,  
3554 ["larrbfs"] = 10527,  
3555 ["rarrbfs"] = 10528,  
3556 ["nwarhk"] = 10531,  
3557 ["nearhk"] = 10532,  
3558 ["searhk"] = 10533,  
3559 ["hksearow"] = 10533,  
3560 ["swarhk"] = 10534,  
3561 ["hkswarow"] = 10534,  
3562 ["nwnear"] = 10535,  
3563 ["nesear"] = 10536,  
3564 ["toea"] = 10536,  
3565 ["seswar"] = 10537,  
3566 ["tosa"] = 10537,  
3567 ["swnwar"] = 10538,  
3568 ["rarrc"] = 10547,  
3569 ["cudarr"] = 10549,  
3570 ["ldca"] = 10550,  
3571 ["rdca"] = 10551,  
3572 ["cudarrl"] = 10552,  
3573 ["larrpl"] = 10553,  
3574 ["curarrm"] = 10556,  
3575 ["cularrp"] = 10557,  
3576 ["rarrpl"] = 10565,  
3577 ["harrcir"] = 10568,  
3578 ["Uarroccir"] = 10569,  
3579 ["lurdshar"] = 10570,

3580 ["ldrushar"] = 10571,  
3581 ["LeftRightVector"] = 10574,  
3582 ["RightUpDownVector"] = 10575,  
3583 ["DownLeftRightVector"] = 10576,  
3584 ["LeftUpDownVector"] = 10577,  
3585 ["LeftVectorBar"] = 10578,  
3586 ["RightVectorBar"] = 10579,  
3587 ["RightUpVectorBar"] = 10580,  
3588 ["RightDownVectorBar"] = 10581,  
3589 ["DownLeftVectorBar"] = 10582,  
3590 ["DownRightVectorBar"] = 10583,  
3591 ["LeftUpVectorBar"] = 10584,  
3592 ["LeftDownVectorBar"] = 10585,  
3593 ["LeftTeeVector"] = 10586,  
3594 ["RightTeeVector"] = 10587,  
3595 ["RightUpTeeVector"] = 10588,  
3596 ["RightDownTeeVector"] = 10589,  
3597 ["DownLeftTeeVector"] = 10590,  
3598 ["DownRightTeeVector"] = 10591,  
3599 ["LeftUpTeeVector"] = 10592,  
3600 ["LeftDownTeeVector"] = 10593,  
3601 ["lHar"] = 10594,  
3602 ["uHar"] = 10595,  
3603 ["rHar"] = 10596,  
3604 ["dHar"] = 10597,  
3605 ["luruhar"] = 10598,  
3606 ["ldrdhar"] = 10599,  
3607 ["ruluhar"] = 10600,  
3608 ["rdldhar"] = 10601,  
3609 ["lharul"] = 10602,  
3610 ["llhard"] = 10603,  
3611 ["rharul"] = 10604,  
3612 ["lrhard"] = 10605,  
3613 ["udhar"] = 10606,  
3614 ["UpEquilibrium"] = 10606,  
3615 ["duhar"] = 10607,  
3616 ["ReverseUpEquilibrium"] = 10607,  
3617 ["RoundImplies"] = 10608,  
3618 ["erarr"] = 10609,  
3619 ["simrarr"] = 10610,  
3620 ["larrsim"] = 10611,  
3621 ["rarrsim"] = 10612,  
3622 ["rarrap"] = 10613,  
3623 ["ltlarr"] = 10614,  
3624 ["gtrarr"] = 10616,  
3625 ["subrarr"] = 10617,  
3626 ["suplarr"] = 10619,

3627 ["lfisht"] = 10620,  
3628 ["rfisht"] = 10621,  
3629 ["ufisht"] = 10622,  
3630 ["dfisht"] = 10623,  
3631 ["lopar"] = 10629,  
3632 ["ropar"] = 10630,  
3633 ["lbrke"] = 10635,  
3634 ["rbrke"] = 10636,  
3635 ["lbrkslu"] = 10637,  
3636 ["rbrksld"] = 10638,  
3637 ["lbrksld"] = 10639,  
3638 ["rbrkslu"] = 10640,  
3639 ["langd"] = 10641,  
3640 ["rangd"] = 10642,  
3641 ["lparlt"] = 10643,  
3642 ["rpargt"] = 10644,  
3643 ["gtlPar"] = 10645,  
3644 ["ltrPar"] = 10646,  
3645 ["vzigzag"] = 10650,  
3646 ["vangrt"] = 10652,  
3647 ["angrtvbd"] = 10653,  
3648 ["ange"] = 10660,  
3649 ["range"] = 10661,  
3650 ["dwangle"] = 10662,  
3651 ["uwangle"] = 10663,  
3652 ["angmsdaa"] = 10664,  
3653 ["angmsdab"] = 10665,  
3654 ["angmsdac"] = 10666,  
3655 ["angmsdad"] = 10667,  
3656 ["angmsdae"] = 10668,  
3657 ["angmsdaf"] = 10669,  
3658 ["angmsdag"] = 10670,  
3659 ["angmsdah"] = 10671,  
3660 ["bemptyv"] = 10672,  
3661 ["demptyv"] = 10673,  
3662 ["cemptyv"] = 10674,  
3663 ["raemptyv"] = 10675,  
3664 ["laemptyv"] = 10676,  
3665 ["ohbar"] = 10677,  
3666 ["omid"] = 10678,  
3667 ["opar"] = 10679,  
3668 ["operp"] = 10681,  
3669 ["olcross"] = 10683,  
3670 ["odsold"] = 10684,  
3671 ["olcir"] = 10686,  
3672 ["ofcir"] = 10687,  
3673 ["olt"] = 10688,

3674 ["ogt"] = 10689,  
 3675 ["cirscir"] = 10690,  
 3676 ["cirE"] = 10691,  
 3677 ["solb"] = 10692,  
 3678 ["bsolb"] = 10693,  
 3679 ["boxbox"] = 10697,  
 3680 ["trisb"] = 10701,  
 3681 ["rtriltri"] = 10702,  
 3682 ["LeftTriangleBar"] = 10703,  
 3683 ["RightTriangleBar"] = 10704,  
 3684 ["race"] = 10714,  
 3685 ["iinfin"] = 10716,  
 3686 ["infintie"] = 10717,  
 3687 ["nvinfin"] = 10718,  
 3688 ["eparsl"] = 10723,  
 3689 ["smeparsl"] = 10724,  
 3690 ["eqvparsl"] = 10725,  
 3691 ["lozf"] = 10731,  
 3692 ["blacklozenge"] = 10731,  
 3693 ["RuleDelayed"] = 10740,  
 3694 ["dsol"] = 10742,  
 3695 ["xodot"] = 10752,  
 3696 ["bigodot"] = 10752,  
 3697 ["xoplus"] = 10753,  
 3698 ["bigoplus"] = 10753,  
 3699 ["xotime"] = 10754,  
 3700 ["bigotimes"] = 10754,  
 3701 ["xuplus"] = 10756,  
 3702 ["biguplus"] = 10756,  
 3703 ["xsqcup"] = 10758,  
 3704 ["bigsqcup"] = 10758,  
 3705 ["qint"] = 10764,  
 3706 ["iiiint"] = 10764,  
 3707 ["fpartint"] = 10765,  
 3708 ["cirfnint"] = 10768,  
 3709 ["awint"] = 10769,  
 3710 ["rppolint"] = 10770,  
 3711 ["scpolint"] = 10771,  
 3712 ["npolint"] = 10772,  
 3713 ["pointint"] = 10773,  
 3714 ["quatint"] = 10774,  
 3715 ["intlarhk"] = 10775,  
 3716 ["pluscir"] = 10786,  
 3717 ["plusacir"] = 10787,  
 3718 ["simplus"] = 10788,  
 3719 ["plusdu"] = 10789,  
 3720 ["plussim"] = 10790,

3721 ["plustwo"] = 10791,  
 3722 ["mcomma"] = 10793,  
 3723 ["minusdu"] = 10794,  
 3724 ["loplus"] = 10797,  
 3725 ["roplus"] = 10798,  
 3726 ["Cross"] = 10799,  
 3727 ["timesd"] = 10800,  
 3728 ["timesbar"] = 10801,  
 3729 ["smashp"] = 10803,  
 3730 ["lotimes"] = 10804,  
 3731 ["rotimes"] = 10805,  
 3732 ["otimesas"] = 10806,  
 3733 ["Otimes"] = 10807,  
 3734 ["odiv"] = 10808,  
 3735 ["triplus"] = 10809,  
 3736 ["triminus"] = 10810,  
 3737 ["tritime"] = 10811,  
 3738 ["iproduct"] = 10812,  
 3739 ["intprod"] = 10812,  
 3740 ["amalg"] = 10815,  
 3741 ["capdot"] = 10816,  
 3742 ["ncup"] = 10818,  
 3743 ["ncap"] = 10819,  
 3744 ["capand"] = 10820,  
 3745 ["cupor"] = 10821,  
 3746 ["cupcap"] = 10822,  
 3747 ["capcup"] = 10823,  
 3748 ["cupbrcap"] = 10824,  
 3749 ["capbrcup"] = 10825,  
 3750 ["cupcup"] = 10826,  
 3751 ["capcap"] = 10827,  
 3752 ["ccups"] = 10828,  
 3753 ["ccaps"] = 10829,  
 3754 ["ccupssm"] = 10832,  
 3755 ["And"] = 10835,  
 3756 ["Or"] = 10836,  
 3757 ["andand"] = 10837,  
 3758 ["oror"] = 10838,  
 3759 ["orslope"] = 10839,  
 3760 ["andslope"] = 10840,  
 3761 ["andv"] = 10842,  
 3762 ["orv"] = 10843,  
 3763 ["andd"] = 10844,  
 3764 ["ord"] = 10845,  
 3765 ["wedbar"] = 10847,  
 3766 ["sdote"] = 10854,  
 3767 ["simdot"] = 10858,

3768 ["congdot"] = 10861,  
 3769 ["easter"] = 10862,  
 3770 ["apacir"] = 10863,  
 3771 ["apE"] = 10864,  
 3772 ["eplus"] = 10865,  
 3773 ["pluse"] = 10866,  
 3774 ["Esim"] = 10867,  
 3775 ["Colone"] = 10868,  
 3776 ["Equal"] = 10869,  
 3777 ["eDDot"] = 10871,  
 3778 ["ddotseq"] = 10871,  
 3779 ["equivDD"] = 10872,  
 3780 ["ltcir"] = 10873,  
 3781 ["gtcir"] = 10874,  
 3782 ["ltquest"] = 10875,  
 3783 ["gtquest"] = 10876,  
 3784 ["les"] = 10877,  
 3785 ["LessSlantEqual"] = 10877,  
 3786 ["leqslant"] = 10877,  
 3787 ["ges"] = 10878,  
 3788 ["GreaterSlantEqual"] = 10878,  
 3789 ["geqslant"] = 10878,  
 3790 ["lesdot"] = 10879,  
 3791 ["gesdot"] = 10880,  
 3792 ["lesdoto"] = 10881,  
 3793 ["gesdoto"] = 10882,  
 3794 ["lesdotor"] = 10883,  
 3795 ["gesdotor"] = 10884,  
 3796 ["lap"] = 10885,  
 3797 ["lessapprox"] = 10885,  
 3798 ["gap"] = 10886,  
 3799 ["gtrapprox"] = 10886,  
 3800 ["lne"] = 10887,  
 3801 ["lneq"] = 10887,  
 3802 ["gne"] = 10888,  
 3803 ["gneq"] = 10888,  
 3804 ["lnap"] = 10889,  
 3805 ["lnapprox"] = 10889,  
 3806 ["gnap"] = 10890,  
 3807 ["gnapprox"] = 10890,  
 3808 ["lEg"] = 10891,  
 3809 ["lesseqqgtr"] = 10891,  
 3810 ["gEl"] = 10892,  
 3811 ["gtreqqless"] = 10892,  
 3812 ["lsime"] = 10893,  
 3813 ["gsime"] = 10894,  
 3814 ["lsimg"] = 10895,

3815 ["gsiml"] = 10896,  
 3816 ["lgE"] = 10897,  
 3817 ["glE"] = 10898,  
 3818 ["lesges"] = 10899,  
 3819 ["gesles"] = 10900,  
 3820 ["els"] = 10901,  
 3821 ["eqslantless"] = 10901,  
 3822 ["egs"] = 10902,  
 3823 ["eqslantgtr"] = 10902,  
 3824 ["elsdot"] = 10903,  
 3825 ["egsdot"] = 10904,  
 3826 ["el"] = 10905,  
 3827 ["eg"] = 10906,  
 3828 ["siml"] = 10909,  
 3829 ["simg"] = 10910,  
 3830 ["simlE"] = 10911,  
 3831 ["simgE"] = 10912,  
 3832 ["LessLess"] = 10913,  
 3833 ["GreaterGreater"] = 10914,  
 3834 ["glj"] = 10916,  
 3835 ["gla"] = 10917,  
 3836 ["ltcc"] = 10918,  
 3837 ["gtcc"] = 10919,  
 3838 ["lescc"] = 10920,  
 3839 ["gescc"] = 10921,  
 3840 ["smt"] = 10922,  
 3841 ["lat"] = 10923,  
 3842 ["smtE"] = 10924,  
 3843 ["late"] = 10925,  
 3844 ["bumpE"] = 10926,  
 3845 ["pre"] = 10927,  
 3846 ["preceq"] = 10927,  
 3847 ["PrecedesEqual"] = 10927,  
 3848 ["sce"] = 10928,  
 3849 ["succeq"] = 10928,  
 3850 ["SucceedsEqual"] = 10928,  
 3851 ["prE"] = 10931,  
 3852 ["scE"] = 10932,  
 3853 ["prnE"] = 10933,  
 3854 ["precneqq"] = 10933,  
 3855 ["scnE"] = 10934,  
 3856 ["succneqq"] = 10934,  
 3857 ["prap"] = 10935,  
 3858 ["precapprox"] = 10935,  
 3859 ["scap"] = 10936,  
 3860 ["succapprox"] = 10936,  
 3861 ["prnap"] = 10937,



3862 ["precnapprox"] = 10937,  
 3863 ["scnap"] = 10938,  
 3864 ["succnapprox"] = 10938,  
 3865 ["Pr"] = 10939,  
 3866 ["Sc"] = 10940,  
 3867 ["subdot"] = 10941,  
 3868 ["supdot"] = 10942,  
 3869 ["subplus"] = 10943,  
 3870 ["supplus"] = 10944,  
 3871 ["submult"] = 10945,  
 3872 ["supmult"] = 10946,  
 3873 ["subedot"] = 10947,  
 3874 ["supedot"] = 10948,  
 3875 ["subE"] = 10949,  
 3876 ["subseteqq"] = 10949,  
 3877 ["supE"] = 10950,  
 3878 ["supseteqq"] = 10950,  
 3879 ["subsim"] = 10951,  
 3880 ["supsim"] = 10952,  
 3881 ["subnE"] = 10955,  
 3882 ["subsetneqq"] = 10955,  
 3883 ["supnE"] = 10956,  
 3884 ["supsetneqq"] = 10956,  
 3885 ["csub"] = 10959,  
 3886 ["csup"] = 10960,  
 3887 ["csube"] = 10961,  
 3888 ["csupe"] = 10962,  
 3889 ["subsup"] = 10963,  
 3890 ["supsub"] = 10964,  
 3891 ["subsub"] = 10965,  
 3892 ["supsup"] = 10966,  
 3893 ["suphsub"] = 10967,  
 3894 ["supdsub"] = 10968,  
 3895 ["forkv"] = 10969,  
 3896 ["topfork"] = 10970,  
 3897 ["mlcp"] = 10971,  
 3898 ["Dashv"] = 10980,  
 3899 ["DoubleLeftTee"] = 10980,  
 3900 ["Vdashl"] = 10982,  
 3901 ["Barv"] = 10983,  
 3902 ["vBar"] = 10984,  
 3903 ["vBarv"] = 10985,  
 3904 ["Vbar"] = 10987,  
 3905 ["Not"] = 10988,  
 3906 ["bNot"] = 10989,  
 3907 ["rnmid"] = 10990,  
 3908 ["cirmid"] = 10991,

3909 ["midcir"] = 10992,  
3910 ["topcir"] = 10993,  
3911 ["nhpar"] = 10994,  
3912 ["parsim"] = 10995,  
3913 ["parsl"] = 11005,  
3914 ["fflig"] = 64256,  
3915 ["filig"] = 64257,  
3916 ["fllig"] = 64258,  
3917 ["ffilig"] = 64259,  
3918 ["ffllig"] = 64260,  
3919 ["Ascr"] = 119964,  
3920 ["Cscr"] = 119966,  
3921 ["Dscr"] = 119967,  
3922 ["Gscr"] = 119970,  
3923 ["Jscr"] = 119973,  
3924 ["Kscr"] = 119974,  
3925 ["Nscr"] = 119977,  
3926 ["Oscr"] = 119978,  
3927 ["Pscr"] = 119979,  
3928 ["Qscr"] = 119980,  
3929 ["Sscr"] = 119982,  
3930 ["Tscr"] = 119983,  
3931 ["Uscr"] = 119984,  
3932 ["Vscr"] = 119985,  
3933 ["Wscr"] = 119986,  
3934 ["Xscr"] = 119987,  
3935 ["Yscr"] = 119988,  
3936 ["Zscr"] = 119989,  
3937 ["ascr"] = 119990,  
3938 ["bscr"] = 119991,  
3939 ["cscr"] = 119992,  
3940 ["dscr"] = 119993,  
3941 ["fscr"] = 119995,  
3942 ["hscr"] = 119997,  
3943 ["iscr"] = 119998,  
3944 ["jscr"] = 119999,  
3945 ["kscr"] = 120000,  
3946 ["lscr"] = 120001,  
3947 ["mscr"] = 120002,  
3948 ["nscr"] = 120003,  
3949 ["pscr"] = 120005,  
3950 ["qscr"] = 120006,  
3951 ["rscr"] = 120007,  
3952 ["sscr"] = 120008,  
3953 ["tscr"] = 120009,  
3954 ["uscr"] = 120010,  
3955 ["vscr"] = 120011,

3956 ["wscr"] = 120012,  
3957 ["xscr"] = 120013,  
3958 ["yscr"] = 120014,  
3959 ["zscr"] = 120015,  
3960 ["Afr"] = 120068,  
3961 ["Bfr"] = 120069,  
3962 ["Dfr"] = 120071,  
3963 ["Efr"] = 120072,  
3964 ["Ffr"] = 120073,  
3965 ["Gfr"] = 120074,  
3966 ["Jfr"] = 120077,  
3967 ["Kfr"] = 120078,  
3968 ["Lfr"] = 120079,  
3969 ["Mfr"] = 120080,  
3970 ["Nfr"] = 120081,  
3971 ["Ofr"] = 120082,  
3972 ["Pfr"] = 120083,  
3973 ["Qfr"] = 120084,  
3974 ["Sfr"] = 120086,  
3975 ["Tfr"] = 120087,  
3976 ["Ufr"] = 120088,  
3977 ["Vfr"] = 120089,  
3978 ["Wfr"] = 120090,  
3979 ["Xfr"] = 120091,  
3980 ["Yfr"] = 120092,  
3981 ["afr"] = 120094,  
3982 ["bfr"] = 120095,  
3983 ["cfr"] = 120096,  
3984 ["dfr"] = 120097,  
3985 ["efr"] = 120098,  
3986 ["ffr"] = 120099,  
3987 ["gfr"] = 120100,  
3988 ["hfr"] = 120101,  
3989 ["ifr"] = 120102,  
3990 ["jfr"] = 120103,  
3991 ["kfr"] = 120104,  
3992 ["lfr"] = 120105,  
3993 ["mfr"] = 120106,  
3994 ["nfr"] = 120107,  
3995 ["ofr"] = 120108,  
3996 ["pfr"] = 120109,  
3997 ["qfr"] = 120110,  
3998 ["rfr"] = 120111,  
3999 ["sfr"] = 120112,  
4000 ["tfr"] = 120113,  
4001 ["ufr"] = 120114,  
4002 ["vfr"] = 120115,

4003 ["wfr"] = 120116,  
4004 ["xfr"] = 120117,  
4005 ["yfr"] = 120118,  
4006 ["zfr"] = 120119,  
4007 ["Aopf"] = 120120,  
4008 ["Bopf"] = 120121,  
4009 ["Dopf"] = 120123,  
4010 ["Eopf"] = 120124,  
4011 ["Fopf"] = 120125,  
4012 ["Gopf"] = 120126,  
4013 ["Iopf"] = 120128,  
4014 ["Jopf"] = 120129,  
4015 ["Kopf"] = 120130,  
4016 ["Lopf"] = 120131,  
4017 ["Mopf"] = 120132,  
4018 ["Oopf"] = 120134,  
4019 ["Sopf"] = 120138,  
4020 ["Topf"] = 120139,  
4021 ["Uopf"] = 120140,  
4022 ["Vopf"] = 120141,  
4023 ["Wopf"] = 120142,  
4024 ["Xopf"] = 120143,  
4025 ["Yopf"] = 120144,  
4026 ["aopf"] = 120146,  
4027 ["bopf"] = 120147,  
4028 ["copf"] = 120148,  
4029 ["dopf"] = 120149,  
4030 ["eopf"] = 120150,  
4031 ["fopf"] = 120151,  
4032 ["gopf"] = 120152,  
4033 ["hopf"] = 120153,  
4034 ["iopf"] = 120154,  
4035 ["jopf"] = 120155,  
4036 ["kopf"] = 120156,  
4037 ["lopf"] = 120157,  
4038 ["mopf"] = 120158,  
4039 ["nopf"] = 120159,  
4040 ["oopf"] = 120160,  
4041 ["popf"] = 120161,  
4042 ["qopf"] = 120162,  
4043 ["ropf"] = 120163,  
4044 ["sopf"] = 120164,  
4045 ["topf"] = 120165,  
4046 ["uopf"] = 120166,  
4047 ["vopf"] = 120167,  
4048 ["wopf"] = 120168,  
4049 ["xopf"] = 120169,

```

4050 ["yopf"] = 120170,
4051 ["zopf"] = 120171,
4052 }

```

Given a string `s` of decimal digits, the `entities.dec_entity` returns the corresponding UTF8-encoded Unicode codepoint.

```

4053 function entities.dec_entity(s)
4054     return unicode.utf8.char(tonumber(s))
4055 end

```

Given a string `s` of hexadecimal digits, the `entities.hex_entity` returns the corresponding UTF8-encoded Unicode codepoint.

```

4056 function entities.hex_entity(s)
4057     return unicode.utf8.char(tonumber("0x"..s))
4058 end

```

Given a character entity name `s` (like `ouml`), the `entities.char_entity` returns the corresponding UTF8-encoded Unicode codepoint.

```

4059 function entities.char_entity(s)
4060     local n = character_entities[s]
4061     if n == nil then
4062         return "&" .. s .. ";"
4063     end
4064     return unicode.utf8.char(n)
4065 end

```

### 3.1.3 Plain T<sub>E</sub>X Writer

This section documents the `writer` object, which implements the routines for producing the T<sub>E</sub>X output. The object is an amalgamate of the generic, T<sub>E</sub>X, L<sup>A</sup>T<sub>E</sub>X writer objects that were located in the `lunamark/writer/generic.lua`, `lunamark/writer/tex.lua`, and `lunamark/writer/latex.lua` files in the Luna-mark Lua module.

Although not specified in the Lua interface (see Section 2.1), the `writer` object is exported, so that the curious user could easily tinker with the methods of the objects produced by the `writer.new` method described below. The user should be aware, however, that the implementation may change in a future revision.

```

4066 M.writer = {}

```

The `writer.new` method creates and returns a new T<sub>E</sub>X writer object associated with the Lua interface options (see Section 2.1.2) `options`. When `options` are unspecified, it is assumed that an empty table was passed to the method.

The objects produced by the `writer.new` method expose instance methods and variables of their own. As a convention, I will refer to these *member*s as `writer->member`. All member variables are immutable unless explicitly stated otherwise.

```

4067 function M.writer.new(options)
4068   local self = {}
4069   options = options or {}

```

Make the `options` table inherit from the `defaultOptions` table.

```

4070   setmetatable(options, { __index = function (_, key)
4071     return defaultOptions[key] end })

```

Parse the `slice` option and define `writer->slice_begin` `writer->slice_end`, and `writer->is_writing`. The `writer->is_writing` member variable is mutable.

```

4072   local slice_specifiers = {}
4073   for specifier in options.slice:gmatch("[^%s]+") do
4074     table.insert(slice_specifiers, specifier)
4075   end
4076
4077   if #slice_specifiers == 2 then
4078     self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
4079     local slice_begin_type = self.slice_begin:sub(1, 1)
4080     if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
4081       self.slice_begin = "^" .. self.slice_begin
4082     end
4083     local slice_end_type = self.slice_end:sub(1, 1)
4084     if slice_end_type ~= "^" and slice_end_type ~= "$" then
4085       self.slice_end = "$" .. self.slice_end
4086     end
4087   elseif #slice_specifiers == 1 then
4088     self.slice_begin = "^" .. slice_specifiers[1]
4089     self.slice_end = "$" .. slice_specifiers[1]
4090   end
4091
4092   if self.slice_begin == "^" and self.slice_end ~= "^" then
4093     self.is_writing = true
4094   else
4095     self.is_writing = false
4096   end

```

Define `writer->suffix` as the suffix of the produced cache files.

```

4097   self.suffix = ".tex"

```

Define `writer->space` as the output format of a space character.

```

4098   self.space = " "

```

Define `writer->nbspace` as the output format of a non-breaking space character.

```

4099   self.nbsp = "\\markdownRendererNbsp{}"

```

Define `writer->plain` as a function that will transform an input plain text block `s` to the output format.

```

4100   function self.plain(s)
4101     return s
4102   end

```

Define `writer->paragraph` as a function that will transform an input paragraph `s` to the output format.

```
4103 function self.paragraph(s)
4104     if not self.is_writing then return "" end
4105     return s
4106 end
```

Define `writer->pack` as a function that will take the filename `name` of the output file prepared by the reader and transform it to the output format.

```
4107 function self.pack(name)
4108     return [[\input ]] .. name .. [[\relax]]
4109 end
```

Define `writer->interblocksep` as the output format of a block element separator.

```
4110 function self.interblocksep()
4111     if not self.is_writing then return "" end
4112     return "\\markdownRendererInterblockSeparator\n{}"
4113 end
```

Define `writer->linebreak` as the output format of a forced line break.

```
4114 self.linebreak = "\\markdownRendererLineBreak\n{}"
```

Define `writer->ellipsis` as the output format of an ellipsis.

```
4115 self.ellipsis = "\\markdownRendererEllipsis{}"
```

Define `writer->hrule` as the output format of a horizontal rule.

```
4116 function self.hrule()
4117     if not self.is_writing then return "" end
4118     return "\\markdownRendererHorizontalRule{}"
4119 end
```

Define tables `escaped_uri_chars`, `escaped_citation_chars`, and `escaped_minimal_strings` containing the mapping from special plain characters and character strings that always need to be escaped.

```
4120 local escaped_uri_chars = {
4121     ["{"] = "\\markdownRendererLeftBrace{}",
4122     ["}"] = "\\markdownRendererRightBrace{}",
4123     ["%"] = "\\markdownRendererPercentSign{}",
4124     ["\\"] = "\\markdownRendererBackslash{}",
4125 }
4126 local escaped_citation_chars = {
4127     ["{"] = "\\markdownRendererLeftBrace{}",
4128     ["}"] = "\\markdownRendererRightBrace{}",
4129     ["%"] = "\\markdownRendererPercentSign{}",
4130     ["\\"] = "\\markdownRendererBackslash{}",
4131     ["#"] = "\\markdownRendererHash{}",
4132 }
4133 local escaped_minimal_strings = {
4134     ["^^"] = "\\markdownRendererCircumflex\\markdownRendererCircumflex ",
```

```

4135     ["☒"] = "\\markdownRendererTickedBox{}",
4136     ["☐"] = "\\markdownRendererHalfTickedBox{}",
4137     ["□"] = "\\markdownRendererUntickedBox{}",
4138   }

```

Define a table `escaped_chars` containing the mapping from special plain T<sub>E</sub>X characters (including the active pipe character (|) of ConT<sub>E</sub>Xt) that need to be escaped for typeset content.

```

4139   local escaped_chars = {
4140     ["{"] = "\\markdownRendererLeftBrace{}",
4141     ["}"] = "\\markdownRendererRightBrace{}",
4142     ["%"] = "\\markdownRendererPercentSign{}",
4143     ["\\"] = "\\markdownRendererBackslash{}",
4144     ["#"] = "\\markdownRendererHash{}",
4145     ["$"] = "\\markdownRendererDollarSign{}",
4146     ["&"] = "\\markdownRendererAmpersand{}",
4147     ["_"] = "\\markdownRendererUnderscore{}",
4148     ["^"] = "\\markdownRendererCircumflex{}",
4149     ["~"] = "\\markdownRendererTilde{}",
4150     ["|"] = "\\markdownRendererPipe{}",
4151   }

```

Use the `escaped_chars`, `escaped_uri_chars`, `escaped_citation_chars`, and `escaped_minimal_strings` tables to create the `escape`, `escape_citation`, `escape_uri`, and `escape_minimal` escaper functions.

```

4152   local escape = util.escaper(escaped_chars, escaped_minimal_strings)
4153   local escape_citation = util.escaper(escaped_citation_chars,
4154     escaped_minimal_strings)
4155   local escape_uri = util.escaper(escaped_uri_chars, escaped_minimal_strings)
4156   local escape_minimal = util.escaper({}, escaped_minimal_strings)

```

Define `writer->string` as a function that will transform an input plain text span `s` to the output format, `writer->citation` as a function that will transform an input citation name `c` to the output format, and `writer->uri` as a function that will transform an input URI `u` to the output format. If the `hybrid` option is enabled, use the `escape_minimal`. Otherwise, use the `escape`, `escape_citation`, and `escape_uri` functions.

```

4157   if options.hybrid then
4158     self.string = escape_minimal
4159     self.citation = escape_minimal
4160     self.uri = escape_minimal
4161   else
4162     self.string = escape
4163     self.citation = escape_citation
4164     self.uri = escape_uri
4165   end

```



Define `writer->escape` as a function that will transform an input plain text span to the output format. Unlike the `writer->string` function, `writer->escape` always uses the `escape` function, even when the `hybrid` option is enabled.

```
4166 self.escape = escape
```

Define `writer->code` as a function that will transform an input inlined code span `s` to the output format.

```
4167 function self.code(s)
4168     return {"\\markdownRendererCodeSpan{" ,self.escape(s),"}"}
4169 end
```

Define `writer->link` as a function that will transform an input hyperlink to the output format, where `lab` corresponds to the label, `src` to URI, and `tit` to the title of the link.

```
4170 function self.link(lab,src,tit)
4171     return {"\\markdownRendererLink{" ,lab,"}"} ,
4172           {"",self.escape(src),"}"} ,
4173           {"",self.uri(src),"}"} ,
4174           {"",self.string(tit or ""),"}"}
4175 end
```

Define `writer->table` as a function that will transform an input table to the output format, where `rows` is a sequence of columns and a column is a sequence of cell texts.

```
4176 function self.table(rows, caption)
4177     if not self.is_writing then return "" end
4178     local buffer = {"\\markdownRendererTable{" ,
4179                   caption or "", "}{" , #rows - 1, "}{" , #rows[1], "}"}
4180     local temp = rows[2] -- put alignments on the first row
4181     rows[2] = rows[1]
4182     rows[1] = temp
4183     for i, row in ipairs(rows) do
4184         table.insert(buffer, "{")
4185         for _, column in ipairs(row) do
4186             if i > 1 then -- do not use braces for alignments
4187                 table.insert(buffer, "{")
4188             end
4189             table.insert(buffer, column)
4190             if i > 1 then
4191                 table.insert(buffer, "}")
4192             end
4193         end
4194         table.insert(buffer, "}")
4195     end
4196     return buffer
4197 end
```

Define `writer->image` as a function that will transform an input image to the output format, where `lab` corresponds to the label, `src` to the URL, and `tit` to the title of the image.

```

4198 function self.image(lab,src,tit)
4199     return {"\\markdownRendererImage{" ,lab,"} ",
4200             "{" ,self.string(src),"} ",
4201             "{" ,self.uri(src),"} ",
4202             "{" ,self.string(tit or ""),"} "}
4203 end

```

The `languages_json` table maps programming language filename extensions to fence infostrings. All `options.contentBlocksLanguageMap` files located by the KPathSea library are loaded into a chain of tables. `languages_json` corresponds to the first table and is chained with the rest via Lua metatables.

```

4204 local languages_json = (function()
4205     local ran_ok, kpse = pcall(require, "kpse")
4206     if ran_ok then
4207         kpse.set_program_name("luatex")

```

If the KPathSea library is unavailable, perhaps because we are using LuaMetaTeX, we will only locate the `options.contentBlocksLanguageMap` in the current working directory:

```

4208     else
4209         kpse = {lookup=function(filename, options) return filename end}
4210     end
4211     local base, prev, curr
4212     for _, filename in ipairs{kpse.lookup(options.contentBlocksLanguageMap,
4213                                         { all=true })} do
4214         local file = io.open(filename, "r")
4215         if not file then goto continue end
4216         json = file:read("*all"):gsub('("[^\\n]-"):','[%1]=')
4217         curr = (function()
4218             local _ENV={ json=json, load=load } -- run in sandbox
4219             return load("return "..json)()
4220         end)()
4221         if type(curr) == "table" then
4222             if base == nil then
4223                 base = curr
4224             else
4225                 setmetatable(prev, { __index = curr })
4226             end
4227             prev = curr
4228         end
4229         ::continue::
4230     end
4231     return base or {}
4232 end)()

```

Define `writer->contentblock` as a function that will transform an input `iA Writer` content block to the output format, where `src` corresponds to the URI prefix, `suf` to the URI extension, `type` to the type of the content block (`localfile` or `onlineimage`), and `tit` to the title of the content block.

```

4233 function self.contentblock(src,suf,type,tit)
4234   if not self.is_writing then return "" end
4235   src = src..".."..suf
4236   suf = suf:lower()
4237   if type == "onlineimage" then
4238     return {"\\markdownRendererContentBlockOnlineImage{" ,suf,"} ",
4239           "{" ,self.string(src),"} ",
4240           "{" ,self.uri(src),"} ",
4241           "{" ,self.string(tit or ""),"}"}
4242   elseif languages_json[suf] then
4243     return {"\\markdownRendererContentBlockCode{" ,suf,"} ",
4244           "{" ,self.string(languages_json[suf]),"} ",
4245           "{" ,self.string(src),"} ",
4246           "{" ,self.uri(src),"} ",
4247           "{" ,self.string(tit or ""),"}"}
4248   else
4249     return {"\\markdownRendererContentBlock{" ,suf,"} ",
4250           "{" ,self.string(src),"} ",
4251           "{" ,self.uri(src),"} ",
4252           "{" ,self.string(tit or ""),"}"}
4253   end
4254 end

```

Define `writer->bulletlist` as a function that will transform an input bulleted list to the output format, where `items` is an array of the list items and `tight` specifies, whether the list is tight or not.

```

4255 local function ulitem(s)
4256   return {"\\markdownRendererUlItem " ,s,
4257         "\\markdownRendererUlItemEnd "}
4258 end
4259
4260 function self.bulletlist(items,tight)
4261   if not self.is_writing then return "" end
4262   local buffer = {}
4263   for _,item in ipairs(items) do
4264     buffer[#buffer + 1] = ulitem(item)
4265   end
4266   local contents = util.intersperse(buffer,"\\n")
4267   if tight and options.tightLists then
4268     return {"\\markdownRendererUlBeginTight\\n" ,contents,
4269           "\\n\\markdownRendererUlEndTight "}
4270   else
4271     return {"\\markdownRendererUlBegin\\n" ,contents,

```

```

4272         "\n\\markdownRendererUlEnd "}
4273     end
4274 end

```

Define `writer->olist` as a function that will transform an input ordered list to the output format, where `items` is an array of the list items and `tight` specifies, whether the list is tight or not. If the optional parameter `startnum` is present, it should be used as the number of the first list item.

```

4275 local function olistem(s,num)
4276     if num ~= nil then
4277         return {"\\markdownRendererOlistemWithNumber{" ,num,"} ",s,
4278             "\\markdownRendererOlistemEnd "}
4279     else
4280         return {"\\markdownRendererOlistem " ,s,
4281             "\\markdownRendererOlistemEnd "}
4282     end
4283 end
4284
4285 function self.orderedlist(items,tight,startnum)
4286     if not self.is_writing then return "" end
4287     local buffer = {}
4288     local num = startnum
4289     for _,item in ipairs(items) do
4290         buffer[#buffer + 1] = olistem(item,num)
4291         if num ~= nil then
4292             num = num + 1
4293         end
4294     end
4295     local contents = util.intersperse(buffer,"\n")
4296     if tight and options.tightLists then
4297         return {"\\markdownRendererOlistBeginTight\n",contents,
4298             "\n\\markdownRendererOlistEndTight "}
4299     else
4300         return {"\\markdownRendererOlistBegin\n",contents,
4301             "\n\\markdownRendererOlistEnd "}
4302     end
4303 end

```

Define `writer->inline_html_comment` as a function that will transform the contents of an inline HTML comment, to the output format, where `contents` are the contents of the HTML comment.

```

4304 function self.inline_html_comment(contents)
4305     return {"\\markdownRendererInlineHtmlComment{" ,contents,"}"}
4306 end

```

Define `writer->block_html_comment` as a function that will transform the contents of a block HTML comment, to the output format, where `contents` are the contents of the HTML comment.

```

4307 function self.block_html_comment(contents)
4308     if not self.is_writing then return "" end
4309     return {"\\markdownRendererBlockHtmlCommentBegin\n",contents,
4310           "\n\\markdownRendererBlockHtmlCommentEnd "}
4311 end

```

Define `writer->inline_html_tag` as a function that will transform the contents of an opening, closing, or empty inline HTML tag to the output format, where `contents` are the contents of the HTML tag.

```

4312 function self.inline_html_tag(contents)
4313     return {"\\markdownRendererInlineHtmlTag{" ,self.string(contents),"}"}
4314 end

```

Define `writer->block_html_element` as a function that will transform the contents of a block HTML element to the output format, where `s` are the contents of the HTML element.

```

4315 function self.block_html_element(s)
4316     if not self.is_writing then return "" end
4317     local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
4318     return {"\\markdownRendererInputBlockHtmlElement{" ,name,""}
4319 end

```

Define `writer->definitionlist` as a function that will transform an input definition list to the output format, where `items` is an array of tables, each of the form `{ term = t, definitions = defs }`, where `t` is a term and `defs` is an array of definitions. `tight` specifies, whether the list is tight or not.

```

4320 local function dlitem(term, defs)
4321     local retVal = {"\\markdownRendererDlItem{" ,term,""}
4322     for _, def in ipairs(defs) do
4323         retVal[#retVal+1] = {"\\markdownRendererDlDefinitionBegin ",def,
4324                             "\\markdownRendererDlDefinitionEnd "}
4325     end
4326     retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
4327     return retVal
4328 end
4329
4330 function self.definitionlist(items,tight)
4331     if not self.is_writing then return "" end
4332     local buffer = {}
4333     for _,item in ipairs(items) do
4334         buffer[#buffer + 1] = dlitem(item.term, item.definitions)
4335     end
4336     if tight and options.tightLists then
4337         return {"\\markdownRendererDlBeginTight\n", buffer,
4338                 "\n\\markdownRendererDlEndTight"}
4339     else
4340         return {"\\markdownRendererDlBegin\n", buffer,

```

```

4341         "\n\\markdownRendererDlEnd"}
4342     end
4343 end

```

Define `writer->emphasis` as a function that will transform an emphasized span `s` of input text to the output format.

```

4344 function self.emphasis(s)
4345     return {"\\markdownRendererEmphasis{" ,s,"}"}
4346 end

```

Define `writer->checkbox` as a function that will transform a number `f` to the output format.

```

4347 function self.checkbox(f)
4348     if f == 1.0 then
4349         return "☒ "
4350     elseif f == 0.0 then
4351         return "☐ "
4352     else
4353         return "☐ "
4354     end
4355 end

```

Define `writer->strong` as a function that will transform a strongly emphasized span `s` of input text to the output format.

```

4356 function self.strong(s)
4357     return {"\\markdownRendererStrongEmphasis{" ,s,"}"}
4358 end

```

Define `writer->blockquote` as a function that will transform an input block quote `s` to the output format.

```

4359 function self.blockquote(s)
4360     if #util.ropetostring(s) == 0 then return "" end
4361     return {"\\markdownRendererBlockQuoteBegin\n",s,
4362         "\n\\markdownRendererBlockQuoteEnd "}
4363 end

```

Define `writer->verbatim` as a function that will transform an input code block `s` to the output format.

```

4364 function self.verbatim(s)
4365     if not self.is_writing then return "" end
4366     s = string.gsub(s, ' [\r\n%s]*$', '')
4367     local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
4368     return {"\\markdownRendererInputVerbatim{" ,name,"}"}
4369 end

```

Define `writer->codeFence` as a function that will transform an input fenced code block `s` with the infostring `i` to the output format.

```

4370 function self.fencedCode(i, s)
4371     if not self.is_writing then return "" end

```

```

4372     s = string.gsub(s, '[\r\n%$]*$', '')
4373     local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
4374     return {"\\markdownRendererInputFencedCode{" ,name,"}{" ,i,"}"}
4375 end

```

Define `writer->document` as a function that will transform a document `d` to the output format.

```

4376 function self.document(d)
4377     local active_attributes = self.active_attributes
4378     local buf = {"\\markdownRendererDocumentBegin\n", d}
4379
4380     -- pop attributes for sections that have ended
4381     if options.headerAttributes and self.is_writing then
4382         while #active_attributes > 0 do
4383             local attributes = active_attributes[#active_attributes]
4384             if #attributes > 0 then
4385                 table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4386             end
4387             table.remove(active_attributes, #active_attributes)
4388         end
4389     end
4390
4391     table.insert(buf, "\\markdownRendererDocumentEnd")
4392
4393     return buf
4394 end

```

Define `writer->jekyllData` as a function that will transform an input YAML table `d` to the output format. The table is the value for the key `p` in the parent table; if `p` is nil, then the table has no parent. All scalar keys and values encountered in the table will be cast to a string following YAML serialization rules. String values will also be transformed using the function `t`.

```

4395 function self.jekyllData(d, t, p)
4396     if not self.is_writing then return "" end
4397
4398     local buf = {}
4399
4400     local keys = {}
4401     for k, _ in pairs(d) do
4402         table.insert(keys, k)
4403     end
4404     table.sort(keys)
4405
4406     if not p then
4407         table.insert(buf, "\\markdownRendererJekyllDataBegin")
4408     end
4409

```

```

4410     if #d > 0 then
4411         table.insert(buf, "\\markdownRendererJekyllDataSequenceBegin{")
4412         table.insert(buf, self.uri(p or "null"))
4413         table.insert(buf, "}{"")
4414         table.insert(buf, #keys)
4415         table.insert(buf, "}")
4416     else
4417         table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
4418         table.insert(buf, self.uri(p or "null"))
4419         table.insert(buf, "}{"")
4420         table.insert(buf, #keys)
4421         table.insert(buf, "}")
4422     end
4423
4424     for _, k in ipairs(keys) do
4425         local v = d[k]
4426         local typ = type(v)
4427         k = tostring(k or "null")
4428         if typ == "table" and next(v) ~= nil then
4429             table.insert(
4430                 buf,
4431                 self.jekyllData(v, t, k)
4432             )
4433         else
4434             k = self.uri(k)
4435             v = tostring(v)
4436             if typ == "boolean" then
4437                 table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
4438                 table.insert(buf, k)
4439                 table.insert(buf, "}{"")
4440                 table.insert(buf, v)
4441                 table.insert(buf, "}")
4442             elseif typ == "number" then
4443                 table.insert(buf, "\\markdownRendererJekyllDataNumber{")
4444                 table.insert(buf, k)
4445                 table.insert(buf, "}{"")
4446                 table.insert(buf, v)
4447                 table.insert(buf, "}")
4448             elseif typ == "string" then
4449                 table.insert(buf, "\\markdownRendererJekyllDataString{")
4450                 table.insert(buf, k)
4451                 table.insert(buf, "}{"")
4452                 table.insert(buf, t(v))
4453                 table.insert(buf, "}")
4454             elseif typ == "table" then
4455                 table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
4456                 table.insert(buf, k)

```



```

4457         table.insert(buf, "}")
4458     else
4459         error(format("Unexpected type %s for value of " ..
4460             "YAML key %s", typ, k))
4461     end
4462 end
4463 end
4464
4465 if #d > 0 then
4466     table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
4467 else
4468     table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
4469 end
4470
4471 if not p then
4472     table.insert(buf, "\\markdownRendererJekyllDataEnd")
4473 end
4474
4475 return buf
4476 end

```

Define `writer->active_attributes` as a stack of attributes of the headings that are currently active. The `writer->active_headings` member variable is mutable.

```

4477 self.active_attributes = {}

```

Define `writer->heading` as a function that will transform an input heading `s` at level `level` with identifiers `identifiers` to the output format.

```

4478 function self.heading(s, level, attributes)
4479     attributes = attributes or {}
4480     for i = 1, #attributes do
4481         attributes[attributes[i]] = true
4482     end
4483
4484     local active_attributes = self.active_attributes
4485     local slice_begin_type = self.slice_begin:sub(1, 1)
4486     local slice_begin_identifier = self.slice_begin:sub(2) or ""
4487     local slice_end_type = self.slice_end:sub(1, 1)
4488     local slice_end_identifier = self.slice_end:sub(2) or ""
4489
4490     local buf = {}
4491
4492     -- push empty attributes for implied sections
4493     while #active_attributes < level-1 do
4494         table.insert(active_attributes, {})
4495     end
4496
4497     -- pop attributes for sections that have ended
4498     while #active_attributes >= level do

```

```

4499     local active_identifiers = active_attributes[#active_attributes]
4500     -- tear down all active attributes at slice end
4501     if active_identifiers["#" .. slice_end_identifier] ~= nil
4502         and slice_end_type == "$" then
4503         for header_level = #active_attributes, 1, -1 do
4504             if options.headerAttributes and #active_attributes[header_level] > 0 then
4505                 table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4506             end
4507         end
4508         self.is_writing = false
4509     end
4510     table.remove(active_attributes, #active_attributes)
4511     if self.is_writing and options.headerAttributes and #active_identifiers > 0 then
4512         table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4513     end
4514     -- apply all active attributes at slice beginning
4515     if active_identifiers["#" .. slice_begin_identifier] ~= nil
4516         and slice_begin_type == "$" then
4517         for header_level = 1, #active_attributes do
4518             if options.headerAttributes and #active_attributes[header_level] > 0 then
4519                 table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
4520             end
4521         end
4522         self.is_writing = true
4523     end
4524 end
4525
4526 -- tear down all active attributes at slice end
4527 if attributes["#" .. slice_end_identifier] ~= nil
4528     and slice_end_type == "^" then
4529     for header_level = #active_attributes, 1, -1 do
4530         if options.headerAttributes and #active_attributes[header_level] > 0 then
4531             table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4532         end
4533     end
4534     self.is_writing = false
4535 end
4536
4537 -- push attributes for the new section
4538 table.insert(active_attributes, attributes)
4539 if self.is_writing and options.headerAttributes and #attributes > 0 then
4540     table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
4541 end
4542
4543 -- apply all active attributes at slice beginning
4544 if attributes["#" .. slice_begin_identifier] ~= nil
4545     and slice_begin_type == "^" then

```

```

4546     for header_level = 1, #active_attributes do
4547         if options.headerAttributes and #active_attributes[header_level] > 0 then
4548             table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
4549         end
4550     end
4551     self.is_writing = true
4552 end
4553
4554 if self.is_writing then
4555     table.sort(attributes)
4556     local key, value
4557     for i = 1, #attributes do
4558         if attributes[i]:sub(1, 1) == "#" then
4559             table.insert(buf, {"\\markdownRendererAttributeIdentifier{" ,
4560                 attributes[i]:sub(2), "}"})
4561         elseif attributes[i]:sub(1, 1) == "." then
4562             table.insert(buf, {"\\markdownRendererAttributeClassName{" ,
4563                 attributes[i]:sub(2), "}"})
4564         else
4565             key, value = attributes[i]:match("(%w+)=(%w+)")
4566             table.insert(buf, {"\\markdownRendererAttributeKeyValue{" ,
4567                 key, "}" , value, "}"})
4568         end
4569     end
4570 end
4571
4572 local cmd
4573 level = level + options.shiftHeadings
4574 if level <= 1 then
4575     cmd = "\\markdownRendererHeadingOne"
4576 elseif level == 2 then
4577     cmd = "\\markdownRendererHeadingTwo"
4578 elseif level == 3 then
4579     cmd = "\\markdownRendererHeadingThree"
4580 elseif level == 4 then
4581     cmd = "\\markdownRendererHeadingFour"
4582 elseif level == 5 then
4583     cmd = "\\markdownRendererHeadingFive"
4584 elseif level >= 6 then
4585     cmd = "\\markdownRendererHeadingSix"
4586 else
4587     cmd = ""
4588 end
4589 if self.is_writing then
4590     table.insert(buf, {cmd, "{", s, "}"})
4591 end
4592

```

```

4593     return buf
4594 end

```

Define `writer->note` as a function that will transform an input footnote `s` to the output format.

```

4595 function self.note(s)
4596     return {"\\markdownRendererFootnote{",s,""}
4597 end

```

Define `writer->citations` as a function that will transform an input array of citations `cites` to the output format. If `text_cites` is enabled, the citations should be rendered in-text, when applicable. The `cites` array contains tables with the following keys and values:

- `suppress_author` – If the value of the key is true, then the author of the work should be omitted in the citation, when applicable.
- `prenote` – The value of the key is either `nil` or a rope that should be inserted before the citation.
- `postnote` – The value of the key is either `nil` or a rope that should be inserted after the citation.
- `name` – The value of this key is the citation name.

```

4598 function self.citations(text_cites, cites)
4599     local buffer = {"\\markdownRenderer", text_cites and "TextCite" or "Cite",
4600         "{", #cites, "}"}
4601     for _,cite in ipairs(cites) do
4602         buffer[#buffer+1] = {cite.suppress_author and "-" or "+", "{",
4603             cite.prenote or "", "}{" , cite.postnote or "", "}{" , cite.name, "}"}
4604     end
4605     return buffer
4606 end

```

Define `writer->get_state` as a function that returns the current state of the writer, where the state of a writer are its mutable member variables.

```

4607 function self.get_state()
4608     return {
4609         is_writing=self.is_writing,
4610         active_attributes={table.unpack(self.active_attributes)},
4611     }
4612 end

```

Define `writer->set_state` as a function that restores the input state `s` and returns the previous state of the writer.

```

4613 function self.set_state(s)
4614     local previous_state = self.get_state()

```

```

4615     for key, value in pairs(s) do
4616         self[key] = value
4617     end
4618     return previous_state
4619 end

```

Define `writer->defer_call` as a function that will encapsulate the input function `f`, so that `f` is called with the state of the writer at the time of calling `writer->defer_call`.

```

4620 function self.defer_call(f)
4621     local previous_state = self.get_state()
4622     return function(...)
4623         local state = self.set_state(previous_state)
4624         local return_value = f(...)
4625         self.set_state(state)
4626         return return_value
4627     end
4628 end
4629
4630 return self
4631 end

```

### 3.1.4 Parsers

The `parsers` hash table stores PEG patterns that are static and can be reused between different `reader` objects.

```

4632 local parsers = {}

```

#### 3.1.4.1 Basic Parsers

```

4633 parsers.percent = P("%")
4634 parsers.at = P("@")
4635 parsers.comma = P(",")
4636 parsers.asterisk = P("*")
4637 parsers.dash = P("-")
4638 parsers.plus = P("+")
4639 parsers.underscore = P("_")
4640 parsers.period = P(".")
4641 parsers.hash = P("#")
4642 parsers.ampersand = P("&")
4643 parsers.backtick = P("`")
4644 parsers.less = P("<")
4645 parsers.more = P(">")
4646 parsers.space = P(" ")
4647 parsers.squote = P("'")
4648 parsers.dquote = P('"')
4649 parsers.lparent = P("(")

```

```

4650 parsers.rparent           = P("(")
4651 parsers.lbracket          = P("[")
4652 parsers.rbracket          = P("]")
4653 parsers.lbrace             = P("{")
4654 parsers.rbrace             = P("}")
4655 parsers.circumflex         = P("^")
4656 parsers.slash              = P("/")
4657 parsers.equal               = P("=")
4658 parsers.colon               = P(":")
4659 parsers.semicolon          = P(";")
4660 parsers.exclamation         = P("!")
4661 parsers.pipe                = P("|")
4662 parsers.tilde               = P("~")
4663 parsers.backslash          = P("\\")
4664 parsers.tab                 = P("\t")
4665 parsers.newline            = P("\n")
4666 parsers.tightblocksep     = P("\001")
4667
4668 parsers.digit                = R("09")
4669 parsers.hexdigit            = R("09", "af", "AF")
4670 parsers.letter              = R("AZ", "az")
4671 parsers.alphanumeric        = R("AZ", "az", "09")
4672 parsers.keyword             = parsers.letter
4673                             * parsers.alphanumeric^0
4674 parsers.citation_chars      = parsers.alphanumeric
4675                             + S("#$%&-+<>~/_")
4676 parsers.internal_punctuation = S(";, .?")
4677
4678 parsers.doubleasterisks     = P("**")
4679 parsers.doubleunderscores   = P("__")
4680 parsers.fourspace           = P("    ")
4681
4682 parsers.any                  = P(1)
4683 parsers.fail                 = parsers.any - 1
4684
4685 parsers.escapable           = S("\\`*_{}[]()+_!.<>#-~:~@;")
4686 parsers.anyescaped          = parsers.backslash / " " * parsers.escapable
4687                             + parsers.any
4688
4689 parsers.spacechar            = S("\t ")
4690 parsers.spacing              = S(" \n\r\t")
4691 parsers.nospacechar          = parsers.any - parsers.spacing
4692 parsers.optionalspace        = parsers.spacechar^0
4693
4694 parsers.specialchar          = S("*_`&[]<!\. @-~")
4695
4696 parsers.normalchar           = parsers.any - (parsers.specialchar

```

```

4697                                     + parsers.spacing
4698                                     + parsers.tightblocksep)
4699 parsers.eof                          = -parsers.any
4700 parsers.nonindentospace               = parsers.space-3 * - parsers.spacechar
4701 parsers.indent                        = parsers.space-3 * parsers.tab
4702                                     + parsers.fourspace / ""
4703 parsers.linechar                       = P(1 - parsers.newline)
4704
4705 parsers.blankline                     = parsers.optionalspace
4706                                     * parsers.newline / "\n"
4707 parsers.blanklines                     = parsers.blankline0
4708 parsers.skipblanklines                 = (parsers.optionalspace * parsers.newline)0
4709 parsers.indentedline                  = parsers.indent / ""
4710                                     * C(parsers.linechar1 * parsers.newline-
4711                                     1)
4711 parsers.optionallyindentedline        = parsers.indent-1 / ""
4712                                     * C(parsers.linechar1 * parsers.newline-
4713                                     1)
4713 parsers.sp                             = parsers.spacing0
4714 parsers.spnl                           = parsers.optionalspace
4715                                     * (parsers.newline * parsers.optionalspace)-
4716                                     1
4716 parsers.line                           = parsers.linechar0 * parsers.newline
4717 parsers.nonemptyline                   = parsers.line - parsers.blankline

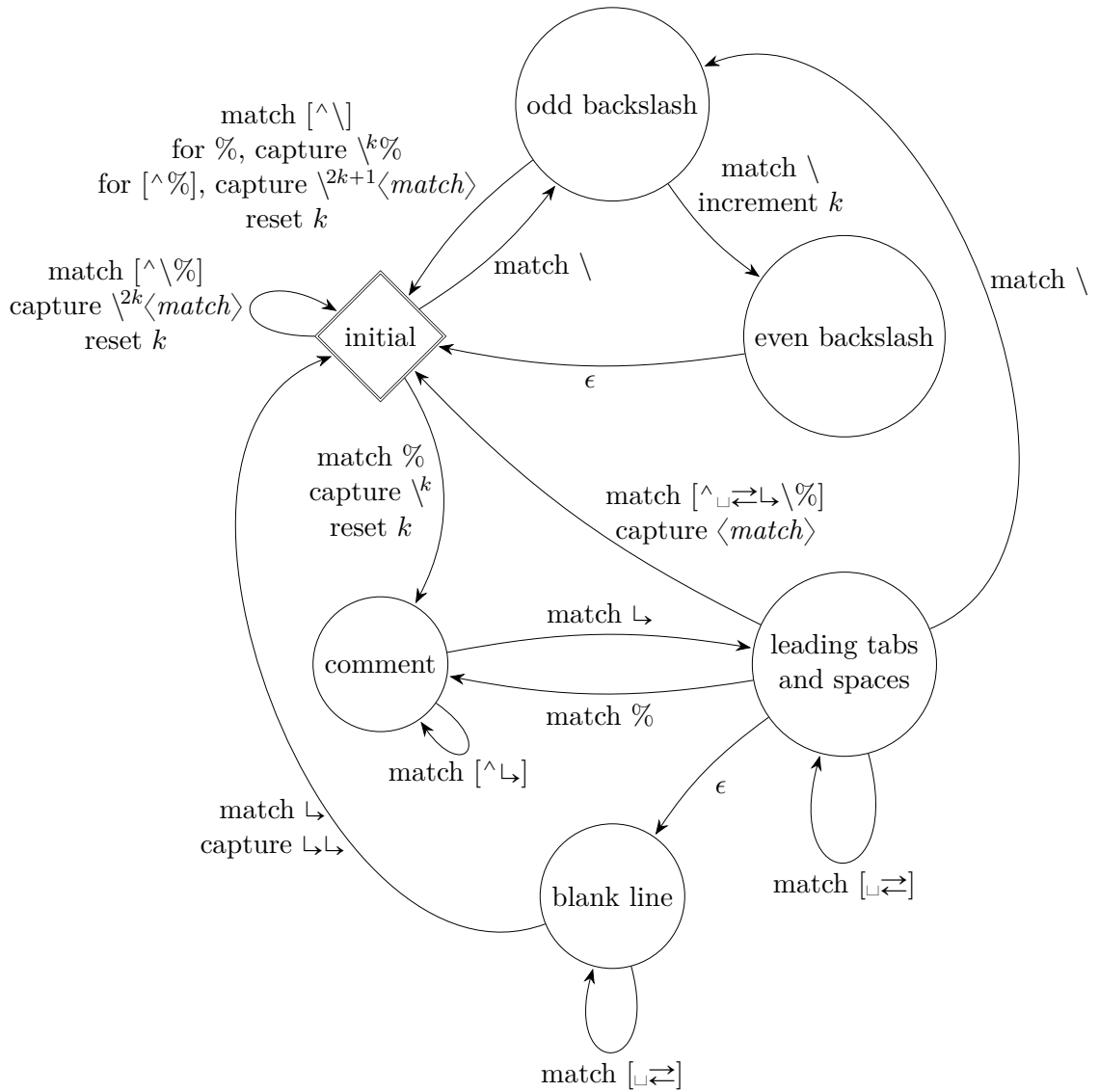
```

The `parsers.commented_line1` parser recognizes the regular language of T<sub>E</sub>X comments, see an equivalent finite automaton in Figure 6.

```

4718 parsers.commented_line_letter          = parsers.linechar
4719                                     + parsers.newline
4720                                     - parsers.backslash
4721                                     - parsers.percent
4722 parsers.commented_line                 = Cg(Cc(""), "backslashes")
4723                                     * ((#(parsers.commented_line_letter
4724                                     - parsers.newline)
4725                                     * Cb("backslashes")
4726                                     * Cs(parsers.commented_line_letter
4727                                     - parsers.newline)1 -- initial
4728                                     * Cg(Cc(""), "backslashes"))
4729                                     + #(parsers.backslash * parsers.backslash)
4730                                     * Cg((parsers.backslash -- even backslash
4731                                     * parsers.backslash)1, "backslashes")
4732                                     + (parsers.backslash
4733                                     * (#parsers.percent
4734                                     * Cb("backslashes")
4735                                     / function(backslashes)
4736                                     return string.rep("\\", #backslashes / 2)
4737                                     end

```



**Figure 6: A pushdown automaton that recognizes TeX comments**



```

4738         * C(parsers.percent)
4739         + #parsers.commented_line_letter
4740         * Cb("backslashes")
4741         * Cc("\\")
4742         * C(parsers.commented_line_letter))
4743         * Cg(Cc(""), "backslashes"))^0
4744     * (#parsers.percent
4745     * Cb("backslashes")
4746     / function(backslashes)
4747     return string.rep("\\", #backslashes / 2)
4748     end
4749     * ((parsers.percent -- comment
4750     * parsers.line
4751     * #parsers.blankline) -- blank line
4752     / "\n"
4753     + parsers.percent -- comment
4754     * parsers.line
4755     * parsers.optionalspace) -- leading tabs and spaces
4756     + #(parsers.newline)
4757     * Cb("backslashes")
4758     * C(parsers.newline))
4759
4760 parsers.chunk = parsers.line * (parsers.optionallyindentedline
4761                             - parsers.blankline)^0
4762
4763 parsers.css_identifier_char = R("AZ", "az", "09") + S("-_")
4764 parsers.css_identifier = (parsers.hash + parsers.period)
4765 * (((parsers.css_identifier_char
4766     - parsers.dash - parsers.digit)
4767     * parsers.css_identifier_char^1)
4768     + (parsers.dash
4769     * (parsers.css_identifier_char
4770     - parsers.digit)
4771     * parsers.css_identifier_char^0))
4772 parsers.attribute_key_char = parsers.any - parsers.space
4773 - parsers.squote - parsers.dquote
4774 - parsers.more - parsers.slash
4775 - parsers.equal
4776 parsers.attribute_value_char = parsers.any - parsers.space
4777 - parsers.dquote - parsers.more
4778
4779 -- block followed by 0 or more optionally
4780 -- indented blocks with first line indented.
4781 parsers.indented_blocks = function(bl)
4782     return Cs( bl
4783         * (parsers.blankline^1 * parsers.indent * -parsers.blankline * bl)^0
4784         * (parsers.blankline^1 + parsers.eof) )

```

```
4785 end
```

### 3.1.4.2 Parsers Used for Markdown Lists

```
4786 parsers.bulletchar = C(parsers.plus + parsers.asterisk + parsers.dash)
4787
4788 parsers.bullet = ( parsers.bulletchar * #parsers.spacing
4789                   * (parsers.tab + parsers.space^-3)
4790                   + parsers.space * parsers.bulletchar * #parsers.spacing
4791                   * (parsers.tab + parsers.space^-2)
4792                   + parsers.space * parsers.space * parsers.bulletchar
4793                   * #parsers.spacing
4794                   * (parsers.tab + parsers.space^-1)
4795                   + parsers.space * parsers.space * parsers.space
4796                   * parsers.bulletchar * #parsers.spacing
4797                   )
4798
4799 local function tickbox(interior)
4800   return parsers.optionalspace * parsers.lbracket
4801         * interior * parsers.rbracket * parsers.spacechar^1
4802 end
4803
4804 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
4805 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
4806 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
4807
```

### 3.1.4.3 Parsers Used for Markdown Code Spans

```
4808 parsers.openticks = Cg(parsers.backtick^1, "ticks")
4809
4810 local function captures_equal_length(s,i,a,b)
4811   return #a == #b and i
4812 end
4813
4814 parsers.closeticks = parsers.space^-1
4815                   * Cmt(C(parsers.backtick^1)
4816                       * Cb("ticks"), captures_equal_length)
4817
4818 parsers.intickschar = (parsers.any - S(" \n\r`"))
4819                   + (parsers.newline * -parsers.blankline)
4820                   + (parsers.space - parsers.closeticks)
4821                   + (parsers.backtick^1 - parsers.closeticks)
4822
4823 parsers.inticks = parsers.openticks * parsers.space^-1
4824                 * C(parsers.intickschar^0) * parsers.closeticks
```

### 3.1.4.4 Parsers Used for Fenced Code Blocks

```
4825 local function captures_geq_length(s,i,a,b)
4826   return #a >= #b and i
4827 end
4828
4829 parsers.infostring      = (parsers.linechar - (parsers.backtick
4830   + parsers.space^1 * (parsers.newline + parsers.eof)))^0
4831
4832 local fenceindent
4833 parsers.fencehead      = function(char)
4834   return               C(parsers.nonindentospace) / function(s) fenceindent = #s end
4835   * Cg(char^3, "fencelength")
4836   * parsers.optionalspace * C(parsers.infostring)
4837   * parsers.optionalspace * (parsers.newline + parsers.eof)
4838 end
4839
4840 parsers.fencetail      = function(char)
4841   return               parsers.nonindentospace
4842   * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
4843   * parsers.optionalspace * (parsers.newline + parsers.eof)
4844   + parsers.eof
4845 end
4846
4847 parsers.fencedline     = function(char)
4848   return               C(parsers.line - parsers.fencetail(char))
4849   / function(s)
4850     i = 1
4851     remaining = fenceindent
4852     while true do
4853       c = s:sub(i, i)
4854       if c == " " and remaining > 0 then
4855         remaining = remaining - 1
4856         i = i + 1
4857       elseif c == "\t" and remaining > 3 then
4858         remaining = remaining - 4
4859         i = i + 1
4860       else
4861         break
4862       end
4863     end
4864     return s:sub(i)
4865   end
4866 end
```

### 3.1.4.5 Parsers Used for Markdown Tags and Links

```
4867 parsers.leader        = parsers.space^-3
```

```

4868
4869 -- content in balanced brackets, parentheses, or quotes:
4870 parsers.bracketed = P{ parsers.lbracket
4871                      * ((parsers.anyescaped - (parsers.lbracket
4872                                                                + parsers.rbracket
4873                                                                + parsers.blankline^2)
4874                      ) + V(1))^0
4875                      * parsers.rbracket }
4876
4877 parsers.inparens   = P{ parsers.lparent
4878                      * ((parsers.anyescaped - (parsers.lparent
4879                                                                + parsers.rparent
4880                                                                + parsers.blankline^2)
4881                      ) + V(1))^0
4882                      * parsers.rparent }
4883
4884 parsers.squoted    = P{ parsers.squote * parsers.alphanumeric
4885                      * ((parsers.anyescaped - (parsers.squote
4886                                                                + parsers.blankline^2)
4887                      ) + V(1))^0
4888                      * parsers.squote }
4889
4890 parsers.dquoted    = P{ parsers.dquote * parsers.alphanumeric
4891                      * ((parsers.anyescaped - (parsers.dquote
4892                                                                + parsers.blankline^2)
4893                      ) + V(1))^0
4894                      * parsers.dquote }
4895
4896 -- bracketed tag for markdown links, allowing nested brackets:
4897 parsers.tag        = parsers.lbracket
4898                      * Cs((parsers.alphanumeric^1
4899                          + parsers.bracketed
4900                          + parsers.inticks
4901                          + (parsers.anyescaped
4902                            - (parsers.rbracket + parsers.blankline^2)))^0)
4903                      * parsers.rbracket
4904
4905 -- url for markdown links, allowing nested brackets:
4906 parsers.url        = parsers.less * Cs((parsers.anyescaped
4907                                         - parsers.more)^0)
4908                                         * parsers.more
4909                                         + Cs((parsers.inparens + (parsers.anyescaped
4910                                             - parsers.spacing
4911                                             - parsers.rparent))^1)
4912
4913 -- quoted text, possibly with nested quotes:
4914 parsers.title_s    = parsers.squote * Cs(((parsers.anyescaped-parsers.squote)

```

```

4915             + parsers.squoted)^0)
4916         * parsers.squote
4917
4918 parsers.title_d   = parsers.dquote * Cs(((parsers.anyescaped-parsers.dquote)
4919             + parsers.dquoted)^0)
4920         * parsers.dquote
4921
4922 parsers.title_p   = parsers.lparent
4923         * Cs((parsers.inparens + (parsers.anyescaped-parsers.rparent))^0)
4924         * parsers.rparent
4925
4926 parsers.title     = parsers.title_d + parsers.title_s + parsers.title_p
4927
4928 parsers.optionaltitle
4929         = parsers.spnl * parsers.title * parsers.spacechar^0
4930         + Cc("")

```

### 3.1.4.6 Parsers Used for iA Writer Content Blocks

```

4931 parsers.contentblock_tail
4932         = parsers.optionaltitle
4933         * (parsers.newline + parsers.eof)
4934
4935 -- case insensitive online image suffix:
4936 parsers.onlineimagesuffix
4937         = (function(...)
4938             local parser = nil
4939             for _,suffix in ipairs({...}) do
4940                 local pattern=nil
4941                 for i=1,#suffix do
4942                     local char=suffix:sub(i,i)
4943                     char = S(char:lower()..char:upper())
4944                     if pattern == nil then
4945                         pattern = char
4946                     else
4947                         pattern = pattern * char
4948                     end
4949                 end
4950                 if parser == nil then
4951                     parser = pattern
4952                 else
4953                     parser = parser + pattern
4954                 end
4955             end
4956             return parser
4957         end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
4958

```

```

4959 -- online image url for iA Writer content blocks with mandatory suffix,
4960 -- allowing nested brackets:
4961 parsers.onlineimageurl
4962     = (parsers.less
4963        * Cs((parsers.anyescaped
4964             - parsers.more
4965             - #(parsers.period
4966                * parsers.onlineimagesuffix
4967                * parsers.more
4968                * parsers.contentblock_tail))^0)
4969        * parsers.period
4970        * Cs(parsers.onlineimagesuffix)
4971        * parsers.more
4972        + (Cs((parsers.inparens
4973             + (parsers.anyescaped
4974                - parsers.spacing
4975                - parsers.rparent
4976                - #(parsers.period
4977                   * parsers.onlineimagesuffix
4978                   * parsers.contentblock_tail)))^0)
4979        * parsers.period
4980        * Cs(parsers.onlineimagesuffix))
4981        ) * Cc("onlineimage")
4982
4983 -- filename for iA Writer content blocks with mandatory suffix:
4984 parsers.localfilepath
4985     = parsers.slash
4986     * Cs((parsers.anyescaped
4987          - parsers.tab
4988          - parsers.newline
4989          - #(parsers.period
4990             * parsers.alphanumeric^1
4991             * parsers.contentblock_tail))^1)
4992     * parsers.period
4993     * Cs(parsers.alphanumeric^1)
4994     * Cc("localfile")

```

### 3.1.4.7 Parsers Used for Citations

```

4995 parsers.citation_name = Cs(parsers.dash^-1) * parsers.at
4996                       * Cs(parsers.citation_chars
4997                            * (((parsers.citation_chars + parsers.internal_punctuation
4998                               - parsers.comma - parsers.semicolon)
4999                               * -#((parsers.internal_punctuation - parsers.comma
5000                                  - parsers.semicolon))^0
5001                               * -(parsers.citation_chars + parsers.internal_punctuat.
5002                                  - parsers.comma - parsers.semicolon)))^0

```

```

5003             * parsers.citation_chars)^-1)
5004
5005 parsers.citation_body_prenote
5006     = Cs((parsers.alphanumeric^1
5007         + parsers.bracketed
5008         + parsers.inticks
5009         + (parsers.anyescaped
5010         - (parsers.rbracket + parsers.blankline^2))
5011         - (parsers.spnl * parsers.dash^-1 * parsers.at))^0)
5012
5013 parsers.citation_body_postnote
5014     = Cs((parsers.alphanumeric^1
5015         + parsers.bracketed
5016         + parsers.inticks
5017         + (parsers.anyescaped
5018         - (parsers.rbracket + parsers.semicolon
5019         + parsers.blankline^2))
5020         - (parsers.spnl * parsers.rbracket))^0)
5021
5022 parsers.citation_body_chunk
5023     = parsers.citation_body_prenote
5024     * parsers.spnl * parsers.citation_name
5025     * (parsers.internal_punctuation - parsers.semicolon)^-
5026     1
5027     * parsers.spnl * parsers.citation_body_postnote
5028
5029 parsers.citation_body
5030     = parsers.citation_body_chunk
5031     * (parsers.semicolon * parsers.spnl
5032     * parsers.citation_body_chunk)^0
5033
5034 parsers.citation_headless_body_postnote
5035     = Cs((parsers.alphanumeric^1
5036         + parsers.bracketed
5037         + parsers.inticks
5038         + (parsers.anyescaped
5039         - (parsers.rbracket + parsers.at
5040         + parsers.semicolon + parsers.blankline^2))
5041         - (parsers.spnl * parsers.rbracket))^0)
5042
5043 parsers.citation_headless_body
5044     = parsers.citation_headless_body_postnote
5045     * (parsers.sp * parsers.semicolon * parsers.spnl
5046     * parsers.citation_body_chunk)^0

```

### 3.1.4.8 Parsers Used for Footnotes

```

5046 local function strip_first_char(s)
5047   return s:sub(2)
5048 end
5049
5050 parsers.RawNoteRef = #(parsers.lbracket * parsers.circumflex)
5051                       * parsers.tag / strip_first_char

```

### 3.1.4.9 Parsers Used for Tables

```

5052 local function make_pipe_table_rectangular(rows)
5053   local num_columns = #rows[2]
5054   local rectangular_rows = {}
5055   for i = 1, #rows do
5056     local row = rows[i]
5057     local rectangular_row = {}
5058     for j = 1, num_columns do
5059       rectangular_row[j] = row[j] or ""
5060     end
5061     table.insert(rectangular_rows, rectangular_row)
5062   end
5063   return rectangular_rows
5064 end
5065
5066 local function pipe_table_row(allow_empty_first_column
5067                               , nonempty_column
5068                               , column_separator
5069                               , column)
5070   local row_beginning
5071   if allow_empty_first_column then
5072     row_beginning = -- empty first column
5073                   #(parsers.spacechar^4
5074                     * column_separator)
5075                   * parsers.optionalspace
5076                   * column
5077                   * parsers.optionalspace
5078                   -- non-empty first column
5079                   + parsers.nonindentSPACE
5080                   * nonempty_column^-1
5081                   * parsers.optionalspace
5082   else
5083     row_beginning = parsers.nonindentSPACE
5084                   * nonempty_column^-1
5085                   * parsers.optionalspace
5086   end
5087
5088   return Ct(row_beginning
5089             * (-- single column with no leading pipes

```



```

5090         #(column_separator
5091         * parsers.optionalspace
5092         * parsers.newline)
5093     * column_separator
5094     * parsers.optionalspace
5095     -- single column with leading pipes or
5096     -- more than a single column
5097     + (column_separator
5098     * parsers.optionalspace
5099     * column
5100     * parsers.optionalspace)^1
5101     * (column_separator
5102     * parsers.optionalspace)^-1))
5103 end
5104
5105 parsers.table_hline_separator = parsers.pipe + parsers.plus
5106 parsers.table_hline_column = (parsers.dash
5107     - #(parsers.dash
5108     * (parsers.spacechar
5109     + parsers.table_hline_separator
5110     + parsers.newline)))^1
5111     * (parsers.colon * Cc("r")
5112     + parsers.dash * Cc("d"))
5113     + parsers.colon
5114     * (parsers.dash
5115     - #(parsers.dash
5116     * (parsers.spacechar
5117     + parsers.table_hline_separator
5118     + parsers.newline)))^1
5119     * (parsers.colon * Cc("c")
5120     + parsers.dash * Cc("l"))
5121 parsers.table_hline = pipe_table_row(false
5122     , parsers.table_hline_column
5123     , parsers.table_hline_separator
5124     , parsers.table_hline_column)
5125 parsers.table_caption_beginning = parsers.skipblanklines
5126     * parsers.nonindentospace
5127     * (P("Table")^-1 * parsers.colon)
5128     * parsers.optionalspace

```

### 3.1.4.10 Parsers Used for HTML

```

5129 -- case-insensitive match (we assume s is lowercase). must be single byte encoding
5130 parsers.keyword_exact = function(s)
5131     local parser = P(0)
5132     for i=1,#s do
5133         local c = s:sub(i,i)

```

```

5134     local m = c .. upper(c)
5135     parser = parser * S(m)
5136 end
5137 return parser
5138 end
5139
5140 parsers.block_keyword =
5141     parsers.keyword_exact("address") + parsers.keyword_exact("blockquote") +
5142     parsers.keyword_exact("center") + parsers.keyword_exact("del") +
5143     parsers.keyword_exact("dir") + parsers.keyword_exact("div") +
5144     parsers.keyword_exact("p") + parsers.keyword_exact("pre") +
5145     parsers.keyword_exact("li") + parsers.keyword_exact("ol") +
5146     parsers.keyword_exact("ul") + parsers.keyword_exact("dl") +
5147     parsers.keyword_exact("dd") + parsers.keyword_exact("form") +
5148     parsers.keyword_exact("fieldset") + parsers.keyword_exact("isindex") +
5149     parsers.keyword_exact("ins") + parsers.keyword_exact("menu") +
5150     parsers.keyword_exact("noframes") + parsers.keyword_exact("frameset") +
5151     parsers.keyword_exact("h1") + parsers.keyword_exact("h2") +
5152     parsers.keyword_exact("h3") + parsers.keyword_exact("h4") +
5153     parsers.keyword_exact("h5") + parsers.keyword_exact("h6") +
5154     parsers.keyword_exact("hr") + parsers.keyword_exact("script") +
5155     parsers.keyword_exact("noscript") + parsers.keyword_exact("table") +
5156     parsers.keyword_exact("tbody") + parsers.keyword_exact("tfoot") +
5157     parsers.keyword_exact("thead") + parsers.keyword_exact("th") +
5158     parsers.keyword_exact("td") + parsers.keyword_exact("tr")
5159
5160 -- There is no reason to support bad html, so we expect quoted attributes
5161 parsers.htmlattributevalue
5162     = parsers.squote * (parsers.any - (parsers.blankline
5163     + parsers.squote))^0
5164     * parsers.squote
5165     + parsers.dquote * (parsers.any - (parsers.blankline
5166     + parsers.dquote))^0
5167     * parsers.dquote
5168
5169 parsers.htmlattribute = parsers.spacing^1
5170     * (parsers.alphanumeric + S("_-"))^1
5171     * parsers.sp * parsers.equal * parsers.sp
5172     * parsers.htmlattributevalue
5173
5174 parsers.htmlcomment = P("<!--")
5175     * parsers.optionalspace
5176     * Cs((parsers.any - parsers.optionalspace * P("-->"))^0)
5177     * parsers.optionalspace
5178     * P("-->")
5179
5180 parsers.htmlinstruction = P("<?") * (parsers.any - P(">"))^0 * P(">")

```

```

5181
5182 parsers.openelt_any = parsers.less * parsers.keyword * parsers.htmlattribute^0
5183         * parsers.sp * parsers.more
5184
5185 parsers.openelt_exact = function(s)
5186   return parsers.less * parsers.sp * parsers.keyword_exact(s)
5187         * parsers.htmlattribute^0 * parsers.sp * parsers.more
5188 end
5189
5190 parsers.openelt_block = parsers.sp * parsers.block_keyword
5191         * parsers.htmlattribute^0 * parsers.sp * parsers.more
5192
5193 parsers.closeelt_any = parsers.less * parsers.sp * parsers.slash
5194         * parsers.keyword * parsers.sp * parsers.more
5195
5196 parsers.closeelt_exact = function(s)
5197   return parsers.less * parsers.sp * parsers.slash * parsers.keyword_exact(s)
5198         * parsers.sp * parsers.more
5199 end
5200
5201 parsers.emptyelt_any = parsers.less * parsers.sp * parsers.keyword
5202         * parsers.htmlattribute^0 * parsers.sp * parsers.slash
5203         * parsers.more
5204
5205 parsers.emptyelt_block = parsers.less * parsers.sp * parsers.block_keyword
5206         * parsers.htmlattribute^0 * parsers.sp * parsers.slash
5207         * parsers.more
5208
5209 parsers.displaytext = (parsers.any - parsers.less)^1
5210
5211 -- return content between two matched HTML tags
5212 parsers.in_matched = function(s)
5213   return { parsers.openelt_exact(s)
5214         * (V(1) + parsers.displaytext
5215         + (parsers.less - parsers.closeelt_exact(s)))^0
5216         * parsers.closeelt_exact(s) }
5217 end
5218
5219 local function parse_matched_tags(s,pos)
5220   local t = string.lower(lpeg.match(C(parsers.keyword),s,pos))
5221   return lpeg.match(parsers.in_matched(t),s,pos-1)
5222 end
5223
5224 parsers.in_matched_block_tags = parsers.less
5225         * Cmt(#parsers.openelt_block, parse_matched_tags)
5226

```

### 3.1.4.11 Parsers Used for HTML Entities

```
5227 parsers.hexentity = parsers.ampersand * parsers.hash * S("Xx")
5228                   * C(parsers.hexdigit^1) * parsers.semicolon
5229 parsers.decentity  = parsers.ampersand * parsers.hash
5230                   * C(parsers.digit^1) * parsers.semicolon
5231 parsers.tagentity  = parsers.ampersand * C(parsers.alphanumeric^1)
5232                   * parsers.semicolon
```

### 3.1.4.12 Helpers for References

```
5233 -- parse a reference definition: [foo]: /bar "title"
5234 parsers.define_reference_parser = parsers.leader * parsers.tag * parsers.colon
5235                               * parsers.spacechar^0 * parsers.url
5236                               * parsers.optionaltitle * parsers.blankline^1
```

### 3.1.4.13 Inline Elements

```
5237 parsers.Inline      = V("Inline")
5238 parsers.IndentedInline = V("IndentedInline")
5239
5240 -- parse many p between starter and ender
5241 parsers.between = function(p, starter, ender)
5242   local ender2 = B(parsers.nonspacechar) * ender
5243   return (starter * #parsers.nonspacechar * Ct(p * (p - ender2)^0) * ender2)
5244 end
5245
5246 parsers.urlchar      = parsers.anyescaped - parsers.newline - parsers.more
```

### 3.1.4.14 Block Elements

```
5247 parsers.OnlineImageURL
5248                   = parsers.leader
5249                   * parsers.onlineimageurl
5250                   * parsers.optionaltitle
5251
5252 parsers.LocalFilePath
5253                   = parsers.leader
5254                   * parsers.localfilepath
5255                   * parsers.optionaltitle
5256
5257 parsers.TildeFencedCode
5258                   = parsers.fencehead(parsers.tilde)
5259                   * Cs(parsers.fencedline(parsers.tilde)^0)
5260                   * parsers.fencetail(parsers.tilde)
5261
5262 parsers.BacktickFencedCode
5263                   = parsers.fencehead(parsers.backtick)
5264                   * Cs(parsers.fencedline(parsers.backtick)^0)
```

```

5265             * parsers.fencetail(parsers.backtick)
5266
5267 parsers.JekyllFencedCode
5268             = parsers.fencehead(parsers.dash)
5269             * Cs(parsers.fencedline(parsers.dash)^0)
5270             * parsers.fencetail(parsers.dash)
5271
5272 parsers.lineof = function(c)
5273     return (parsers.leader * (P(c) * parsers.optionalspace)^3
5274           * (parsers.newline * parsers.blankline^1
5275             + parsers.newline^-1 * parsers.eof))
5276 end

```

### 3.1.4.15 Lists

```

5277 parsers.defstartchar = S("~:")
5278 parsers.defstart     = ( parsers.defstartchar * #parsers.spacing
5279                       * (parsers.tab + parsers.space^-
5280                          3)
5281                       + parsers.space * parsers.defstartchar * #parsers.spacing
5282                       * (parsers.tab + parsers.space^-2)
5283                       + parsers.space * parsers.space * parsers.defstartchar
5284                       * #parsers.spacing
5285                       * (parsers.tab + parsers.space^-1)
5286                       + parsers.space * parsers.space * parsers.space
5287                       * parsers.defstartchar * #parsers.spacing
5288                       )
5289 parsers.dlchunk = Cs(parsers.line * (parsers.indentedline - parsers.blankline)^0)

```

### 3.1.4.16 Headings

```

5290 parsers.heading_attribute = C(parsers.css_identifier)
5291                          + C((parsers.attribute_key_char
5292                             - parsers.rbrace)^1
5293                             * parsers.equal
5294                             * (parsers.attribute_value_char
5295                               - parsers.rbrace)^1)
5296 parsers.HeadingAttributes = parsers.lbrace
5297                          * parsers.heading_attribute
5298                          * (parsers.spacechar^1
5299                             * parsers.heading_attribute)^0
5300                          * parsers.rbrace
5301
5302 -- parse Atx heading start and return level
5303 parsers.HeadingStart = #parsers.hash * C(parsers.hash^-6)
5304                   * -parsers.hash / length
5305

```

```

5306 -- parse setext header ending and return level
5307 parsers.HeadingLevel = parsers.equal^1 * Cc(1) + parsers.dash^1 * Cc(2)
5308
5309 local function strip_atx_end(s)
5310   return s:gsub("#%s*\n$", "")
5311 end

```

### 3.1.5 Markdown Reader

This section documents the `reader` object, which implements the routines for parsing the markdown input. The object corresponds to the markdown reader object that was located in the `lunamark/reader/markdown.lua` file in the Lunamark Lua module.

Although not specified in the Lua interface (see Section 2.1), the `reader` object is exported, so that the curious user could easily tinker with the methods of the objects produced by the `reader.new` method described below. The user should be aware, however, that the implementation may change in a future revision.

The `reader.new` method creates and returns a new TeX reader object associated with the Lua interface options (see Section 2.1.2) `options` and with a writer object `writer`. When `options` are unspecified, it is assumed that an empty table was passed to the method.

The objects produced by the `reader.new` method expose instance methods and variables of their own. As a convention, I will refer to these *member*s as `reader->member`.

```

5312 M.reader = {}
5313 function M.reader.new(writer, options)
5314   local self = {}
5315   options = options or {}

```

Make the `options` table inherit from the `defaultOptions` table.

```

5316   setmetatable(options, { __index = function (_, key)
5317     return defaultOptions[key] end })

```

**3.1.5.1 Top-Level Helper Functions** Define `normalize_tag` as a function that normalizes a markdown reference tag by lowercasing it, and by collapsing any adjacent whitespace characters.

```

5318   local function normalize_tag(tag)
5319     return string.lower(
5320       gsub(util.ropetostring(tag), "[\n\r\t]+", " "))
5321   end

```

Define `iterlines` as a function that iterates over the lines of the input string `s`, transforms them using an input function `f`, and reassembles them into a new string, which it returns.

```

5322   local function iterlines(s, f)
5323     rope = lpeg.match(Ct((parsers.line / f)^1), s)

```

```

5324     return util.ropetostring(ropetostring)
5325 end

```

Define `expandtabs` either as an identity function, when the `preserveTabs` Lua interface option is enabled, or to a function that expands tabs into spaces otherwise.

```

5326 local expandtabs
5327 if options.preserveTabs then
5328     expandtabs = function(s) return s end
5329 else
5330     expandtabs = function(s)
5331         if s:find("\t") then
5332             return iterlines(s, util.expand_tabs_in_line)
5333         else
5334             return s
5335         end
5336     end
5337 end

```

The `larsers` (as in ‘`local \luamref{parsers}'' hash table stores \acro{peg} patterns`’, which impedes their reuse between different `reader` objects.

```

5338 local larsers = {}

```

### 3.1.5.2 Top-Level Parser Functions

```

5339 local function create_parser(name, grammar, toplevel)
5340     return function(str)

```

If the parser is top-level and the `stripIndent` Lua option is enabled, we will first expand tabs in the input string `str` into spaces and then we will count the minimum indent across all lines, skipping blank lines. Next, we will remove the minimum indent from all lines.

```

5341     if toplevel and options.stripIndent then
5342         local min_prefix_length, min_prefix = nil, ''
5343         str = iterlines(str, function(line)
5344             if lpeg.match(parsers.nonemptyline, line) == nil then
5345                 return line
5346             end
5347             line = util.expand_tabs_in_line(line)
5348             prefix = lpeg.match(C(parsers.optionalspace), line)
5349             local prefix_length = #prefix
5350             local is_shorter = min_prefix_length == nil
5351             is_shorter = is_shorter or prefix_length < min_prefix_length
5352             if is_shorter then
5353                 min_prefix_length, min_prefix = prefix_length, prefix
5354             end
5355             return line
5356         end)
5357         str = str:gsub('^' .. min_prefix, '')

```

```
5358     end
```

If the parser is top-level and the `texComments` or `hybrid` Lua options are enabled, we will strip all plain TeX comments from the input string `str` together with the trailing newline characters.

```
5359     if toplevel and (options.texComments or options.hybrid) then
5360         str = lpeg.match(Ct(parsers.commented_line^1), str)
5361         str = util.rope_to_string(str)
5362     end
5363     local res = lpeg.match(grammar(), str)
5364     if res == nil then
5365         error(format("%s failed on:\n%s", name, str:sub(1,20)))
5366     else
5367         return res
5368     end
5369 end
5370 end
5371
5372 local parse_blocks
5373     = create_parser("parse_blocks",
5374         function()
5375             return larsers.blocks
5376         end, true)
5377
5378 local parse_blocks_nested
5379     = create_parser("parse_blocks_nested",
5380         function()
5381             return larsers.blocks_nested
5382         end, false)
5383
5384 local parse_inlines
5385     = create_parser("parse_inlines",
5386         function()
5387             return larsers.inlines
5388         end, false)
5389
5390 local parse_inlines_no_link
5391     = create_parser("parse_inlines_no_link",
5392         function()
5393             return larsers.inlines_no_link
5394         end, false)
5395
5396 local parse_inlines_no_inline_note
5397     = create_parser("parse_inlines_no_inline_note",
5398         function()
5399             return larsers.inlines_no_inline_note
5400         end, false)
```



```

5401
5402 local parse_inlines_no_html
5403     = create_parser("parse_inlines_no_html",
5404                   function()
5405                       return larsers.inlines_no_html
5406                   end, false)
5407
5408 local parse_inlines_nbsp
5409     = create_parser("parse_inlines_nbsp",
5410                   function()
5411                       return larsers.inlines_nbsp
5412                   end, false)

```

### 3.1.5.3 Parsers Used for Markdown Lists (local)

```

5413 if options.hashEnumerators then
5414     larsers.dig = parsers.digit + parsers.hash
5415 else
5416     larsers.dig = parsers.digit
5417 end
5418
5419 larsers.enumerator = C(larsers.dig^3 * parsers.period) * #parsers.spacing
5420                   + C(larsers.dig^2 * parsers.period) * #parsers.spacing
5421                   * (parsers.tab + parsers.space^1)
5422                   + C(larsers.dig * parsers.period) * #parsers.spacing
5423                   * (parsers.tab + parsers.space^-2)
5424                   + parsers.space * C(larsers.dig^2 * parsers.period)
5425                   * #parsers.spacing
5426                   + parsers.space * C(larsers.dig * parsers.period)
5427                   * #parsers.spacing
5428                   * (parsers.tab + parsers.space^-1)
5429                   + parsers.space * parsers.space * C(larsers.dig^1
5430                   * parsers.period) * #parsers.spacing

```

### 3.1.5.4 Parsers Used for Blockquotes (local)

```

5431 -- strip off leading > and indents, and run through blocks
5432 larsers.blockquote_body = ((parsers.leader * parsers.more * parsers.space^-
5433                             1)/""
5434                             * parsers.linechar^0 * parsers.newline)^1
5435                             * (-(parsers.leader * parsers.more
5436                               + parsers.blankline) * parsers.linechar^1
5437                               * parsers.newline)^0
5438
5439 if not options.breakableBlockquotes then
5440     larsers.blockquote_body = larsers.blockquote_body
5441     * (parsers.blankline^0 / "")
5442 end

```

### 3.1.5.5 Parsers Used for Citations (local)

```
5442 larsers.citations = function(text_cites, raw_cites)
5443   local function normalize(str)
5444     if str == "" then
5445       str = nil
5446     else
5447       str = (options.citationNbsps and parse_inlines_nbsp or
5448         parse_inlines)(str)
5449     end
5450     return str
5451   end
5452
5453   local cites = {}
5454   for i = 1,#raw_cites,4 do
5455     cites[#cites+1] = {
5456       prenote = normalize(raw_cites[i]),
5457       suppress_author = raw_cites[i+1] == "-",
5458       name = writer.citation(raw_cites[i+2]),
5459       postnote = normalize(raw_cites[i+3]),
5460     }
5461   end
5462   return writer.citations(text_cites, cites)
5463 end
```

### 3.1.5.6 Parsers Used for Footnotes (local)

```
5464 local rawnotes = {}
5465
5466 -- like indirect_link
5467 local function lookup_note(ref)
5468   return writer.defer_call(function()
5469     local found = rawnotes[normalize_tag(ref)]
5470     if found then
5471       return writer.note(parse_blocks_nested(found))
5472     else
5473       return {"[", parse_inlines("^" .. ref), "]" }
5474     end
5475   end)
5476 end
5477
5478 local function register_note(ref,rawnote)
5479   rawnotes[normalize_tag(ref)] = rawnote
5480   return ""
5481 end
5482
5483 larsers.NoteRef = parsers.RawNoteRef / lookup_note
5484
```

```

5485
5486 larsers.NoteBlock = parsers.leader * parsers.RawNoteRef * parsers.colon
5487                   * parsers.spnl * parsers.indented_blocks(parsers.chunk)
5488                   / register_note
5489
5490 larsers.InlineNote = parsers.circumflex
5491                   * (parsers.tag / parse_inlines_no_inline_note) -- no notes inside
5492                   / writer.note

```

### 3.1.5.7 Parsers Used for Tables (local)

```

5493 larsers.table_row = pipe_table_row(true
5494                                   , (C((parsers.linechar - parsers.pipe)^1)
5495                                   / parse_inlines)
5496                                   , parsers.pipe
5497                                   , (C((parsers.linechar - parsers.pipe)^0)
5498                                   / parse_inlines))
5499
5500 if options.tableCaptions then
5501   larsers.table_caption = #parsers.table_caption_beginning
5502                           * parsers.table_caption_beginning
5503                           * Ct(parsers.IndentedInline^1)
5504                           * parsers.newline
5505 else
5506   larsers.table_caption = parsers.fail
5507 end
5508
5509 larsers.PipeTable = Ct(larsers.table_row * parsers.newline
5510                       * parsers.table_hline
5511                       * (parsers.newline * larsers.table_row)^0)
5512                       / make_pipe_table_rectangular
5513                       * larsers.table_caption^-1
5514                       / writer.table

```

### 3.1.5.8 Helpers for Links and References (local)

```

5515 -- List of references defined in the document
5516 local references
5517
5518 -- add a reference to the list
5519 local function register_link(tag,url,title)
5520   references[normalize_tag(tag)] = { url = url, title = title }
5521   return ""
5522 end
5523
5524 -- lookup link reference and return either
5525 -- the link or nil and fallback text.
5526 local function lookup_reference(label,sps,tag)

```

```

5527     local tagpart
5528     if not tag then
5529         tag = label
5530         tagpart = ""
5531     elseif tag == "" then
5532         tag = label
5533         tagpart = "[]"
5534     else
5535         tagpart = {"[", parse_inlines(tag), "]" }
5536     end
5537     if sps then
5538         tagpart = {sps, tagpart}
5539     end
5540     local r = references[normalize_tag(tag)]
5541     if r then
5542         return r
5543     else
5544         return nil, {"[", parse_inlines(label), "]", tagpart}
5545     end
5546 end
5547
5548 -- lookup link reference and return a link, if the reference is found,
5549 -- or a bracketed label otherwise.
5550 local function indirect_link(label,sps,tag)
5551     return writer.defer_call(function()
5552         local r, fallback = lookup_reference(label,sps,tag)
5553         if r then
5554             return writer.link(parse_inlines_no_link(label), r.url, r.title)
5555         else
5556             return fallback
5557         end
5558     end)
5559 end
5560
5561 -- lookup image reference and return an image, if the reference is found,
5562 -- or a bracketed label otherwise.
5563 local function indirect_image(label,sps,tag)
5564     return writer.defer_call(function()
5565         local r, fallback = lookup_reference(label,sps,tag)
5566         if r then
5567             return writer.image(writer.string(label), r.url, r.title)
5568         else
5569             return {"!", fallback}
5570         end
5571     end)
5572 end

```

### 3.1.5.9 Inline Elements (local)

```
5573 larsers.Str      = (parsers.normalchar * (parsers.normalchar + parsers.at)^0)
5574                / writer.string
5575
5576 larsers.Symbol   = (parsers.specialchar - parsers.tightblocksep)
5577                / writer.string
5578
5579 larsers.Ellipsis = P("...") / writer.ellipsis
5580
5581 larsers.Smart    = larsers.Ellipsis
5582
5583 larsers.Code     = parsers.inticks / writer.code
5584
5585 if options.blankBeforeBlockquote then
5586   larsers.bqstart = parsers.fail
5587 else
5588   larsers.bqstart = parsers.more
5589 end
5590
5591 if options.blankBeforeHeading then
5592   larsers.headerstart = parsers.fail
5593 else
5594   larsers.headerstart = parsers.hash
5595                       + (parsers.line * (parsers.equal^1 + parsers.dash^1)
5596                          * parsers.optionalspace * parsers.newline)
5597 end
5598
5599 if not options.fencedCode or options.blankBeforeCodeFence then
5600   larsers.fencestart = parsers.fail
5601 else
5602   larsers.fencestart = parsers.fencehead(parsers.backtick)
5603                       + parsers.fencehead(parsers.tilde)
5604 end
5605
5606 larsers.Endline  = parsers.newline * -( -- newline, but not before...
5607                                     parsers.blankline -- paragraph break
5608                                     + parsers.tightblocksep -- nested list
5609                                     + parsers.eof          -- end of document
5610                                     + larsers.bqstart
5611                                     + larsers.headerstart
5612                                     + larsers.fencestart
5613                                     ) * parsers.spacechar^0
5614                / (options.hardLineBreaks and writer.linebreak
5615                   or writer.space)
5616
5617 larsers.OptionalIndent
5618                = parsers.spacechar^1 / writer.space
```

```

5619
5620 larsers.Space      = parsers.spacechar^2 * larsers.Endline / writer.linebreak
5621                   + parsers.spacechar^1 * larsers.Endline^-1 * parsers.eof / ""
5622                   + parsers.spacechar^1 * larsers.Endline
5623                               * parsers.optionalspace
5624                               / (options.hardLineBreaks
5625                                   and writer.linebreak
5626                                   or writer.space)
5627                   + parsers.spacechar^1 * parsers.optionalspace
5628                               / writer.space
5629
5630 larsers.NonbreakingEndline
5631     = parsers.newline * -( -- newline, but not before...
5632         parsers.blankline -- paragraph break
5633         + parsers.tightblocksep -- nested list
5634         + parsers.eof        -- end of document
5635         + larsers.bqstart
5636         + larsers.headerstart
5637         + larsers.fencestart
5638     ) * parsers.spacechar^0
5639     / (options.hardLineBreaks and writer.linebreak
5640         or writer.nbsp)
5641
5642 larsers.NonbreakingSpace
5643     = parsers.spacechar^2 * larsers.Endline / writer.linebreak
5644     + parsers.spacechar^1 * larsers.Endline^-1 * parsers.eof / ""
5645     + parsers.spacechar^1 * larsers.Endline
5646                               * parsers.optionalspace
5647                               / (options.hardLineBreaks
5648                                   and writer.linebreak
5649                                   or writer.nbsp)
5650     + parsers.spacechar^1 * parsers.optionalspace
5651                               / writer.nbsp
5652
5653 if options.underscores then
5654   larsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,
5655                                   parsers.doubleasterisks)
5656                   + parsers.between(parsers.Inline, parsers.doubleunderscores,
5657                                   parsers.doubleunderscores)
5658   ) / writer.strong
5659
5660   larsers.Emph  = ( parsers.between(parsers.Inline, parsers.asterisk,
5661                                   parsers.asterisk)
5662                   + parsers.between(parsers.Inline, parsers.underscore,
5663                                   parsers.underscore)
5664   ) / writer.emphasis
5665 else

```

```

5666     larsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,
5667                                     parsers.doubleasterisks)
5668                                     ) / writer.strong
5669
5670     larsers.Emph   = ( parsers.between(parsers.Inline, parsers.asterisk,
5671                                     parsers.asterisk)
5672                                     ) / writer.emphasis
5673 end
5674
5675 larsers.AutoLinkUrl   = parsers.less
5676                       * C(parsers.alphanumeric^1 * P("://") * parsers.urlchar^1)
5677                       * parsers.more
5678                       / function(url)
5679                           return writer.link(writer.escape(url), url)
5680                       end
5681
5682 larsers.AutoLinkEmail = parsers.less
5683                       * C((parsers.alphanumeric + S("-._+"))^1
5684                           * P("@") * parsers.urlchar^1)
5685                       * parsers.more
5686                       / function(email)
5687                           return writer.link(writer.escape(email),
5688                                               "mailto:".email)
5689                       end
5690
5691 larsers.AutoLinkRelativeReference
5692     = parsers.less
5693       * C(parsers.urlchar^1)
5694       * parsers.more
5695       / function(url)
5696           return writer.link(writer.escape(url), url)
5697       end
5698
5699 larsers.DirectLink   = (parsers.tag / parse_inlines_no_link) -- no links inside link
5700                       * parsers.spnl
5701                       * parsers.lparent
5702                       * (parsers.url + Cc("")) -- link can be empty [foo]()
5703                       * parsers.optionaltitle
5704                       * parsers.rparent
5705                       / writer.link
5706
5707 larsers.IndirectLink = parsers.tag * (C(parsers.spnl) * parsers.tag)^-
5708                               1
5709                               / indirect_link
5710
5711 -- parse a link or image (direct or indirect)
5711 larsers.Link         = larsers.DirectLink + larsers.IndirectLink

```

```

5712
5713 larsers.DirectImage = parsers.exclamation
5714 * (parsers.tag / parse_inlines)
5715 * parsers.spnl
5716 * parsers.lparent
5717 * (parsers.url + Cc("")) -- link can be empty [foo]()
5718 * parsers.optionaltitle
5719 * parsers.rparent
5720 / writer.image
5721
5722 larsers.IndirectImage = parsers.exclamation * parsers.tag
5723 * (C(parsers.spnl) * parsers.tag)^-1 / indirect_image
5724
5725 larsers.Image = larsers.DirectImage + larsers.IndirectImage
5726
5727 larsers.TextCitations = Ct((parsers.spnl
5728 * Cc(""))
5729 * parsers.citation_name
5730 * ((parsers.spnl
5731 * parsers.lbracket
5732 * parsers.citation_headless_body
5733 * parsers.rbracket) + Cc("")))^1)
5734 / function(raw_cites)
5735 return larsers.citations(true, raw_cites)
5736 end
5737
5738 larsers.ParenthesizedCitations
5739 = Ct((parsers.spnl
5740 * parsers.lbracket
5741 * parsers.citation_body
5742 * parsers.rbracket)^1)
5743 / function(raw_cites)
5744 return larsers.citations(false, raw_cites)
5745 end
5746
5747 larsers.Citations = larsers.TextCitations + larsers.ParenthesizedCitations
5748
5749 -- avoid parsing long strings of * or _ as emph/strong
5750 larsers.UlOrStarLine = parsers.asterisk^4 + parsers.underscore^4
5751 / writer.string
5752
5753 larsers.EscapedChar = parsers.backslash * C(parsers.escapable) / writer.string
5754
5755 larsers.InlineHtml = parsers.emptyelt_any / writer.inline_html_tag
5756 + (parsers.htmlcomment / parse_inlines_no_html)
5757 / writer.inline_html_comment
5758 + parsers.htmlinstruction

```



```

5759         + parsers.openelt_any / writer.inline_html_tag
5760         + parsers.closeelt_any / writer.inline_html_tag
5761
5762     larsers.HtmlEntity = parsers.hexentity / entities.hex_entity / writer.string
5763         + parsers.decententity / entities.dec_entity / writer.string
5764         + parsers.tagentity / entities.char_entity / writer.string

```

### 3.1.5.10 Block Elements (local)

```

5765     larsers.ContentBlock = parsers.leader
5766         * (parsers.localfilepath + parsers.onlineimageurl)
5767         * parsers.contentblock_tail
5768         / writer.contentblock
5769
5770     larsers.DisplayHtml = (parsers.htmlcomment / parse_blocks_nested)
5771         / writer.block_html_comment
5772         + parsers.emptyelt_block / writer.block_html_element
5773         + parsers.openelt_exact("hr") / writer.block_html_element
5774         + parsers.in_matched_block_tags / writer.block_html_element
5775         + parsers.htmlinstruction
5776
5777     larsers.Verbatim = Cs( (parsers.blanklines
5778         * ((parsers.indentedline - parsers.blankline)^1)^1
5779         ) / expandtabs / writer.verbatim
5780
5781     larsers.FencedCode = (parsers.TildeFencedCode
5782         + parsers.BacktickFencedCode)
5783         / function(infostring, code)
5784             return writer.fencedCode(writer.string(infostring),
5785                 expandtabs(code))
5786         end
5787
5788     larsers.JekyllData = Cmt( C((parsers.line - P("---") - P("..."))^0)
5789         , function(s, i, text)
5790             local data
5791             local ran_ok, error = pcall(function()
5792                 local tinyyaml = require("markdown-tinyyaml")
5793                 data = tinyyaml.parse(text, {timestamps=false})
5794             end)
5795             if ran_ok and data ~= nil then
5796                 return true, writer.jekyllData(data, function(s)
5797                     return parse_blocks_nested(s)
5798                 end, nil)
5799             else
5800                 return false
5801             end
5802         end

```

```

5803         )
5804
5805 larsers.UnexpectedJekyllData
5806     = P("----")
5807     * parsers.blankline / 0
5808     * #(-parsers.blankline) -- if followed by blank, it's an hrule
5809     * larsers.JekyllData
5810     * (P("----") + P("..."))
5811
5812 larsers.ExpectedJekyllData
5813     = ( P("----")
5814     * parsers.blankline / 0
5815     * #(-parsers.blankline) -- if followed by blank, it's an hrule
5816     )^-1
5817     * larsers.JekyllData
5818     * (P("----") + P("..."))^-1
5819
5820 larsers.Blockquote = Cs(larsers.blockquote_body^1)
5821 / parse_blocks_nested / writer.blockquote
5822
5823 larsers.HorizontalRule = ( parsers.lineof(parsers.asterisk)
5824 + parsers.lineof(parsers.dash)
5825 + parsers.lineof(parsers.underscore)
5826 ) / writer.hrule
5827
5828 larsers.Reference = parsers.define_reference_parser / register_link
5829
5830 larsers.Paragraph = parsers.nonindentspace * Ct(parsers.Inline^1)
5831 * ( parsers.newline
5832 * ( parsers.blankline^1
5833 + #parsers.hash
5834 + #(parsers.leader * parsers.more * parsers.space^-
5835 1)
5835 + parsers.eof
5836 )
5837 + parsers.eof )
5838 / writer.paragraph
5839
5840 larsers.Plain = parsers.nonindentspace * Ct(parsers.Inline^1)
5841 / writer.plain

```

### 3.1.5.11 Lists (local)

```

5842 larsers.starter = parsers.bullet + larsers.enumerator
5843
5844 if options.taskLists then
5845     larsers.tickbox = ( parsers.ticked_box

```

```

5846             + parsers.halfticked_box
5847             + parsers.unticked_box
5848             ) / writer.tickbox
5849     else
5850         larsers.tickbox = parsers.fail
5851     end
5852
5853     -- we use \001 as a separator between a tight list item and a
5854     -- nested list under it.
5855     larsers.NestedList           = Cs((parsers.optionallyindentedline
5856                                 - larsers.starter)^1)
5857                                 / function(a) return "\001"..a end
5858
5859     larsers.ListBlockLine       = parsers.optionallyindentedline
5860                                 - parsers.blankline - (parsers.indent^-1
5861                                                         * larsers.starter)
5862
5863     larsers.ListBlock           = parsers.line * larsers.ListBlockLine^0
5864
5865     larsers.ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
5866                                 * larsers.ListBlock
5867
5868     larsers.TightListItem = function(starter)
5869         return -larsers.HorizontalRule
5870             * (Cs(starter / "" * larsers.tickbox^-1 * larsers.ListBlock * larsers.Nested
5871 1)
5872             / parse_blocks_nested)
5873             * -(parsers.blanklines * parsers.indent)
5874     end
5875
5876     larsers.LooseListItem = function(starter)
5877         return -larsers.HorizontalRule
5878             * Cs( starter / "" * larsers.tickbox^-1 * larsers.ListBlock * Cc("\n")
5879             * (larsers.NestedList + larsers.ListContinuationBlock^0)
5880             * (parsers.blanklines / "\n\n")
5881             ) / parse_blocks_nested
5882     end
5883
5884     larsers.BulletList = ( Ct(larsers.TightListItem(parsers.bullet)^1) * Cc(true)
5885                             * parsers.skipblanklines * -parsers.bullet
5886                             + Ct(larsers.LooseListItem(parsers.bullet)^1) * Cc(false)
5887                             * parsers.skipblanklines )
5888                             / writer.bulletlist
5889
5890     local function ordered_list(items,tight,startNumber)
5891         if options.startNumber then
5892             startNumber = tonumber(startNumber) or 1 -- fallback for '#'

```

```

5892     if startNumber ~= nil then
5893         startNumber = math.floor(startNumber)
5894     end
5895     else
5896         startNumber = nil
5897     end
5898     return writer.orderedlist(items,tight,startNumber)
5899 end
5900
5901 larsers.OrderedList = Cg(larsers.enumerator, "listtype") *
5902     ( Ct(larsers.TightListItem(Cb("listtype")))
5903       * larsers.TightListItem(larsers.enumerator)^0)
5904     * Cc(true) * parsers.skipblanklines * -larsers.enumerator
5905     + Ct(larsers.LooseListItem(Cb("listtype")))
5906       * larsers.LooseListItem(larsers.enumerator)^0)
5907     * Cc(false) * parsers.skipblanklines
5908     ) * Cb("listtype") / ordered_list
5909
5910 local function definition_list_item(term, defs, tight)
5911     return { term = parse_inlines(term), definitions = defs }
5912 end
5913
5914 larsers.DefinitionListItemLoose = C(parsers.line) * parsers.skipblanklines
5915     * Ct((parsers.defstart
5916         * parsers.indented_blocks(parsers.dlchunk)
5917         / parse_blocks_nested)^1)
5918     * Cc(false) / definition_list_item
5919
5920 larsers.DefinitionListItemTight = C(parsers.line)
5921     * Ct((parsers.defstart * parsers.dlchunk
5922         / parse_blocks_nested)^1)
5923     * Cc(true) / definition_list_item
5924
5925 larsers.DefinitionList = ( Ct(larsers.DefinitionListItemLoose^1) * Cc(false)
5926     + Ct(larsers.DefinitionListItemTight^1)
5927     * (parsers.skipblanklines
5928       * -larsers.DefinitionListItemLoose * Cc(true))
5929     ) / writer.definitionlist

```

### 3.1.5.12 Blank (local)

```

5930 larsers.Blank           = parsers.blankline / ""
5931                        + larsers.NoteBlock
5932                        + larsers.Reference
5933                        + (parsers.tightblocksep / "\n")

```

### 3.1.5.13 Headings (local)

```

5934 -- parse atx header
5935 if options.headerAttributes then
5936     larsers.AtHeading = Cg(parsers.HeadingStart,"level")
5937         * parsers.optionalspace
5938         * (C(((parsers.linechar
5939             - ((parsers.hash^1
5940                 * parsers.optionalspace
5941                 * parsers.HeadingAttributes~-1
5942                 + parsers.HeadingAttributes)
5943                 * parsers.optionalspace
5944                 * parsers.newline))
5945             * (parsers.linechar
5946                 - parsers.hash
5947                 - parsers.lbrace)^0)^1)
5948             / parse_inlines)
5949         * Cg(Ct(parsers.newline
5950             + (parsers.hash^1
5951                 * parsers.optionalspace
5952                 * parsers.HeadingAttributes~-1
5953                 + parsers.HeadingAttributes)
5954                 * parsers.optionalspace
5955                 * parsers.newline), "attributes")
5956         * Cb("level")
5957         * Cb("attributes")
5958         / writer.heading
5959
5960     larsers.SetextHeading = #(parsers.line * S("--"))
5961         * (C(((parsers.linechar
5962             - (parsers.HeadingAttributes
5963                 * parsers.optionalspace
5964                 * parsers.newline))
5965             * (parsers.linechar
5966                 - parsers.lbrace)^0)^1)
5967             / parse_inlines)
5968         * Cg(Ct(parsers.newline
5969             + (parsers.HeadingAttributes
5970                 * parsers.optionalspace
5971                 * parsers.newline)), "attributes")
5972         * parsers.HeadingLevel
5973         * Cb("attributes")
5974         * parsers.optionalspace
5975         * parsers.newline
5976         / writer.heading
5977 else
5978     larsers.AtHeading = Cg(parsers.HeadingStart,"level")
5979         * parsers.optionalspace
5980         * (C(parsers.line) / strip_atx_end / parse_inlines)

```

```

5981             * Cb("level")
5982             / writer.heading
5983
5984     larsers.SettextHeading = #(parsers.line * S("--"))
5985             * Ct(parsers.linechar^1 / parse_inlines)
5986             * parsers.newline
5987             * parsers.HeadingLevel
5988             * parsers.optionalspace
5989             * parsers.newline
5990             / writer.heading
5991 end
5992
5993 larsers.Heading = larsers.AtxHeading + larsers.SettextHeading

```

### 3.1.5.14 Syntax Specification

```

5994 local syntax =
5995     { "Blocks",
5996
5997         Blocks = ( V("ExpectedJekyllData")
5998             * (V("Blank")^0 / writer.interblocksep)
5999             )^-1
6000             * V("Blank")^0
6001             * V("Block")^-1
6002             * (V("Blank")^0 / writer.interblocksep
6003             * V("Block"))^0
6004             * V("Blank")^0 * parsers.eof,
6005
6006         Blank = larsers.Blank,
6007
6008         UnexpectedJekyllData = larsers.UnexpectedJekyllData,
6009         ExpectedJekyllData = larsers.ExpectedJekyllData,
6010
6011         Block = V("ContentBlock")
6012             + V("UnexpectedJekyllData")
6013             + V("Blockquote")
6014             + V("PipeTable")
6015             + V("Verbatim")
6016             + V("FencedCode")
6017             + V("HorizontalRule")
6018             + V("BulletList")
6019             + V("OrderedList")
6020             + V("Heading")
6021             + V("DefinitionList")
6022             + V("DisplayHtml")
6023             + V("Paragraph")
6024             + V("Plain"),

```

```

6025
6026     ContentBlock           = larsers.ContentBlock,
6027     Blockquote             = larsers.Blockquote,
6028     Verbatim               = larsers.Verbatim,
6029     FencedCode             = larsers.FencedCode,
6030     HorizontalRule         = larsers.HorizontalRule,
6031     BulletList             = larsers.BulletList,
6032     OrderedList            = larsers.OrderedList,
6033     Heading                = larsers.Heading,
6034     DefinitionList         = larsers.DefinitionList,
6035     DisplayHtml            = larsers.DisplayHtml,
6036     Paragraph              = larsers.Paragraph,
6037     PipeTable              = larsers.PipeTable,
6038     Plain                  = larsers.Plain,
6039
6040     Inline                  = V("Str")
6041                             + V("Space")
6042                             + V("Endline")
6043                             + V("U1OrStarLine")
6044                             + V("Strong")
6045                             + V("Emph")
6046                             + V("InlineNote")
6047                             + V("NoteRef")
6048                             + V("Citations")
6049                             + V("Link")
6050                             + V("Image")
6051                             + V("Code")
6052                             + V("AutoLinkUrl")
6053                             + V("AutoLinkEmail")
6054                             + V("AutoLinkRelativeReference")
6055                             + V("InlineHtml")
6056                             + V("HtmlEntity")
6057                             + V("EscapedChar")
6058                             + V("Smart")
6059                             + V("Symbol"),
6060
6061     IndentedInline          = V("Str")
6062                             + V("OptionalIndent")
6063                             + V("Endline")
6064                             + V("U1OrStarLine")
6065                             + V("Strong")
6066                             + V("Emph")
6067                             + V("InlineNote")
6068                             + V("NoteRef")
6069                             + V("Citations")
6070                             + V("Link")
6071                             + V("Image")

```

```

6072         + V("Code")
6073         + V("AutoLinkUrl")
6074         + V("AutoLinkEmail")
6075         + V("AutoLinkRelativeReference")
6076         + V("InlineHtml")
6077         + V("HtmlEntity")
6078         + V("EscapedChar")
6079         + V("Smart")
6080         + V("Symbol"),
6081
6082     Str           = larsers.Str,
6083     Space        = larsers.Space,
6084     OptionalIndent = larsers.OptionalIndent,
6085     Endline      = larsers.Endline,
6086     U1OrStarLine = larsers.U1OrStarLine,
6087     Strong       = larsers.Strong,
6088     Emph         = larsers.Emph,
6089     InlineNote   = larsers.InlineNote,
6090     NoteRef      = larsers.NoteRef,
6091     Citations    = larsers.Citations,
6092     Link         = larsers.Link,
6093     Image        = larsers.Image,
6094     Code         = larsers.Code,
6095     AutoLinkUrl  = larsers.AutoLinkUrl,
6096     AutoLinkEmail = larsers.AutoLinkEmail,
6097     AutoLinkRelativeReference
6098         = larsers.AutoLinkRelativeReference,
6099     InlineHtml   = larsers.InlineHtml,
6100     HtmlEntity   = larsers.HtmlEntity,
6101     EscapedChar  = larsers.EscapedChar,
6102     Smart        = larsers.Smart,
6103     Symbol       = larsers.Symbol,
6104 }
6105
6106 if not options.citations then
6107     syntax.Citations = parsers.fail
6108 end
6109
6110 if not options.contentBlocks then
6111     syntax.ContentBlock = parsers.fail
6112 end
6113
6114 if not options.codeSpans then
6115     syntax.Code = parsers.fail
6116 end
6117
6118 if not options.definitionLists then

```



```

6119     syntax.DefinitionList = parsers.fail
6120 end
6121
6122 if not options.fencedCode then
6123     syntax.FencedCode = parsers.fail
6124 end
6125
6126 if not options.footnotes then
6127     syntax.NoteRef = parsers.fail
6128 end
6129
6130 if not options.html then
6131     syntax.DisplayHtml = parsers.fail
6132     syntax.InlineHtml = parsers.fail
6133     syntax.HtmlEntity = parsers.fail
6134 end
6135
6136 if not options.inlineFootnotes then
6137     syntax.InlineNote = parsers.fail
6138 end
6139
6140 if not options.jekyllData then
6141     syntax.UnexpectedJekyllData = parsers.fail
6142 end
6143
6144 if not options.jekyllData or not options.expectJekyllData then
6145     syntax.ExpectedJekyllData = parsers.fail
6146 end
6147
6148 if options.preserveTabs then
6149     options.stripIndent = false
6150 end
6151
6152 if not options.pipeTables then
6153     syntax.PipeTable = parsers.fail
6154 end
6155
6156 if not options.smartEllipses then
6157     syntax.Smart = parsers.fail
6158 end
6159
6160 if not options.relativeReferences then
6161     syntax.AutoLinkRelativeReference = parsers.fail
6162 end
6163
6164 local blocks_nested_t = util.table_copy(syntax)
6165 blocks_nested_t.ExpectedJekyllData = parsers.fail

```

```

6166 larsers.blocks_nested = Ct(blocks_nested_t)
6167
6168 larsers.blocks = Ct(syntax)
6169
6170 local inlines_t = util.table_copy(syntax)
6171 inlines_t[1] = "Inlines"
6172 inlines_t.Inlines = parsers.Inline^0 * (parsers.spacing^0 * parsers.eof / "")
6173 larsers.inlines = Ct(inlines_t)
6174
6175 local inlines_no_link_t = util.table_copy(inlines_t)
6176 inlines_no_link_t.Link = parsers.fail
6177 larsers.inlines_no_link = Ct(inlines_no_link_t)
6178
6179 local inlines_no_inline_note_t = util.table_copy(inlines_t)
6180 inlines_no_inline_note_t.InlineNote = parsers.fail
6181 larsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
6182
6183 local inlines_no_html_t = util.table_copy(inlines_t)
6184 inlines_no_html_t.DisplayHtml = parsers.fail
6185 inlines_no_html_t.InlineHtml = parsers.fail
6186 inlines_no_html_t.HtmlEntity = parsers.fail
6187 larsers.inlines_no_html = Ct(inlines_no_html_t)
6188
6189 local inlines_nbsp_t = util.table_copy(inlines_t)
6190 inlines_nbsp_t.Endline = larsers.NonbreakingEndline
6191 inlines_nbsp_t.Space = larsers.NonbreakingSpace
6192 larsers.inlines_nbsp = Ct(inlines_nbsp_t)

```

**3.1.5.15 Exported Conversion Function** Define `reader->convert` as a function that converts markdown string `input` into a plain `TEX` output and returns it. Note that the converter assumes that the input has UNIX line endings.

```

6193 function self.convert(input)
6194     references = {}

```

When determining the name of the cache file, create salt for the hashing function out of the package version and the passed options recognized by the Lua interface (see Section 2.1.2). The `cacheDir` option is disregarded.

```

6195     local opt_string = {}
6196     for k,_ in pairs(defaultOptions) do
6197         local v = options[k]
6198         if k ~= "cacheDir" then
6199             opt_string[#opt_string+1] = k .. "=" .. tostring(v)
6200         end
6201     end
6202     table.sort(opt_string)
6203     local salt = table.concat(opt_string, ",") .. "," .. metadata.version

```

```

6204     local output
        If we cache markdown documents, produce the cache file and transform its filename
        to plain TeX output via the writer->pack method.
6205     local function convert(input)
6206         local document = parse_blocks(input)
6207         return util.rope_to_string(writer.document(document))
6208     end
6209     if options.eagerCache or options.finalizeCache then
6210         local name = util.cache(options.cacheDir, input, salt, convert, ".md" .. writer.s
6211         output = writer.pack(name)
        Otherwise, return the result of the conversion directly.
6212     else
6213         output = convert(input)
6214     end
        If the finalizeCache option is enabled, populate the frozen cache in the
        file frozenCacheFileName with an entry for markdown document number
        frozenCacheCounter.
6215     if options.finalizeCache then
6216         local file, mode
6217         if options.frozenCacheCounter > 0 then
6218             mode = "a"
6219         else
6220             mode = "w"
6221         end
6222         file = assert(io.open(options.frozenCacheFileName, mode),
6223             [[could not open file ]] .. options.frozenCacheFileName
6224             .. [[ for writing]])
6225         assert(file:write([[\\expandafter\\global\\expandafter\\def\\csname ]]
6226             .. [[markdownFrozenCache]] .. options.frozenCacheCounter
6227             .. [[\\endcsname{]] .. output .. [[]]] .. "\\n"))
6228         assert(file:close())
6229     end
6230     return output
6231 end
6232 return self
6233 end

```

### 3.1.6 Conversion from Markdown to Plain TeX

The `new` method returns the `reader->convert` function of a reader object associated with the Lua interface options (see Section 2.1.2) `options` and with a writer object associated with `options`.

```

6234 function M.new(options)
6235     local writer = M.writer.new(options)

```

```

6236 local reader = M.reader.new(writer, options)
6237 return reader.convert
6238 end
6239
6240 return M

```

### 3.1.7 Command-Line Implementation

The command-line implementation provides the actual conversion routine for the command-line interface described in Section 2.1.5.

```

6241
6242 local input
6243 if input_filename then
6244   local input_file = assert(io.open(input_filename, "r"),
6245     [[could not open file ]] .. input_filename .. [[ for reading]])
6246   input = assert(input_file:read("*a"))
6247   assert(input_file:close())
6248 else
6249   input = assert(io.read("*a"))
6250 end
6251

```

First, ensure that the `options.cacheDir` directory exists.

```

6252 local lfs = require("lfs")
6253 if options.cacheDir and not lfs.isdir(options.cacheDir) then
6254   assert(lfs.mkdir(options["cacheDir"]))
6255 end
6256
6257 local ran_ok, kpse = pcall(require, "kpse")
6258 if ran_ok then kpse.set_program_name("luatex") end
6259 local md = require("markdown")

```

Since we are loading the rest of the Lua implementation dynamically, check that both the `markdown` module and the command line implementation are the same version.

```

6260 if metadata.version ~= md.metadata.version then
6261   warn("markdown-cli.lua " .. metadata.version .. " used with " ..
6262     "markdown.lua " .. md.metadata.version .. ".")
6263 end
6264 local convert = md.new(options)

```

Since the Lua converter expects UNIX line endings, normalize the input. Also add a line ending at the end of the file in case the input file has none.

```

6265 local output = convert(input:gsub("\r\n?", "\n") .. "\n")
6266
6267 if output_filename then
6268   local output_file = assert(io.open(output_filename, "w"),
6269     [[could not open file ]] .. output_filename .. [[ for writing]])
6270   assert(output_file:write(output))

```

```

6271  assert(output_file:close())
6272  else
6273    assert(io.write(output))
6274  end

```

## 3.2 Plain T<sub>E</sub>X Implementation

The plain T<sub>E</sub>X implementation provides macros for the interfacing between T<sub>E</sub>X and Lua and for the buffering of input text. These macros are then used to implement the macros for the conversion from markdown to plain T<sub>E</sub>X exposed by the plain T<sub>E</sub>X interface (see Section 2.2).

### 3.2.1 Logging Facilities

```

6275 \ifx\markdownInfo\undefined
6276   \def\markdownInfo#1{%
6277     \immediate\write-1{(1.\the\inputlineno) markdown.tex info: #1}}%
6278 \fi
6279 \ifx\markdownWarning\undefined
6280   \def\markdownWarning#1{%
6281     \immediate\write16{(1.\the\inputlineno) markdown.tex warning: #1}}%
6282 \fi
6283 \ifx\markdownError\undefined
6284   \def\markdownError#1#2{%
6285     \errhelp{#2.}%
6286     \errmessage{(1.\the\inputlineno) markdown.tex error: #1}}%
6287 \fi

```

### 3.2.2 Finalizing and Freezing the Cache

When the `\markdownOptionFinalizeCache` option is enabled, then the `\markdownFrozenCacheCounter` counter is used to enumerate the markdown documents using the Lua interface `frozenCacheCounter` option.

When the `\markdownOptionFrozenCache` option is enabled, then the `\markdownFrozenCacheCounter` counter is used to render markdown documents from the frozen cache without invoking Lua.

```

6288 \newcount\markdownFrozenCacheCounter

```

### 3.2.3 Token Renderer Prototypes

The following definitions should be considered placeholder.

```

6289 \def\markdownRendererInterblockSeparatorPrototype{\par}%
6290 \def\markdownRendererLineBreakPrototype{\hfil\break}%
6291 \let\markdownRendererEllipsisPrototype\dots
6292 \def\markdownRendererNbspPrototype{~}%
6293 \def\markdownRendererLeftBracePrototype{\char`\{ }%

```

```

6294 \def\markdownRendererRightBracePrototype{\char`}\}%
6295 \def\markdownRendererDollarSignPrototype{\char`$}%
6296 \def\markdownRendererPercentSignPrototype{\char`\}%
6297 \def\markdownRendererAmpersandPrototype{\&}%
6298 \def\markdownRendererUnderscorePrototype{\char`_}%
6299 \def\markdownRendererHashPrototype{\char`#}%
6300 \def\markdownRendererCircumflexPrototype{\char`^}%
6301 \def\markdownRendererBackslashPrototype{\char`\}%
6302 \def\markdownRendererTildePrototype{\char`~}%
6303 \def\markdownRendererPipePrototype{|}%
6304 \def\markdownRendererCodeSpanPrototype#1{\tt#1}%
6305 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
6306 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
6307   \markdownInput{#3}}%
6308 \def\markdownRendererContentBlockOnlineImagePrototype{%
6309   \markdownRendererImage}%
6310 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
6311   \markdownRendererInputFencedCode{#3}{#2}}%
6312 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
6313 \def\markdownRendererUlBeginPrototype{}%
6314 \def\markdownRendererUlBeginTightPrototype{}%
6315 \def\markdownRendererUlItemPrototype{}%
6316 \def\markdownRendererUlItemEndPrototype{}%
6317 \def\markdownRendererUlEndPrototype{}%
6318 \def\markdownRendererUlEndTightPrototype{}%
6319 \def\markdownRendererOlBeginPrototype{}%
6320 \def\markdownRendererOlBeginTightPrototype{}%
6321 \def\markdownRendererOlItemPrototype{}%
6322 \def\markdownRendererOlItemWithNumberPrototype#1{}%
6323 \def\markdownRendererOlItemEndPrototype{}%
6324 \def\markdownRendererOlEndPrototype{}%
6325 \def\markdownRendererOlEndTightPrototype{}%
6326 \def\markdownRendererDlBeginPrototype{}%
6327 \def\markdownRendererDlBeginTightPrototype{}%
6328 \def\markdownRendererDlItemPrototype#1{#1}%
6329 \def\markdownRendererDlItemEndPrototype{}%
6330 \def\markdownRendererDlDefinitionBeginPrototype{}%
6331 \def\markdownRendererDlDefinitionEndPrototype{\par}%
6332 \def\markdownRendererDlEndPrototype{}%
6333 \def\markdownRendererDlEndTightPrototype{}%
6334 \def\markdownRendererEmphasisPrototype#1{\it#1}%
6335 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
6336 \def\markdownRendererBlockQuoteBeginPrototype{\par\begin\group\it}%
6337 \def\markdownRendererBlockQuoteEndPrototype{\end\group\par}%
6338 \def\markdownRendererInputVerbatimPrototype#1{%
6339   \par{\tt\input#1\relax{}}\par}%
6340 \def\markdownRendererInputFencedCodePrototype#1#2{%

```

```

6341 \markdownRendererInputVerbatimPrototype{#1}%
6342 \def\markdownRendererHeadingOnePrototype#1{#1}%
6343 \def\markdownRendererHeadingTwoPrototype#1{#1}%
6344 \def\markdownRendererHeadingThreePrototype#1{#1}%
6345 \def\markdownRendererHeadingFourPrototype#1{#1}%
6346 \def\markdownRendererHeadingFivePrototype#1{#1}%
6347 \def\markdownRendererHeadingSixPrototype#1{#1}%
6348 \def\markdownRendererHorizontalRulePrototype{}%
6349 \def\markdownRendererFootnotePrototype#1{#1}%
6350 \def\markdownRendererCitePrototype#1{}%
6351 \def\markdownRendererTextCitePrototype#1{}%
6352 \def\markdownRendererTickedBoxPrototype{[X]}%
6353 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
6354 \def\markdownRendererUntickedBoxPrototype{[ ]}%

```

**3.2.3.1 YAML Metadata Renderer Prototypes** To keep track of the current type of structure we inhabit when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_datatypes_seq` stack. At every step of the traversal, the stack will contain one of the following constants at any position  $p$ :

`\c_@@_jekyll_data_sequence_tl` The currently traversed branch of the YAML document contains a sequence at depth  $p$ .

`\c_@@_jekyll_data_mapping_tl` The currently traversed branch of the YAML document contains a mapping at depth  $p$ .

`\c_@@_jekyll_data_scalar_tl` The currently traversed branch of the YAML document contains a scalar value at depth  $p$ .

```

6355 \ExplSyntaxOn
6356 \seq_new:N \g_@@_jekyll_data_datatypes_seq
6357 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
6358 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
6359 \tl_const:Nn \c_@@_jekyll_data_scalar_tl { scalar }

```

To keep track of our current place when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_wildcard_absolute_address_seq` stack of keys using the `\markdown_jekyll_data_push_address_segment:n` macro.

```

6360 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
6361 \cs_new:Nn \markdown_jekyll_data_push_address_segment:n
6362 {
6363   \seq_if_empty:NF
6364     \g_@@_jekyll_data_datatypes_seq
6365     {
6366       \seq_get_right:NN
6367         \g_@@_jekyll_data_datatypes_seq
6368         \l_tmpa_tl

```

If we are currently in a sequence, we will put an asterisk (\*) instead of a key into `\g_@@_jekyll_data_wildcard_absolute_address_seq` to make it represent a *wildcard*. Keeping a wildcard instead of a precise address makes it easy for the users to react to *any* item of a sequence regardless of how many there are, which can often be useful.

```

6369     \tl_if_eq:NNTF
6370         \l_tmpa_tl
6371         \c_@@_jekyll_data_sequence_tl
6372         {
6373             \seq_put_right:Nn
6374                 \g_@@_jekyll_data_wildcard_absolute_address_seq
6375                 { * }
6376         }
6377         {
6378             \seq_put_right:Nn
6379                 \g_@@_jekyll_data_wildcard_absolute_address_seq
6380                 { #1 }
6381         }
6382     }
6383 }

```

Out of `\g_@@_jekyll_data_wildcard_absolute_address_seq`, we will construct the following two token lists:

`\g_@@_jekyll_data_wildcard_absolute_address_tl` An *absolute wildcard*: The wildcard from the root of the document prefixed with a slash (/) with individual keys and asterisks also delimited by slashes. Allows the users to react to complex context-sensitive structures with ease.

For example, the `name` key in the following YAML document would correspond to the `/*/person/name` absolute wildcard:

```
[{person: {name: Elon, surname: Musk}}]
```

`\g_@@_jekyll_data_wildcard_relative_address_tl` A *relative wildcard*: The rightmost segment of the wildcard. Allows the users to react to simple context-free structures.

For example, the `name` key in the following YAML document would correspond to the `name` relative wildcard:

```
[{person: {name: Elon, surname: Musk}}]
```

We will construct `\g_@@_jekyll_data_wildcard_absolute_address_tl` using the `\markdown_jekyll_data_concatenate_address:NN` macro and we will construct both token lists using the `\markdown_jekyll_data_update_address_tls:` macro.



```

6384 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
6385 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
6386 \cs_new:Nn \markdown_jekyll_data_concatenate_address:NN
6387 {
6388   \seq_pop_left:NN #1 \l_tmpa_tl
6389   \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
6390   \seq_put_left:NV #1 \l_tmpa_tl
6391 }
6392 \cs_new:Nn \markdown_jekyll_data_update_address_tls:
6393 {
6394   \markdown_jekyll_data_concatenate_address:NN
6395   \g_@@_jekyll_data_wildcard_absolute_address_seq
6396   \g_@@_jekyll_data_wildcard_absolute_address_tl
6397   \seq_get_right:NN
6398   \g_@@_jekyll_data_wildcard_absolute_address_seq
6399   \g_@@_jekyll_data_wildcard_relative_address_tl
6400 }

```

To make sure that the stacks and token lists stay in sync, we will use the `\markdown_jekyll_data_push:nN` and `\markdown_jekyll_data_pop:` macros.

```

6401 \cs_new:Nn \markdown_jekyll_data_push:nN
6402 {
6403   \markdown_jekyll_data_push_address_segment:n
6404   { #1 }
6405   \seq_put_right:NV
6406   \g_@@_jekyll_data_datatypes_seq
6407   #2
6408   \markdown_jekyll_data_update_address_tls:
6409 }
6410 \cs_new:Nn \markdown_jekyll_data_pop:
6411 {
6412   \seq_pop_right:NN
6413   \g_@@_jekyll_data_wildcard_absolute_address_seq
6414   \l_tmpa_tl
6415   \seq_pop_right:NN
6416   \g_@@_jekyll_data_datatypes_seq
6417   \l_tmpa_tl
6418   \markdown_jekyll_data_update_address_tls:
6419 }

```

To set a single key–value, we will use the `\markdown_jekyll_data_set_keyval:Nn` macro, ignoring unknown keys. To set key–values for both absolute and relative wildcards, we will use the `\markdown_jekyll_data_set_keyvals:nn` macro.

```

6420 \cs_new:Nn \markdown_jekyll_data_set_keyval:nn
6421 {
6422   \keys_set_known:nn
6423   { markdown/jekyllData }
6424   { { #1 } = { #2 } }

```

```

6425 }
6426 \cs_generate_variant:Nn
6427 \markdown_jekyll_data_set_keyval:nn
6428 { Vn }
6429 \cs_new:Nn \markdown_jekyll_data_set_keyvals:nn
6430 {
6431   \markdown_jekyll_data_push:nN
6432   { #1 }
6433   \c_@@_jekyll_data_scalar_tl
6434   \markdown_jekyll_data_set_keyval:Vn
6435   \g_@@_jekyll_data_wildcard_absolute_address_tl
6436   { #2 }
6437   \markdown_jekyll_data_set_keyval:Vn
6438   \g_@@_jekyll_data_wildcard_relative_address_tl
6439   { #2 }
6440   \markdown_jekyll_data_pop:
6441 }

```

Finally, we will register our macros as token renderer prototypes to be able to react to the traversal of a YAML document.

```

6442 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
6443   \markdown_jekyll_data_push:nN
6444   { #1 }
6445   \c_@@_jekyll_data_sequence_tl
6446 }
6447 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
6448   \markdown_jekyll_data_push:nN
6449   { #1 }
6450   \c_@@_jekyll_data_mapping_tl
6451 }
6452 \def\markdownRendererJekyllDataSequenceEndPrototype{
6453   \markdown_jekyll_data_pop:
6454 }
6455 \def\markdownRendererJekyllDataMappingEndPrototype{
6456   \markdown_jekyll_data_pop:
6457 }
6458 \def\markdownRendererJekyllDataBooleanPrototype#1#2{
6459   \markdown_jekyll_data_set_keyvals:nn
6460   { #1 }
6461   { #2 }
6462 }
6463 \def\markdownRendererJekyllDataEmptyPrototype#1{}
6464 \def\markdownRendererJekyllDataNumberPrototype#1#2{
6465   \markdown_jekyll_data_set_keyvals:nn
6466   { #1 }
6467   { #2 }
6468 }

```

```

6469 \def\markdownRendererJekyllDataStringPrototype#1#2{
6470   \markdown_jekyll_data_set_keyvals:nn
6471     { #1 }
6472     { #2 }
6473 }
6474 \ExplSyntaxOff

```

### 3.2.4 Lua Snippets

After the `\markdownPrepareLuaOptions` macro has been fully expanded, the `\markdownLuaOptions` macro will expand to a Lua table that contains the plain TeX options (see Section 2.2.2) in a format recognized by Lua (see Section 2.1.2).

```

6475 \ExplSyntaxOn
6476 \tl_new:N \g_@@_formatted_lua_options_tl
6477 \tl_const:Nn \c_@@_lua_option_type_boolean { boolean }
6478 \tl_const:Nn \c_@@_lua_option_type_counter { counter }
6479 \tl_const:Nn \c_@@_lua_option_type_number { number }
6480 \tl_const:Nn \c_@@_lua_option_type_string { string }
6481 \cs_new:Nn \@@_format_lua_options:
6482 {
6483   \tl_gclear:N
6484   \g_@@_formatted_lua_options_tl
6485   \seq_map_function:NN
6486     \g_@@_lua_options_seq
6487     \@@_format_lua_option:n
6488 }
6489 \cs_new:Nn \@@_format_lua_option:n
6490 {
6491   \@@_typecheck_lua_option:n { #1 }
6492   \tl_set:Nn
6493     \l_tmpb_tl
6494   %   TODO: Replace with \str_uppercase:n in TeX Live 2020.
6495     { \str_upper_case:n { #1 } }
6496   \tl_set:Nx
6497     \l_tmpa_tl
6498     {
6499       markdownOption
6500       \tl_head:f { \l_tmpb_tl }
6501       \tl_tail:n { #1 }
6502     }
6503   \prop_get:NnN
6504     \g_@@_lua_option_types_prop
6505     { #1 }
6506     \l_tmpb_tl
6507   \cs_if_free:cTF
6508     { \l_tmpa_tl }

```

```

6509     { }
6510     {
6511         \tl_case:NnF
6512         \l_tmpb_tl
6513         {
6514             \c_@@_lua_option_type_string
6515             {
6516                 \tl_gput_right:Nx
6517                 \g_@@_formatted_lua_options_tl
6518                 { #1~=="      \cs:w \l_tmpa_tl \cs_end: ",~ }
6519             }
6520             \c_@@_lua_option_type_counter
6521             {
6522                 \tl_gput_right:Nx
6523                 \g_@@_formatted_lua_options_tl
6524                 { #1~== \the \cs:w \l_tmpa_tl \cs_end: ,~ }
6525             }
6526         }
6527         {
6528             \tl_gput_right:Nx
6529             \g_@@_formatted_lua_options_tl
6530             { #1~==      \cs:w \l_tmpa_tl \cs_end: ,~ }
6531         }
6532     }
6533 }
6534 \msg_new:nnn
6535 { markdown }
6536 { undefined-lua-option }
6537 {
6538     Lua~option~#1~is~undefined.
6539 }
6540 \msg_new:nnn
6541 { markdown }
6542 { failed-typecheck-for-boolean-lua-option }
6543 {
6544     Lua~option~#1~has~value~#2,~
6545     but~a~boolean~(true~or~false)~was~expected.
6546 }
6547 \cs_new:Nn \@@_typecheck_lua_option:n
6548 {
6549     \tl_set:Nn
6550     \l_tmpb_tl
6551     %     TODO: Replace with \str_uppercase:n in TeX Live 2020.
6552     { \str_upper_case:n { #1 } }
6553     \tl_set:Nx
6554     \l_tmpa_tl
6555     {

```

```

6556     markdownOption
6557     \tl_head:f { \l_tmpb_tl }
6558     \tl_tail:n { #1 }
6559   }
6560 \prop_get:NnNTF
6561 \g_@@_lua_option_types_prop
6562 { #1 }
6563 \l_tmpb_tl
6564 {
6565   \cs_if_free:cTF
6566   { \l_tmpa_tl }
6567   { }
6568   {
6569     \tl_case:Nn
6570     \l_tmpb_tl
6571     {
6572       \c_@@_lua_option_type_boolean
6573       {
6574         \tl_case:cnF
6575         { \l_tmpa_tl }
6576         {
6577           \c_@@_lua_option_value_true { }
6578           \c_@@_lua_option_value_false { }
6579         }
6580         {
6581           \msg_error:nxxx
6582           { markdown }
6583           { failed-typecheck-for-boolean-lua-option }
6584           { #1 }
6585           { \l_tmpa_tl }
6586         }
6587       }
6588     }
6589   }
6590 }
6591 {
6592   \msg_error:nnn
6593   { markdown }
6594   { undefined-lua-option }
6595   { #1 }
6596 }
6597 }
6598 \let\markdownPrepareLuaOptions=\@@_format_lua_options:
6599 \def\markdownLuaOptions{{ \g_@@_formatted_lua_options_tl }}
6600 \ExplSyntaxOff

```

The `\markdownPrepare` macro contains the Lua code that is executed prior to

any conversion from markdown to plain T<sub>E</sub>X. It exposes the `convert` function for the use by any further Lua code.

```
6601 \def\markdownPrepare{%
```

First, ensure that the `\markdownOptionCacheDir` directory exists.

```
6602 local lfs = require("lfs")
6603 local cacheDir = "\markdownOptionCacheDir"
6604 if not lfs.isdir(cacheDir) then
6605     assert(lfs.mkdir(cacheDir))
6606 end
```

Next, load the `markdown` module and create a converter function using the plain T<sub>E</sub>X options, which were serialized to a Lua table via the `\markdownLuaOptions` macro.

```
6607 local md = require("markdown")
6608 local convert = md.new(\markdownLuaOptions)
6609 }%
```

### 3.2.5 Buffering Markdown Input

The `\markdownIfOption{<name>}{<iftrue>}{<iffalse>}` macro is provided for testing, whether the value of `\markdownOption<name>` is `true`. If the value is `true`, then `<iftrue>` is expanded, otherwise `<iffalse>` is expanded.

```
6610 \ExplSyntaxOn
6611 \tl_const:Nn \c_@@_lua_option_value_true { true }
6612 \tl_const:Nn \c_@@_lua_option_value_false { false }
6613 \cs_new:Nn \@@_if_option:nTF
6614 {
6615     \tl_set:Nn
6616         \l_tmpb_tl
6617     %     TODO: Replace with \str_uppercase:n in TeX Live 2020.
6618         { \str_upper_case:n { #1 } }
6619     \tl_set:Nx
6620         \l_tmpa_tl
6621         {
6622             markdownOption
6623             \tl_head:f { \l_tmpb_tl }
6624             \tl_tail:n { #1 }
6625         }
6626     \cs_if_free:cTF
6627         { \l_tmpa_tl }
6628         {
6629             \prop_get:NnN
6630                 \g_@@_default_lua_options_prop
6631                 { #1 }
6632                 \l_tmpb_tl
6633         }
6634 }
```

```

6635     \tl_set:Nf
6636     \l_tmpb_tl
6637     { \cs:w \l_tmpa_tl \cs_end: }
6638   }
6639   \tl_if_eq:NNTF
6640   \l_tmpb_tl
6641   \c_@@_lua_option_value_true
6642   { #2 }
6643   { #3 }
6644 }
6645 \let\markdownIfOption=\@@_if_option:nTF
6646 \ExplSyntaxOff

```

The macros `\markdownInputFileStream` and `\markdownOutputFileStream` contain the number of the input and output file streams that will be used for the IO operations of the package.

```

6647 \csname newread\endcsname\markdownInputFileStream
6648 \csname newwrite\endcsname\markdownOutputFileStream

```

The `\markdownReadAndConvertTab` macro contains the tab character literal.

```

6649 \begingroup
6650 \catcode\^^I=12%
6651 \gdef\markdownReadAndConvertTab{^^I}%
6652 \endgroup

```

The `\markdownReadAndConvert` macro is largely a rewrite of the  $\LaTeX 2\epsilon$  `\filecontents` macro to plain  $\TeX$ .

```

6653 \begingroup

```

Make the newline and tab characters active and swap the character codes of the backslash symbol (`\`) and the pipe symbol (`|`), so that we can use the backslash as an ordinary character inside the macro definition. Likewise, swap the character codes of the percent sign (`%`) and the ampersand (`@`), so that we can remove percent signs from the beginning of lines when `\markdownOptionStripPercentSigns` is enabled.

```

6654 \catcode\^^M=13%
6655 \catcode\^^I=13%
6656 \catcode|=0%
6657 \catcode\\=12%
6658 |catcode`=14%
6659 |catcode`|=12@
6660 |gdef|markdownReadAndConvert#1#2{@
6661 |begingroup@

```

If we are not reading markdown documents from the frozen cache, open the `\markdownOptionInputTempFileName` file for writing.

```

6662 |markdownIfOption{frozenCache}{-}{@
6663 |immediate|openout|markdownOutputFileStream@
6664 |markdownOptionInputTempFileName|relax@

```

```

6665     |markdownInfo{Buffering markdown input into the temporary @
6666         input file "|markdownOptionInputTempFileName" and scanning @
6667         for the closing token sequence "#1"}@
6668     }@

```

Locally change the category of the special plain T<sub>E</sub>X characters to *other* in order to prevent unwanted interpretation of the input. Change also the category of the space character, so that we can retrieve it unaltered.

```

6669     |def|do##1{|catcode`##1=12}|dospecials@
6670     |catcode`| =12@
6671     |markdownMakeOther@

```

The `\markdownReadAndConvertStripPercentSigns` macro will process the individual lines of output, stripping away leading percent signs (%) when `\markdownOptionStripPercentSigns` is enabled. Notice the use of the comments (@) to ensure that the entire macro is at a single line and therefore no (active) newline symbols ( $\text{\char"000D}$ ) are produced.

```

6672     |def|markdownReadAndConvertStripPercentSign##1{@
6673         |markdownIfOption{stripPercentSigns}{@
6674             |if##1%@
6675                 |expandafter|expandafter|expandafter@
6676                 |markdownReadAndConvertProcessLine@
6677             |else@
6678                 |expandafter|expandafter|expandafter@
6679                 |markdownReadAndConvertProcessLine@
6680                 |expandafter|expandafter|expandafter##1@
6681             |fi@
6682         }{@
6683             |expandafter@
6684             |markdownReadAndConvertProcessLine@
6685             |expandafter##1@
6686         }@
6687     }@

```

The `\markdownReadAndConvertProcessLine` macro will process the individual lines of output. Notice the use of the comments (@) to ensure that the entire macro is at a single line and therefore no (active) newline symbols ( $\text{\char"000D}$ ) are produced.

```

6688     |def|markdownReadAndConvertProcessLine##1##2##3|relax{@

```

If we are not reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, store the line in the `\markdownOptionInputTempFileName` file. If we are reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, gobble the line.

```

6689         |ifx|relax##3|relax@
6690         |markdownIfOption{frozenCache}{}@
6691         |immediate|write|markdownOutputFileStream{##1}@

```



```

6692     }@
6693     |else@

```

When the ending token sequence appears in the line, make the next newline character close the `\markdownOptionInputTempFileName` file, return the character categories back to the former state, convert the `\markdownOptionInputTempFileName` file from markdown to plain T<sub>E</sub>X, `\input` the result of the conversion, and expand the ending control sequence.

```

6694     |def^^M{@
6695         |markdownInfo{The ending token sequence was found}@
6696         |markdownIfOption{frozenCache}{}@
6697         |immediate|closeout|markdownOutputFileStream@
6698     }@
6699     |endgroup@
6700     |markdownInput{@
6701         |markdownOptionOutputDir@
6702         /|markdownOptionInputTempFileName@
6703     }@
6704     #2}@
6705     |fi@

```

Repeat with the next line.

```

6706     ^^M}@

```

Make the tab character active at expansion time and make it expand to a literal tab character.

```

6707     |catcode`|^I=13@
6708     |def^^I{|markdownReadAndConvertTab}@

```

Make the newline character active at expansion time and make it consume the rest of the line on expansion. Throw away the rest of the first line and pass the second line to the `\markdownReadAndConvertProcessLine` macro.

```

6709     |catcode`|^M=13@
6710     |def^^M##1^^M{@
6711         |def^^M###1^^M{@
6712             |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
6713         ^^M}@
6714     ^^M}@

```

Reset the character categories back to the former state.

```

6715 |endgroup

```

### 3.2.6 Lua Shell Escape Bridge

The following T<sub>E</sub>X code is intended for T<sub>E</sub>X engines that do not provide direct access to Lua, but expose the shell of the operating system. This corresponds to the `\markdownMode` values of 0 and 1.

The `\markdownLuaExecute` macro defined here and in Section 3.2.7 are meant to be indistinguishable to the remaining code.

The package assumes that although the user is not using the LuaTeX engine, their TeX distribution contains it, and uses shell access to produce and execute Lua scripts using the TeXLua interpreter [1, Section 3.1.1].

```
6716 \ifnum\markdownMode<2\relax
6717 \ifnum\markdownMode=0\relax
6718   \markdownInfo{Using mode 0: Shell escape via write18}%
6719 \else
6720   \markdownInfo{Using mode 1: Shell escape via os.execute}%
6721 \fi
```

The `\markdownExecuteShellEscape` macro contains the numeric value indicating whether the shell access is enabled (1), disabled (0), or restricted (2).

Inherit the value of the the `\pdfshellescape` (LuaTeX, PdfTeX) or the `\shellescape` (XeTeX) commands. If neither of these commands is defined and Lua is available, attempt to access the `status.shell_escape` configuration item.

If you cannot detect, whether the shell access is enabled, act as if it were.

```
6722 \ifx\pdfshellescape\undefined
6723   \ifx\shellescape\undefined
6724     \ifnum\markdownMode=0\relax
6725       \def\markdownExecuteShellEscape{1}%
6726     \else
6727       \def\markdownExecuteShellEscape{%
6728         \directlua{tex.sprint(status.shell_escape or "1")}}%
6729     \fi
6730   \else
6731     \let\markdownExecuteShellEscape\shellescape
6732   \fi
6733 \else
6734   \let\markdownExecuteShellEscape\pdfshellescape
6735 \fi
```

The `\markdownExecuteDirect` macro executes the code it has received as its first argument by writing it to the output file stream 18, if Lua is unavailable, or by using the Lua `os.execute` method otherwise.

```
6736 \ifnum\markdownMode=0\relax
6737   \def\markdownExecuteDirect#1{\immediate\write18{#1}}%
6738 \else
6739   \def\markdownExecuteDirect#1{%
6740     \directlua{os.execute("\luaescapestring{#1}")}}%
6741 \fi
```

The `\markdownExecute` macro is a wrapper on top of `\markdownExecuteDirect` that checks the value of `\markdownExecuteShellEscape` and prints an error message if the shell is inaccessible.

```

6742 \def\markdownExecute#1{%
6743   \ifnum\markdownExecuteShellEscape=1\relax
6744     \markdownExecuteDirect{#1}%
6745   \else
6746     \markdownError{I can not access the shell}{Either run the TeX
6747       compiler with the --shell-escape or the --enable-write18 flag,
6748       or set shell_escape=t in the texmf.cnf file}%
6749   \fi}%

```

The `\markdownLuaExecute` macro executes the Lua code it has received as its first argument. The Lua code may not directly interact with the TeX engine, but it can use the `print` function in the same manner it would use the `tex.print` method.

```
6750 \begingroup
```

Swap the category code of the backslash symbol and the pipe symbol, so that we may use the backslash symbol freely inside the Lua code.

```

6751 \catcode`\|=0%
6752 \catcode`\|=12%
6753 \gdef\markdownLuaExecute#1{%

```

Create the file `\markdownOptionHelperScriptFileName` and fill it with the input Lua code prepended with `kpathsea` initialization, so that Lua modules from the TeX distribution are available.

```

6754   |immediate|openout|markdownOutputFileStream=%
6755   |markdownOptionHelperScriptFileName
6756   |markdownInfo{Writing a helper Lua script to the file
6757     "|markdownOptionHelperScriptFileName"}%
6758   |immediate|write|markdownOutputFileStream{%
6759     local ran_ok, error = pcall(function()
6760       local ran_ok, kpse = pcall(require, "kpse")
6761       if ran_ok then kpse.set_program_name("luatex") end
6762       #1
6763     end)

```

If there was an error, use the file `\markdownOptionErrorTempFileName` to store the error message.

```

6764     if not ran_ok then
6765       local file = io.open("%
6766         |markdownOptionOutputDir
6767         /|markdownOptionErrorTempFileName", "w")
6768       if file then
6769         file:write(error .. "\n")
6770         file:close()
6771       end
6772       print('\|markdownError{An error was encountered while executing
6773         Lua code}{For further clues, examine the file
6774         "|markdownOptionOutputDir
6775         /|markdownOptionErrorTempFileName}')

```

```

6776     end}%
6777 |immediate|closeout|markdownOutputFileStream

```

Execute the generated `\markdownOptionHelperScriptFileName` Lua script using the `TeXLua` binary and store the output in the `\markdownOptionOutputTempFileName` file.

```

6778 |markdownInfo{Executing a helper Lua script from the file
6779   "|markdownOptionHelperScriptFileName" and storing the result in the
6780   file "|markdownOptionOutputTempFileName"}%
6781 |markdownExecute{texlua "|markdownOptionOutputDir
6782   /|markdownOptionHelperScriptFileName" > %
6783   "|markdownOptionOutputDir
6784   /|markdownOptionOutputTempFileName"}%

```

`\input` the generated `\markdownOptionOutputTempFileName` file.

```

6785 |input|markdownOptionOutputTempFileName|relax}%
6786 |endgroup

```

### 3.2.7 Direct Lua Access

The following `TeX` code is intended for `TeX` engines that provide direct access to Lua (`LuaTeX`). The macro `\markdownLuaExecute` defined here and in Section 3.2.6 are meant to be indistinguishable to the remaining code. This corresponds to the `\markdownMode` value of 2.

```

6787 \else
6788 \markdownInfo{Using mode 2: Direct Lua access}%

```

The direct Lua access version of the `\markdownLuaExecute` macro is defined in terms of the `\directlua` primitive. The `print` function is set as an alias to the `\tex.print` method in order to mimic the behaviour of the `\markdownLuaExecute` definition from Section 3.2.6,

```

6789 \begingroup

```

Swap the category code of the backslash symbol and the pipe symbol, so that we may use the backslash symbol freely inside the Lua code.

```

6790 \catcode`\|=0%
6791 \catcode`\|=12%
6792 |gdef|markdownLuaExecute#1{%
6793   |directlua{%
6794     local function print(input)
6795       local output = {}
6796       for line in input:gmatch("[^\r\n]+") do
6797         table.insert(output, line)
6798       end
6799       tex.print(output)
6800     end
6801     #1

```

```

6802     }%
6803   }%
6804 |endgroup
6805 \fi

```

### 3.2.8 Typesetting Markdown

The `\markdownInput` macro uses an implementation of the `\markdownLuaExecute` macro to convert the contents of the file whose filename it has received as its single argument from markdown to plain TeX.

```
6806 \begingroup
```

Swap the category code of the backslash symbol and the pipe symbol, so that we may use the backslash symbol freely inside the Lua code.

```

6807   \catcode`\=0%
6808   \catcode`\|=12%
6809   |gdef|markdownInput#1{%

```

Change the category code of the percent sign (%) to other, so that a user of the `hybrid` Lua option or a malevolent actor can't produce TeX comments in the plain TeX output of the Markdown package.

```

6810     |begingroup
6811     |catcode`\%=12

```

If we are reading from the frozen cache, input it, expand the corresponding `\markdownFrozenCache<number>` macro, and increment `\markdownFrozenCacheCounter`.

```

6812     |markdownIfOption{frozenCache}{%
6813       |ifnum|markdownFrozenCacheCounter=0|relax
6814         |markdownInfo{Reading frozen cache from
6815           "|markdownOptionFrozenCacheFileName"}%
6816         |input|markdownOptionFrozenCacheFileName|relax
6817       |fi
6818       |markdownInfo{Including markdown document number
6819         "|the|markdownFrozenCacheCounter" from frozen cache}%
6820       |csname markdownFrozenCache|the|markdownFrozenCacheCounter|endcsname
6821       |global|advance|markdownFrozenCacheCounter by 1|relax
6822     }-%
6823     |markdownInfo{Including markdown document "#1"}%

```

Attempt to open the markdown document to record it in the `.log` and `.fls` files. This allows external programs such as L<sup>A</sup>T<sub>E</sub>X<sub>M</sub>k to track changes to the markdown document.

```

6824     |openin|markdownInputFileStream#1
6825     |closein|markdownInputFileStream
6826     |markdownPrepareLuaOptions
6827     |markdownLuaExecute{%
6828     |markdownPrepare

```

```

6829     local file = assert(io.open("#1", "r"),
6830     [[could not open file "#1" for reading]])
6831     local input = assert(file:read("*a"))
6832     assert(file:close())

```

Since the Lua converter expects UNIX line endings, normalize the input. Also add a line ending at the end of the file in case the input file has none.

```

6833     print(convert(input:gsub("\r\n?", "\n") .. "\n"))}%

```

If we are finalizing the frozen cache, increment `\markdownFrozenCacheCounter`.

```

6834     |markdownIfOption{finalizeCache}{%
6835     |global|advance|markdownFrozenCacheCounter by 1|relax
6836     }%
6837     }%
6838     |endgroup
6839     }%
6840 |endgroup

```

### 3.3 $\LaTeX$ Implementation

The  $\LaTeX$  implementation makes use of the fact that, apart from some subtle differences,  $\LaTeX$  implements the majority of the plain  $\TeX$  format [10, Section 9]. As a consequence, we can directly reuse the existing plain  $\TeX$  implementation.

The  $\LaTeX$  implementation redefines the plain  $\TeX$  logging macros (see Section 3.2.1) to use the  $\LaTeX$  `\PackageInfo`, `\PackageWarning`, and `\PackageError` macros.

```

6841 \newcommand\markdownInfo[1]{\PackageInfo{markdown}{#1}}%
6842 \newcommand\markdownWarning[1]{\PackageWarning{markdown}{#1}}%
6843 \newcommand\markdownError[2]{\PackageError{markdown}{#1}{#2.}}%
6844 \input markdown/markdown
6845 \def\markdownVersionSpace{ }%
6846 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
6847 \markdownVersion\markdownVersionSpace markdown renderer]%

```

Use reflection to define the `renderers` and `rendererPrototypes` keys of `\markdownSetup` as well as the keys that correspond to Lua options.

```

6848 \ExplSyntaxOn
6849 \@@_latex_define_renderers:
6850 \@@_latex_define_renderer_prototypes:
6851 \ExplSyntaxOff

```

#### 3.3.1 Logging Facilities

The  $\LaTeX$  implementation redefines the plain  $\TeX$  logging macros (see Section 3.2.1) to use the  $\LaTeX$  `\PackageInfo`, `\PackageWarning`, and `\PackageError` macros.

### 3.3.2 Typesetting Markdown

The `\markdownInputPlainTeX` macro is used to store the original plain  $\TeX$  implementation of the `\markdownInput` macro. The `\markdownInput` is then redefined to accept an optional argument with options recognized by the  $\LaTeX$  interface (see Section 2.3.2).

```
6852 \let\markdownInputPlainTeX\markdownInput
6853 \renewcommand\markdownInput[2] []{%
6854   \begingroup
6855     \markdownSetup{#1}%
6856     \markdownInputPlainTeX{#2}%
6857   \endgroup}%
```

The `markdown`, and `markdown*`  $\LaTeX$  environments are implemented using the `\markdownReadAndConvert` macro.

```
6858 \renewenvironment{markdown}{%
6859   \markdownReadAndConvert@markdown{}}{%
6860   \markdownEnd}%
6861 \renewenvironment{markdown*}[1]{%
6862   \markdownSetup{#1}%
6863   \markdownReadAndConvert@markdown*}{%
6864   \markdownEnd}%
6865 \begingroup
```

Locally swap the category code of the backslash symbol with the pipe symbol, and of the left (`{`) and right brace (`}`) with the less-than (`<`) and greater-than (`>`) signs. This is required in order that all the special symbols that appear in the first argument of the `\markdownReadAndConvert` macro have the category code *other*.

```
6866 \catcode`\|=0\catcode`\<=1\catcode`\>=2%
6867 \catcode`\|=12\catcode`\{=12\catcode`\}=12%
6868 \gdef\markdownReadAndConvert@markdown#1<%
6869   \markdownReadAndConvert<\end{markdown#1}>%
6870   <|\end<markdown#1>>>%
6871 \endgroup
```

**3.3.2.1  $\LaTeX$  Themes** This section implements the theme-loading mechanism and the example themes provided with the Markdown package.

```
6872 \ExplSyntaxOn
```

To keep track of our current place when packages themes have been nested, we will maintain the `\g_@@_latex_themes_seq` stack of theme names.

```
6873 \newcommand\markdownLaTeXThemeName{}
6874 \seq_new:N \g_@@_latex_themes_seq
6875 \seq_put_right:NV
6876   \g_@@_latex_themes_seq
6877   \markdownLaTeXThemeName
```

```

6878 \newcommand\markdownLaTeXThemeLoad[2]{
6879   \def\@tempa{%
6880     \def\markdownLaTeXThemeName{#2}
6881     \seq_put_right:NV
6882       \g_@@_latex_themes_seq
6883       \markdownLaTeXThemeName
6884     \RequirePackage{#1}
6885     \seq_pop_right:NN
6886       \g_@@_latex_themes_seq
6887     \l_tmpa_tl
6888     \seq_get_right:NN
6889       \g_@@_latex_themes_seq
6890     \l_tmpa_tl
6891     \exp_args:NNV
6892       \def
6893         \markdownLaTeXThemeName
6894         \l_tmpa_tl}
6895   \ifmarkdownLaTeXLoaded
6896     \@tempa
6897   \else
6898     \exp_args:No
6899     \AtEndOfPackage
6900     { \@tempa }
6901   \fi}
6902 \ExplSyntaxOff

```

The `witiko/dot` theme enables the `fencedCode` Lua option:

```
6903 \markdownSetup{fencedCode}%
```

We load the `ifthen` and `grffile` packages, see also Section 1.1.3:

```
6904 \RequirePackage{ifthen,grffile}
```

We store the previous definition of the fenced code token renderer prototype:

```
6905 \let\markdown@witiko@dot@oldRendererInputFencedCodePrototype
6906 \markdownRendererInputFencedCodePrototype
```

If the infostring starts with `dot ...`, we redefine the fenced code block token renderer prototype, so that it typesets the code block via Graphviz tools if and only if the `\markdownOptionFrozenCache` plain  $\TeX$  option is disabled and the code block has not been previously typeset:

```

6907 \renewcommand\markdownRendererInputFencedCode[2]{%
6908   \def\next##1 ##2\relax{%
6909     \ifthenelse{\equal{##1}{dot}}{%
6910       \markdownIfOption{frozenCache}{}{}%
6911       \immediate\write18{%
6912         if ! test -e #1.pdf.source || ! diff #1 #1.pdf.source;
6913         then
6914           dot -Tpdf -o #1.pdf #1;

```



```

6915         cp #1 #1.pdf.source;
6916         fi}}%

```

We include the typeset image using the image token renderer:

```

6917     \markdownRendererImage{Graphviz image}{#1.pdf}{#1.pdf}{##2}%

```

If the infostring does not start with `dot ...`, we use the previous definition of the fenced code token renderer prototype:

```

6918     }{%
6919     \markdown@witiko@dot@oldRendererInputFencedCodePrototype{#1}{#2}%
6920     }%
6921     }%
6922     \next#2 \relax}%

```

The `witiko/graphicx/http` theme stores the previous definition of the image token renderer prototype:

```

6923 \let\markdown@witiko@graphicx@http@oldRendererImagePrototype
6924 \markdownRendererImagePrototype

```

We load the catchfile and grffile packages, see also Section 1.1.3:

```

6925 \RequirePackage{catchfile,grffile}

```

We define the `\markdown@witiko@graphicx@http@counter` counter to enumerate the images for caching and the `\markdown@witiko@graphicx@http@filename` command, which will store the pathname of the file containing the pathname of the downloaded image file.

```

6926 \newcount\markdown@witiko@graphicx@http@counter
6927 \markdown@witiko@graphicx@http@counter=0
6928 \newcommand\markdown@witiko@graphicx@http@filename{%
6929     \markdownOptionCacheDir/witiko_graphicx_http%
6930     .\the\markdown@witiko@graphicx@http@counter}%

```

We define the `\markdown@witiko@graphicx@http@download` command, which will receive two arguments that correspond to the URL of the online image and to the pathname, where the online image should be downloaded. The command will produce a shell command that tries to download the online image to the pathname.

```

6931 \newcommand\markdown@witiko@graphicx@http@download[2]{%
6932     wget -O #2 #1 || curl --location -o #2 #1 || rm -f #2}

```

We locally swap the category code of the percentage sign with the line feed control character, so that we can use percentage signs in the shell code:

```

6933 \begingroup
6934 \catcode`\%=12
6935 \catcode`\^^A=14

```

We redefine the image token renderer prototype, so that it tries to download an online image.

```

6936 \global\def\markdownRendererImagePrototype#1#2#3#4{^^A
6937     \begingroup
6938     \edef\filename{\markdown@witiko@graphicx@http@filename}^^A

```

The image will be downloaded only if the image URL has the http or https protocols and the `\markdownOptionFrozenCache` plain TeX option is disabled:

```
6939 \markdownIfOption{frozenCache}{-}{^^A
6940 \immediate\write18{^^A
6941   if printf '%s' "#3" | grep -q -E '^https?:';
6942   then
```

The image will be downloaded to the pathname `\markdownOptionCacheDir/⟨the MD5 digest of the image URL⟩.⟨the suffix of the image URL⟩`:

```
6943     OUTPUT_PREFIX="\markdownOptionCacheDir";
6944     OUTPUT_BODY="$ (printf '%s' '#3' | md5sum | cut -d' ' -f1)";
6945     OUTPUT_SUFFIX="$ (printf '%s' '#3' | sed 's/.*[.]//')";
6946     OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";
```

The image will be downloaded only if it has not already been downloaded:

```
6947     if ! [ -e "$OUTPUT" ];
6948     then
6949         \markdown@witiko@graphicx@http@download{'#3'}{"$OUTPUT"};
6950         printf '%s' "$OUTPUT" > "\filename";
6951     fi;
```

If the image does not have the http or https protocols or the image has already been downloaded, the URL will be stored as-is:

```
6952     else
6953         printf '%s' '#3' > "\filename";
6954     fi}}^^A
```

We load the pathname of the downloaded image and we typeset the image using the previous definition of the image renderer prototype:

```
6955 \CatchFileDef{\filename}{\filename}{\newlinechar=-1}^^A
6956 \markdown@witiko@graphicx@http@oldRendererImagePrototype^^A
6957   {#1}{#2}{\filename}{#4}^^A
6958 \endgroup
6959 \global\advance\markdown@witiko@graphicx@http@counter by 1\relax}^^A
6960 \endgroup
```

The `witiko/tilde` theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```
6961 \renewcommand\markdownRendererTildePrototype{~}%
```

### 3.3.3 Options

The supplied package options are processed using the `\markdownSetup` macro.

```
6962 \DeclareOption*{%
6963   \expandafter\markdownSetup\expandafter{\CurrentOption}}%
6964 \ProcessOptions\relax
```

After processing the options, activate the `renderers`, `rendererPrototypes`, and `code` keys. The `code` key is used to immediately expand and execute code, which can be especially useful in L<sup>A</sup>T<sub>E</sub>X setup snippets.

```

6965 \define@key{markdownOptions}{renderers}{%
6966   \setkeys{markdownRenderers}{#1}%
6967   \def\KV@prefix{KV@markdownOptions@}}%
6968 \define@key{markdownOptions}{rendererPrototypes}{%
6969   \setkeys{markdownRendererPrototypes}{#1}%
6970   \def\KV@prefix{KV@markdownOptions@}}%
6971 \define@key{markdownOptions}{code}{#1}%

```

### 3.3.4 Token Renderer Prototypes

The following configuration should be considered placeholder. If the `plain` package option has been enabled (see Section 2.3.2.1), none of it will take effect.

```

6972 \ifmarkdownLaTeXplain\else
    If the tightLists Lua option is disabled or the current document class is beamer,
    do not load the paralist package.
6973 \markdownIfOption{tightLists}{
6974   \@ifclassloaded{beamer}{\RequirePackage{paralist}}%
6975 }{}

```

If we loaded the `paralist` package, define the respective renderer prototypes to make use of the capabilities of the package. Otherwise, define the renderer prototypes to fall back on the corresponding renderers for the non-tight lists.

```

6976 \@ifpackageloaded{paralist}{
6977   \markdownSetup{rendererPrototypes={
6978     ulBeginTight = {\begin{compactitem}},
6979     ulEndTight = {\end{compactitem}},
6980     olBeginTight = {\begin{compactenum}},
6981     olEndTight = {\end{compactenum}},
6982     dlBeginTight = {\begin{compactdesc}},
6983     dlEndTight = {\end{compactdesc}}}}
6984 }{
6985   \markdownSetup{rendererPrototypes={
6986     ulBeginTight = {\markdownRendererUlBegin},
6987     ulEndTight = {\markdownRendererUlEnd},
6988     olBeginTight = {\markdownRendererOlBegin},
6989     olEndTight = {\markdownRendererOlEnd},
6990     dlBeginTight = {\markdownRendererDlBegin},
6991     dlEndTight = {\markdownRendererDlEnd}}}}
6992 \RequirePackage{amsmath,ifthen}

```

Unless the `unicode-math` package has been loaded, load the `amssymb` package with symbols to be used for tickboxes.

```

6993 \@ifpackageloaded{unicode-math}{

```

```

6994 \markdownSetup{rendererPrototypes={
6995     untickedBox = {\$mdlgwhtsquare$},
6996 }}
6997 }{
6998 \RequirePackage{amssymb}
6999 \markdownSetup{rendererPrototypes={
7000     untickedBox = {\$square$},
7001 }}
7002 }
7003 \RequirePackage{csvsimple}
7004 \RequirePackage{fancyvrb}
7005 \RequirePackage{graphicx}
7006 \markdownSetup{rendererPrototypes={
7007     lineBreak = {\},
7008     leftBrace = {\textbraceleft},
7009     rightBrace = {\textbraceright},
7010     dollarSign = {\textdollar},
7011     underscore = {\textunderscore},
7012     circumflex = {\textasciicircum},
7013     backslash = {\textbackslash},
7014     tilde = {\textasciitilde},
7015     pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by T<sub>E</sub>X during the typesetting. Therefore, even if we don't know whether a span of text is part of math formula or not when we are parsing markdown,<sup>8</sup> we can reliably detect math mode inside the renderer.

Here, we will redefine the code span renderer prototype to typeset upright text in math formulae and typewriter text outside math formulae.

```

7016 codeSpan = {%
7017     \ifmmode
7018         \text{#1}%
7019     \else
7020         \texttt{#1}%
7021     \fi
7022 },
7023 contentBlock = {%
7024     \ifthenelse{\equal{#1}{csv}}{%
7025         \begin{table}%
7026             \begin{center}%
7027                 \csvautotabular{#3}%
7028             \end{center}
7029         \ifx\empty#4\empty\else

```

---

<sup>8</sup>This property may actually be undecidable. Suppose a span of text is a part of a macro definition. Then, whether the span of text is part of a math formula or not depends on where the macro is later used, which may easily be *both* inside and outside a math formula.

```

7030         \caption{#4}%
7031         \fi
7032     \end{table}%
7033 }{%
7034     \ifthenelse{\equal{#1}{tex}}{%
7035         \catcode`\%=14\relax
7036         \input #3\relax
7037         \catcode`\%=12\relax
7038     }{%
7039         \markdownInput{#3}%
7040     }%
7041 }%
7042 },
7043 image = {%
7044     \begin{figure}%
7045         \begin{center}%
7046             \includegraphics{#3}%
7047         \end{center}%
7048         \ifx\empty#4\empty\else
7049             \caption{#4}%
7050         \fi
7051     \end{figure}},
7052 ulBegin = {\begin{itemize}},
7053 ulEnd = {\end{itemize}},
7054 olBegin = {\begin{enumerate}},
7055 olItem = {\item{}},
7056 olItemWithNumber = {\item[#1.]},
7057 olEnd = {\end{enumerate}},
7058 dlBegin = {\begin{description}},
7059 dlItem = {\item[#1]},
7060 dlEnd = {\end{description}},
7061 emphasis = {\emph{#1}},
7062 tickedBox = {$\boxtimes$},
7063 halfTickedBox = {$\boxdot$},

```

If identifier attributes appear at the beginning of a section, we make the next heading produce the `\label` macro.

```

7064 headerAttributeContextBegin = {
7065     \markdownSetup{
7066         rendererPrototypes = {
7067             attributeIdentifier = {%
7068                 \begingroup
7069                 \def\next###1{%
7070                     \def#####1#####1{%
7071                         \endgroup
7072                         #####1{#####1}%
7073                     \label{##1}%

```

```

7074         }%
7075     }%
7076     \next\markdownRendererHeadingOne
7077     \next\markdownRendererHeadingTwo
7078     \next\markdownRendererHeadingThree
7079     \next\markdownRendererHeadingFour
7080     \next\markdownRendererHeadingFive
7081     \next\markdownRendererHeadingSix
7082 },
7083 },
7084 }%
7085 },
7086 blockQuoteBegin = {\begin{quotation}},
7087 blockQuoteEnd = {\end{quotation}},
7088 inputVerbatim = {\VerbatimInput{#1}},
7089 inputFencedCode = {%
7090     \ifx\relax#2\relax
7091     \VerbatimInput{#1}%
7092     \else
7093     \@ifundefined{minted@code}{%
7094     \@ifundefined{lst@version}{%
7095     \markdownRendererInputFencedCode{#1}{%

```

When the listings package is loaded, use it for syntax highlighting.

```

7096     }{%
7097     \lstinputlisting[language=#2]{#1}%
7098     }%

```

When the minted package is loaded, use it for syntax highlighting. The minted package is preferred over listings.

```

7099     }{%
7100     \inputminted{#2}{#1}%
7101     }%
7102     \fi},
7103 horizontalRule = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
7104 footnote = {\footnote{#1}}

```

Support the nesting of strong emphasis.

```

7105 \def\markdownLATEXStrongEmphasis#1{%
7106     \IfSubStr\@series{b}{\textnormal{#1}}{\textbf{#1}}
7107 \markdownSetup{rendererPrototypes={strongEmphasis={%
7108     \protect\markdownLATEXStrongEmphasis{#1}}}}

```

Support L<sup>A</sup>T<sub>E</sub>X document classes that do not provide chapters.

```

7109 \@ifundefined{chapter}{%
7110     \markdownSetup{rendererPrototypes = {
7111         headingOne = {\section{#1}},
7112         headingTwo = {\subsection{#1}},
7113         headingThree = {\subsubsection{#1}},

```

```

7114     headingFour = {\paragraph{#1}\leavevmode},
7115     headingFive = {\subparagraph{#1}\leavevmode}}
7116 }{%
7117 \markdownSetup{rendererPrototypes = {
7118     headingOne = {\chapter{#1}},
7119     headingTwo = {\section{#1}},
7120     headingThree = {\subsection{#1}},
7121     headingFour = {\subsubsection{#1}},
7122     headingFive = {\paragraph{#1}\leavevmode},
7123     headingSix = {\subparagraph{#1}\leavevmode}}}
7124 }%

```

**3.3.4.1 Tickboxes** If the `taskLists` option is enabled, we will hide bullets in unordered list items with tickboxes.

```

7125 \markdownSetup{
7126     rendererPrototypes = {
7127         ulItem = {%
7128             \futurelet\markdownLaTeXCheckbox\markdownLaTeXUListItem
7129         },
7130     },
7131 }
7132 \def\markdownLaTeXUListItem{%
7133     \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
7134         \item[\markdownLaTeXCheckbox]%
7135         \expandafter\@gobble
7136     \else
7137         \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
7138             \item[\markdownLaTeXCheckbox]%
7139             \expandafter\expandafter\expandafter\@gobble
7140         \else
7141             \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
7142                 \item[\markdownLaTeXCheckbox]%
7143                 \expandafter\expandafter\expandafter\expandafter
7144                 \expandafter\expandafter\expandafter\@gobble
7145             \else
7146                 \item{}%
7147             \fi
7148         \fi
7149     \fi
7150 }

```

**3.3.4.2 HTML elements** If the `html` option is enabled and we are using `TeX4ht`<sup>9</sup>, we will pass HTML elements to the output HTML document unchanged.

```

7151 \@ifundefined{HCode}{-}{

```

---

<sup>9</sup>See <https://tug.org/tex4ht/>.

```

7152 \markdownSetup{
7153   rendererPrototypes = {
7154     inlineHtmlTag = {%
7155       \ifvmode
7156         \IgnorePar
7157         \EndP
7158       \fi
7159       \HCode{#1}%
7160     },
7161     inputBlockHtmlElement = {%
7162       \ifvmode
7163         \IgnorePar
7164         \fi
7165         \EndP
7166         \special{t4ht*<#1}%
7167         \par
7168         \ShowPar
7169     },
7170   },
7171 }
7172 }

```

**3.3.4.3 Citations** Here is a basic implementation for citations that uses the L<sup>A</sup>T<sub>E</sub>X `\cite` macro. There are also implementations that use the natbib `\citep`, and `\citet` macros, and the BibL<sup>A</sup>T<sub>E</sub>X `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```

7173 \newcount\markdownLaTeXCitationsCounter
7174
7175 % Basic implementation
7176 \RequirePackage{gobble}
7177 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
7178   \advance\markdownLaTeXCitationsCounter by 1\relax
7179   \ifx\relax#4\relax
7180     \ifx\relax#5\relax
7181       \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
7182         \cite{#1#2#6}% Without prenotes and postnotes, just accumulate cites
7183         \expandafter\expandafter\expandafter
7184         \expandafter\expandafter\expandafter\expandafter
7185         \@gobblethree
7186       \fi
7187     \else% Before a postnote (#5), dump the accumulator
7188       \ifx\relax#1\relax\else
7189         \cite{#1}%
7190       \fi
7191       \cite[#5]{#6}%
7192     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax

```



```

7193     \else
7194         \expandafter\expandafter\expandafter
7195         \expandafter\expandafter\expandafter\expandafter
7196         \expandafter\expandafter\expandafter
7197         \expandafter\expandafter\expandafter\expandafter
7198         \markdownLaTeXBasicCitations
7199     \fi
7200     \expandafter\expandafter\expandafter
7201     \expandafter\expandafter\expandafter\expandafter{%
7202     \expandafter\expandafter\expandafter
7203     \expandafter\expandafter\expandafter\expandafter}%
7204     \expandafter\expandafter\expandafter
7205     \expandafter\expandafter\expandafter\expandafter{%
7206     \expandafter\expandafter\expandafter
7207     \expandafter\expandafter\expandafter\expandafter}%
7208     \expandafter\expandafter\expandafter
7209     \@gobblethree
7210 \fi
7211 \else% Before a prenote (#4), dump the accumulator
7212 \ifx\relax#1\relax\else
7213     \cite{#1}%
7214 \fi
7215 \ifnum\markdownLaTeXCitationsCounter>1\relax
7216     \space % Insert a space before the prenote in later citations
7217 \fi
7218 #4-\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
7219 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
7220 \else
7221     \expandafter\expandafter\expandafter
7222     \expandafter\expandafter\expandafter\expandafter
7223     \markdownLaTeXBasicCitations
7224 \fi
7225 \expandafter\expandafter\expandafter{%
7226 \expandafter\expandafter\expandafter}%
7227 \expandafter\expandafter\expandafter{%
7228 \expandafter\expandafter\expandafter}%
7229 \expandafter
7230 \@gobblethree
7231 \fi\markdownLaTeXBasicCitations{#1#2#6},}
7232 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
7233
7234 % Natbib implementation
7235 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
7236 \advance\markdownLaTeXCitationsCounter by 1\relax
7237 \ifx\relax#3\relax
7238     \ifx\relax#4\relax
7239         \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax

```

```

7240     \citep{#1,#5}% Without prenotes and postnotes, just accumulate cites
7241     \expandafter\expandafter\expandafter
7242     \expandafter\expandafter\expandafter\expandafter
7243     \@gobbletwo
7244     \fi
7245 \else% Before a postnote (#4), dump the accumulator
7246     \ifx\relax#1\relax\else
7247         \citep{#1}%
7248     \fi
7249     \citep[] [#4]{#5}%
7250     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
7251     \else
7252         \expandafter\expandafter\expandafter
7253         \expandafter\expandafter\expandafter\expandafter
7254         \expandafter\expandafter\expandafter
7255         \expandafter\expandafter\expandafter\expandafter
7256         \markdownLaTeXNatbibCitations
7257     \fi
7258     \expandafter\expandafter\expandafter
7259     \expandafter\expandafter\expandafter\expandafter{%
7260     \expandafter\expandafter\expandafter
7261     \expandafter\expandafter\expandafter\expandafter}%
7262     \expandafter\expandafter\expandafter
7263     \@gobbletwo
7264     \fi
7265 \else% Before a prenote (#3), dump the accumulator
7266     \ifx\relax#1\relax\relax\else
7267         \citep{#1}%
7268     \fi
7269     \citep[#3] [#4]{#5}%
7270     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
7271     \else
7272         \expandafter\expandafter\expandafter
7273         \expandafter\expandafter\expandafter\expandafter
7274         \markdownLaTeXNatbibCitations
7275     \fi
7276     \expandafter\expandafter\expandafter{%
7277     \expandafter\expandafter\expandafter}%
7278     \expandafter
7279     \@gobbletwo
7280 \fi\markdownLaTeXNatbibCitations{#1,#5}}
7281 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
7282 \advance\markdownLaTeXCitationsCounter by 1\relax
7283 \ifx\relax#3\relax
7284     \ifx\relax#4\relax
7285         \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
7286         \citet{#1,#5}% Without prenotes and postnotes, just accumulate cites

```

```

7287     \expandafter\expandafter\expandafter
7288     \expandafter\expandafter\expandafter\expandafter
7289     \@gobbletwo
7290     \fi
7291 \else% After a prenote or a postnote, dump the accumulator
7292     \ifx\relax#1\relax\else
7293         \citet{#1}%
7294     \fi
7295     , \citet[#3][#4]{#5}%
7296     \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
7297         ,
7298     \else
7299         \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
7300             ,
7301         \fi
7302     \fi
7303     \expandafter\expandafter\expandafter
7304     \expandafter\expandafter\expandafter\expandafter
7305     \markdownLaTeXNatbibTextCitations
7306     \expandafter\expandafter\expandafter
7307     \expandafter\expandafter\expandafter\expandafter{%
7308     \expandafter\expandafter\expandafter
7309     \expandafter\expandafter\expandafter\expandafter}%
7310     \expandafter\expandafter\expandafter
7311     \@gobbletwo
7312     \fi
7313 \else% After a prenote or a postnote, dump the accumulator
7314     \ifx\relax#1\relax\relax\else
7315         \citet{#1}%
7316     \fi
7317     , \citet[#3][#4]{#5}%
7318     \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
7319         ,
7320     \else
7321         \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
7322             ,
7323         \fi
7324     \fi
7325     \expandafter\expandafter\expandafter
7326     \markdownLaTeXNatbibTextCitations
7327     \expandafter\expandafter\expandafter{%
7328     \expandafter\expandafter\expandafter}%
7329     \expandafter
7330     \@gobbletwo
7331     \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
7332
7333 % BibLaTeX implementation

```

```

7334 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
7335   \advance\markdownLaTeXCitationsCounter by 1\relax
7336   \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
7337     \autocites#1[#3][#4]{#5}%
7338     \expandafter\@gobbletwo
7339   \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}
7340 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
7341   \advance\markdownLaTeXCitationsCounter by 1\relax
7342   \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
7343     \textcites#1[#3][#4]{#5}%
7344     \expandafter\@gobbletwo
7345   \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}
7346
7347 \markdownSetup{rendererPrototypes = {
7348   cite = {%
7349     \markdownLaTeXCitationsCounter=1%
7350     \def\markdownLaTeXCitationsTotal{#1}%
7351     \@ifundefined{autocites}{%
7352       \@ifundefined{citep}{%
7353         \expandafter\expandafter\expandafter
7354         \markdownLaTeXBasicCitations
7355         \expandafter\expandafter\expandafter{%
7356         \expandafter\expandafter\expandafter}%
7357         \expandafter\expandafter\expandafter{%
7358         \expandafter\expandafter\expandafter}%
7359       }{%
7360         \expandafter\expandafter\expandafter
7361         \markdownLaTeXNatbibCitations
7362         \expandafter\expandafter\expandafter{%
7363         \expandafter\expandafter\expandafter}%
7364       }%
7365     }{%
7366       \expandafter\expandafter\expandafter
7367       \markdownLaTeXBibLaTeXCitations
7368       \expandafter{\expandafter}%
7369     }},
7370   textCite = {%
7371     \markdownLaTeXCitationsCounter=1%
7372     \def\markdownLaTeXCitationsTotal{#1}%
7373     \@ifundefined{autocites}{%
7374       \@ifundefined{citep}{%
7375         \expandafter\expandafter\expandafter
7376         \markdownLaTeXBasicTextCitations
7377         \expandafter\expandafter\expandafter{%
7378         \expandafter\expandafter\expandafter}%
7379         \expandafter\expandafter\expandafter{%
7380         \expandafter\expandafter\expandafter}%

```

```

7381     }{%
7382     \expandafter\expandafter\expandafter
7383     \markdownLaTeXNatbibTextCitations
7384     \expandafter\expandafter\expandafter{%
7385     \expandafter\expandafter\expandafter}%
7386     }%
7387   }{%
7388   \expandafter\expandafter\expandafter
7389   \markdownLaTeXBibLaTeXTextCitations
7390   \expandafter{\expandafter}%
7391   }}}

```

**3.3.4.4 Links** Before consuming the parameters for the hyperlink renderer, we change the category code of the hash sign (#) to other, so that it cannot be mistaken for a parameter character.

```

7392 \RequirePackage{url}
7393 \RequirePackage{expl3}
7394 \ExplSyntaxOn
7395 \def\markdownRendererLinkPrototype{
7396   \begingroup
7397   \catcode`\#=12
7398   \def\next##1##2##3##4{
7399     \endgroup

```

If the URL begins with a hash sign, then we assume that it is a relative reference. Otherwise, we assume that it is an absolute URL.

```

7400   \tl_set:Nx
7401   \l_tmpa_tl
7402   { \str_range:nnn { ##3 } { 1 } { 1 } }
7403   \str_if_eq:NNTF
7404   \l_tmpa_tl
7405   \c_hash_str
7406   {
7407     \exp_args:No
7408     \markdownLaTeXRendererRelativeLink
7409     { \str_range:nnn { ##3 } { 2 } { -1 } }
7410   }{
7411     \markdownLaTeXRendererAbsoluteLink { ##1 } { ##2 } { ##3 } { ##4 }
7412   }
7413 }
7414 \next
7415 }
7416 \ExplSyntaxOff
7417 \def\markdownLaTeXRendererAbsoluteLink#1#2#3#4{%
7418   #1\footnote{\ifx\empty#4\empty\else#4: \fi\texttt<\url{#3}\texttt>}}
7419 \def\markdownLaTeXRendererRelativeLink#1{%
7420   \ref{#1}}

```

**3.3.4.5 Tables** Here is a basic implementation of tables. If the booktabs package is loaded, then it is used to produce horizontal lines.

```

7421 \newcount\markdownLaTeXRowCounter
7422 \newcount\markdownLaTeXRowTotal
7423 \newcount\markdownLaTeXColumnCounter
7424 \newcount\markdownLaTeXColumnTotal
7425 \newtoks\markdownLaTeXTable
7426 \newtoks\markdownLaTeXTableAlignment
7427 \newtoks\markdownLaTeXTableEnd
7428 \AtBeginDocument{%
7429   \@ifpackageloaded{booktabs}{%
7430     \def\markdownLaTeXTopRule{\toprule}%
7431     \def\markdownLaTeXMidRule{\midrule}%
7432     \def\markdownLaTeXBottomRule{\bottomrule}%
7433   }{%
7434     \def\markdownLaTeXTopRule{\hline}%
7435     \def\markdownLaTeXMidRule{\hline}%
7436     \def\markdownLaTeXBottomRule{\hline}%
7437   }%
7438 }
7439 \markdownSetup{rendererPrototypes={
7440   table = {%
7441     \markdownLaTeXTable={}%
7442     \markdownLaTeXTableAlignment={}%
7443     \markdownLaTeXTableEnd={%
7444       \markdownLaTeXBottomRule
7445     \end{tabular}}}%
7446   \ifx\empty#1\empty\else
7447     \addto@hook\markdownLaTeXTable{%
7448       \begin{table}
7449       \centering}%
7450     \addto@hook\markdownLaTeXTableEnd{%
7451       \caption{#1}
7452       \end{table}}}%
7453   \fi
7454   \addto@hook\markdownLaTeXTable{\begin{tabular}}%
7455   \markdownLaTeXRowCounter=0%
7456   \markdownLaTeXRowTotal=#2%
7457   \markdownLaTeXColumnTotal=#3%
7458   \markdownLaTeXRenderTableRow
7459 }
7460 }}
7461 \def\markdownLaTeXRenderTableRow#1{%
7462   \markdownLaTeXColumnCounter=0%
7463   \ifnum\markdownLaTeXRowCounter=0\relax
7464     \markdownLaTeXReadAlignments#1%
7465     \markdownLaTeXTable=\expandafter\expandafter\expandafter{%

```

```

7466     \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
7467     \the\markdownLaTeXTableAlignment}}}%
7468     \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
7469 \else
7470     \markdownLaTeXRenderTableCell#1%
7471 \fi
7472 \ifnum\markdownLaTeXRowCount=1\relax
7473     \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
7474 \fi
7475 \advance\markdownLaTeXRowCount by 1\relax
7476 \ifnum\markdownLaTeXRowCount>\markdownLaTeXRowTotal\relax
7477     \the\markdownLaTeXTable
7478     \the\markdownLaTeXTableEnd
7479     \expandafter\@gobble
7480 \fi\markdownLaTeXRenderTableRow}
7481 \def\markdownLaTeXReadAlignments#1{%
7482     \advance\markdownLaTeXColumnCounter by 1\relax
7483     \if#1d%
7484         \addto@hook\markdownLaTeXTableAlignment{1}%
7485     \else
7486         \addto@hook\markdownLaTeXTableAlignment{#1}%
7487     \fi
7488     \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
7489         \expandafter\@gobble
7490     \fi\markdownLaTeXReadAlignments}
7491 \def\markdownLaTeXRenderTableCell#1{%
7492     \advance\markdownLaTeXColumnCounter by 1\relax
7493     \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
7494         \addto@hook\markdownLaTeXTable{#1&}%
7495     \else
7496         \addto@hook\markdownLaTeXTable{#1\\}%
7497         \expandafter\@gobble
7498     \fi\markdownLaTeXRenderTableCell}
7499 \fi

```

**3.3.4.6 YAML Metadata** The default setup of YAML metadata will invoke the `\title`, `\author`, and `\date` macros when scalar values for keys that correspond to the `title`, `author`, and `date` relative wildcards are encountered, respectively.

```

7500 \ExplSyntaxOn
7501 \keys_define:nn
7502 { markdown/jekyllData }
7503 {
7504     author .code:n = { \author{#1} },
7505     date   .code:n = { \date{#1}   },
7506     title  .code:n = { \title{#1}  },
7507 }

```

To complement the default setup of our key-values, we will use the `\maketitle` macro to typeset the title page of a document at the end of YAML metadata. If we are in the preamble, we will wait macro until after the beginning of the document. Otherwise, we will use the `\maketitle` macro straight away.

```

7508 % TODO: Remove the command definition in TeX Live 2021.
7509 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
7510 \markdownSetup{
7511   rendererPrototypes = {
7512     jekyllDataEnd = {
7513 %       TODO: Remove the else branch in TeX Live 2021.
7514       \IfFormatAtLeastTF
7515         { 2020-10-01 }
7516         { \AddToHook{begindocument/end}{\maketitle} }
7517         {
7518           \ifx\@onlypreamble\@notprerr
7519             % We are in the document
7520             \maketitle
7521           \else
7522             % We are in the preamble
7523             \RequirePackage{etoolbox}
7524             \AfterEndPreamble{\maketitle}
7525           \fi
7526         }
7527     },
7528   },
7529 }
7530 \ExplSyntaxOff

```

### 3.3.5 Miscellanea

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `inputenc` package. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the `filecontents` package.

```

7531 \newcommand\markdownMakeOther{%
7532   \count0=128\relax
7533   \loop
7534     \catcode\count0=11\relax
7535     \advance\count0 by 1\relax
7536   \ifnum\count0<256\repeat}%

```

## 3.4 ConTeXt Implementation

The ConTeXt implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV ConTeXt formats *seem* to implement (the



documentation is scarce) the majority of the plain  $\TeX$  format required by the plain  $\TeX$  implementation. As a consequence, we can directly reuse the existing plain  $\TeX$  implementation after supplying the missing plain  $\TeX$  macros.

The Con $\TeX$ t implementation redefines the plain  $\TeX$  logging macros (see Section 3.2.1) to use the Con $\TeX$ t `\writestatus` macro.

```
7537 \def\markdownInfo#1{\writestatus{markdown}{#1.}}%
7538 \def\markdownWarning#1{\writestatus{markdown\space warn}{#1.}}%
7539 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
7540 \do\#\do\~\do\_ \do\% \do\~}%
7541 \input markdown/markdown
```

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `\enableregime` macro. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the filecontents  $\LaTeX$  package.

```
7542 \def\markdownMakeOther{%
7543 \count0=128\relax
7544 \loop
7545 \catcode\count0=11\relax
7546 \advance\count0 by 1\relax
7547 \ifnum\count0<256\repeat
```

On top of that, make the pipe character (`|`) inactive during the scanning. This is necessary, since the character is active in Con $\TeX$ t.

```
7548 \catcode`|=12}%
```

### 3.4.1 Typesetting Markdown

The `\startmarkdown` and `\stopmarkdown` macros are implemented using the `\markdownReadAndConvert` macro.

In Knuth’s  $\TeX$ , trailing spaces are removed very early on when a line is being put to the input buffer. [11, sec. 31]. According to Eijkhout [12, sec. 2.2], this is because “these spaces are hard to see in an editor”. At the moment, there is no option to suppress this behavior in (Lua) $\TeX$ , but Con $\TeX$ t MkIV funnels all input through its own input handler. This makes it possible to suppress the removal of trailing spaces in Con $\TeX$ t MkIV and therefore to insert hard line breaks into markdown text.

```
7549 \ifx\startluacode\undefined % MkII
7550 \begingroup
7551 \catcode`|=0%
7552 \catcode`\|=12%
7553 |gdef|startmarkdown{%
7554 |markdownReadAndConvert{\stopmarkdown}%
7555 |stopmarkdown}%
7556 |gdef|stopmarkdown{%
7557 |markdownEnd}%
```

```

7558 |endgroup
7559 \else % MkIV
7560 \startluacode
7561     document.markdown_buffering = false
7562     local function preserve_trailing_spaces(line)
7563         if document.markdown_buffering then
7564             line = line:gsub("[ \t][ \t]$", "\t\t")
7565         end
7566         return line
7567     end
7568     resolvers.installinputlinehandler(preserve_trailing_spaces)
7569 \stopluacode
7570 \beginingroup
7571     \catcode`\|=0%
7572     \catcode`\|=12%
7573     |gdef|startmarkdown{%
7574         |ctxlua{document.markdown_buffering = true}%
7575         |markdownReadAndConvert{\stopmarkdown}%
7576                                     {|stopmarkdown}}%
7577     |gdef|stopmarkdown{%
7578         |ctxlua{document.markdown_buffering = false}%
7579         |markdownEnd}%
7580 |endgroup
7581 \fi

```

### 3.4.2 Token Renderer Prototypes

The following configuration should be considered placeholder.

```

7582 \def\markdownRendererLineBreakPrototype{\blank}%
7583 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
7584 \def\markdownRendererRightBracePrototype{\textbraceright}%
7585 \def\markdownRendererDollarSignPrototype{\textdollar}%
7586 \def\markdownRendererPercentSignPrototype{\percent}%
7587 \def\markdownRendererUnderscorePrototype{\textunderscore}%
7588 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
7589 \def\markdownRendererBackslashPrototype{\textbackslash}%
7590 \def\markdownRendererTildePrototype{\textasciitilde}%
7591 \def\markdownRendererPipePrototype{\char`|}%
7592 \def\markdownRendererLinkPrototype#1#2#3#4{%
7593     \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
7594     \fi\tt<\hyphenatedurl{#3}>}}%
7595 \usemodule[database]
7596 \defineseparatedlist
7597     [MarkdownConTeXtCSV]
7598     [separator={,},
7599     before=\bTABLE,after=\eTABLE,
7600     first=\bTR,last=\eTR,

```

```

7601 left=\bTD,right=\eTD]
7602 \def\markdownConTeXtCSV{csv}
7603 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
7604 \def\markdownConTeXtCSV@arg{#1}%
7605 \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
7606 \placetable [] [tab:#1]{#4}{%
7607 \processeparatedfile [MarkdownConTeXtCSV] [#3]}%
7608 \else
7609 \markdownInput{#3}%
7610 \fi}%
7611 \def\markdownRendererImagePrototype#1#2#3#4{%
7612 \placefigure [] []{#4}{\externalfigure [#3]}}%
7613 \def\markdownRendererUlBeginPrototype{\startitemize}%
7614 \def\markdownRendererUlBeginTightPrototype{\startitemize [packed]}%
7615 \def\markdownRendererUlItemPrototype{\item}%
7616 \def\markdownRendererUlEndPrototype{\stopitemize}%
7617 \def\markdownRendererUlEndTightPrototype{\stopitemize}%
7618 \def\markdownRendererOlBeginPrototype{\startitemize [n]}%
7619 \def\markdownRendererOlBeginTightPrototype{\startitemize [packed,n]}%
7620 \def\markdownRendererOlItemPrototype{\item}%
7621 \def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
7622 \def\markdownRendererOlEndPrototype{\stopitemize}%
7623 \def\markdownRendererOlEndTightPrototype{\stopitemize}%
7624 \definedescription
7625 [MarkdownConTeXtDlItemPrototype]
7626 [location=hanging,
7627 margin=standard,
7628 headstyle=bold]%
7629 \definestartstop
7630 [MarkdownConTeXtDlPrototype]
7631 [before=\blank,
7632 after=\blank]%
7633 \definestartstop
7634 [MarkdownConTeXtDlTightPrototype]
7635 [before=\blank\startpacked,
7636 after=\stoppacked\blank]%
7637 \def\markdownRendererDlBeginPrototype{%
7638 \startMarkdownConTeXtDlPrototype}%
7639 \def\markdownRendererDlBeginTightPrototype{%
7640 \startMarkdownConTeXtDlTightPrototype}%
7641 \def\markdownRendererDlItemPrototype#1{%
7642 \startMarkdownConTeXtDlItemPrototype{#1}}%
7643 \def\markdownRendererDlItemEndPrototype{%
7644 \stopMarkdownConTeXtDlItemPrototype}%
7645 \def\markdownRendererDlEndPrototype{%
7646 \stopMarkdownConTeXtDlPrototype}%
7647 \def\markdownRendererDlEndTightPrototype{%

```

```

7648 \stopMarkdownConTeXtDlTightPrototype}%
7649 \def\markdownRendererEmphasisPrototype#1{\em#1}%
7650 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
7651 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
7652 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
7653 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%
7654 \def\markdownRendererInputFencedCodePrototype#1#2{%
7655   \ifx\relax#2\relax
7656     \typefile{#1}%
7657   \else

```

The code fence infosting is used as a name from the ConT<sub>E</sub>Xt `\definetyping` macro. This allows the user to set up code highlighting mapping as follows:

```

\definetyping [latex]
\setuptyping [latex] [option=TEX]

\starttext
  \startmarkdown
~~~ latex
\documentclass{article}
\begin{document}
  Hello world!
\end{document}
~~~
  \stopmarkdown
\stoptext

```

```

7658   \typefile[#2] []{#1}%
7659   \fi}%
7660 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
7661 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
7662 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
7663 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
7664 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
7665 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
7666 \def\markdownRendererHorizontalRulePrototype{%
7667   \blackrule[height=1pt, width=\hsize]}%
7668 \def\markdownRendererFootnotePrototype#1{\footnote{#1}}%
7669 \stopmodule\protect

```

There is a basic implementation of tables.

```

7670 \newcount\markdownConTeXtRowCounter
7671 \newcount\markdownConTeXtRowTotal
7672 \newcount\markdownConTeXtColumnCounter
7673 \newcount\markdownConTeXtColumnTotal

```

```

7674 \newtoks\markdownConTeXtTable
7675 \newtoks\markdownConTeXtTableFloat
7676 \def\markdownRendererTablePrototype#1#2#3{%
7677   \markdownConTeXtTable={}%
7678   \ifx\empty#1\empty
7679     \markdownConTeXtTableFloat={%
7680       \the\markdownConTeXtTable}%
7681   \else
7682     \markdownConTeXtTableFloat={%
7683       \placetable{#1}{\the\markdownConTeXtTable}}%
7684   \fi
7685   \begingroup
7686   \setupTABLE[r][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
7687   \setupTABLE[c][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
7688   \setupTABLE[r][1][topframe=on, bottomframe=on]
7689   \setupTABLE[r][#1][bottomframe=on]
7690   \markdownConTeXtRowCounter=0%
7691   \markdownConTeXtRowTotal=#2%
7692   \markdownConTeXtColumnTotal=#3%
7693   \markdownConTeXtRenderTableRow}
7694 \def\markdownConTeXtRenderTableRow#1{%
7695   \markdownConTeXtColumnCounter=0%
7696   \ifnum\markdownConTeXtRowCounter=0\relax
7697     \markdownConTeXtReadAlignments#1%
7698     \markdownConTeXtTable={\bTABLE}%
7699   \else
7700     \markdownConTeXtTable=\expandafter{%
7701       \the\markdownConTeXtTable\bTR}%
7702     \markdownConTeXtRenderTableCell#1%
7703     \markdownConTeXtTable=\expandafter{%
7704       \the\markdownConTeXtTable\eTR}%
7705   \fi
7706   \advance\markdownConTeXtRowCounter by 1\relax
7707   \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
7708     \markdownConTeXtTable=\expandafter{%
7709       \the\markdownConTeXtTable\eTABLE}%
7710     \the\markdownConTeXtTableFloat
7711     \endgroup
7712     \expandafter\gobbleoneargument
7713   \fi\markdownConTeXtRenderTableRow}
7714 \def\markdownConTeXtReadAlignments#1{%
7715   \advance\markdownConTeXtColumnCounter by 1\relax
7716   \if#1d%
7717     \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
7718   \fi\if#1l%
7719     \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
7720   \fi\if#1c%

```

```

7721 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
7722 \fi\if#1r%
7723 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
7724 \fi
7725 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
7726 \expandafter\gobbleoneargument
7727 \fi\markdownConTeXtReadAlignments}
7728 \def\markdownConTeXtRenderTableCell#1{%
7729 \advance\markdownConTeXtColumnCounter by 1\relax
7730 \markdownConTeXtTable=\expandafter{%
7731 \the\markdownConTeXtTable\bTD#1\eTD}%
7732 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
7733 \expandafter\gobbleoneargument
7734 \fi\markdownConTeXtRenderTableCell}
7735 \def\markdownRendererTickedBox{\$ \boxtimes$}
7736 \def\markdownRendererHalfTickedBox{\$ \boxdot$}
7737 \def\markdownRendererUntickedBox{\$ \square$}

```

## References

- [1] LuaTeX development team. *LuaTeX reference manual*. Feb. 2017. URL: <http://www.luatex.org/svn/trunk/manual/luatex.pdf> (visited on 01/08/2018).
- [2] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: <https://www.muni.cz/en/research/projects/32984> (visited on 02/19/2018).
- [3] Anton Sotkov. *File transclusion syntax for Markdown*. Jan. 19, 2017. URL: <https://github.com/iainc/Markdown-Content-Blocks> (visited on 01/08/2018).
- [4] Donald Ervin Knuth. *The TeXbook*. 3rd ed. Vol. A. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. ix, 479. ISBN: 0-201-13447-0.
- [5] Frank Mittelbach. *The doc and shortvrb Packages*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/doc.pdf> (visited on 02/19/2018).
- [6] Till Tantau, Joseph Wright, and Vedran Miletić. *The Beamer class*. Feb. 10, 2021. URL: <https://mirrors.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf> (visited on 02/11/2021).
- [7] Vít Novotný. *L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> no longer keys packages by pathnames*. Feb. 20, 2021. URL: <https://github.com/latex3/latex2e/issues/510> (visited on 02/21/2021).
- [8] Geoffrey M. Poore. *The minted Package. Highlighted source code in L<sup>A</sup>T<sub>E</sub>X*. July 19, 2017. URL: <https://mirrors.ctan.org/macros/latex/contrib/minted/minted.pdf> (visited on 09/01/2020).

- [9] Roberto Ierusalimsky. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.
- [10] Johannes Braams et al. *The L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> Sources*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/source2e.pdf> (visited on 01/08/2018).
- [11] Donald Ervin Knuth. *T<sub>E</sub>X: The Program*. Vol. B. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. xvi, 594. ISBN: 0-201-13437-7.
- [12] Victor Eijkhout. *T<sub>E</sub>X by Topic. A T<sub>E</sub>Xnician's Reference*. Wokingham, England: Addison-Wesley, Feb. 1, 1992. 307 pp. ISBN: 0-201-56882-0.

## Index

<code>\author</code>	223
<code>\autocites</code>	216
<code>blankBeforeBlockquote</code>	9
<code>blankBeforeCodeFence</code>	9
<code>blankBeforeHeading</code>	10
<code>breakableBlockquotes</code>	10
<code>cacheDir</code>	8, 14, 36, 37, 186
<code>citationNbsps</code>	11
<code>citations</code>	11, 62, 63
<code>\cite</code>	216
<code>\citep</code>	216
<code>\citet</code>	216
<code>codeSpans</code>	12
<code>compactdesc</code>	4
<code>compactenum</code>	4
<code>compactitem</code>	4
<code>contentBlocks</code>	12
<code>contentBlocksLanguageMap</code>	13, 138
<code>convert</code>	198
<code>\csvautotabular</code>	4
<code>\date</code>	223
<code>defaultOptions</code>	7, 33, 134, 166
<code>\definotyping</code>	228
<code>definitionLists</code>	13, 51
<code>\directlua</code>	3, 36, 204
<code>eagerCache</code>	14

<code>\enableregime</code>	225
<code>\endmarkdown</code>	71
<code>entities.char_entity</code>	133
<code>entities.dec_entity</code>	133
<code>entities.hex_entity</code>	133
<code>escape</code>	136, 136, 137
<code>escape_citation</code>	136, 136
<code>escape_minimal</code>	136, 136
<code>escape_uri</code>	136, 136
<code>escaped_chars</code>	136, 136
<code>escaped_citation_chars</code>	135, 136
<code>escaped_minimal_strings</code>	135, 136
<code>escaped_uri_chars</code>	135, 136
<code>expandtabs</code>	167
<code>expectJekyllData</code>	15
<code>fencedCode</code>	16, 56, 208
<code>\filecontents</code>	199
<code>finalizeCache</code>	8, 14, 17, 19, 36, 187
<code>footnotes</code>	18, 62
<code>frozenCacheCounter</code>	19, 187, 189
<code>frozenCacheFileName</code>	8, 17, 37, 187
<code>hardLineBreaks</code>	19
<code>hashEnumerators</code>	20
<code>headerAttributes</code>	20, 25, 65, 66
<code>html</code>	21, 63, 64, 215
<code>hybrid</code>	21, 29, 30, 42, 76, 136, 137, 168, 205
<code>\includegraphics</code>	4
<code>inlineFootnotes</code>	22
<code>\input</code>	34, 84, 201, 204
<code>isdir</code>	3
<code>iterlines</code>	166
<code>jekyllData</code>	3, 15, 22, 56–59
<code>\jobname</code>	37
<code>\label</code>	213
<code>languages_json</code>	138, 138
<code>larsers</code>	167
<code>\maketitle</code>	224



<code>\markdown</code>	71
<code>markdown</code>	70, 71, 207
<code>markdown*</code>	4, 70, 71, 72, 207
<code>\markdown_jekyll_data_concatenate_address:NN</code>	192
<code>\markdown_jekyll_data_pop:</code>	193
<code>\markdown_jekyll_data_push:nN</code>	193
<code>\markdown_jekyll_data_push_address_segment:n</code>	191
<code>\markdown_jekyll_data_set_keyval:Nn</code>	193
<code>\markdown_jekyll_data_set_keyvals:nn</code>	193
<code>\markdown_jekyll_data_update_address_tls:</code>	192
<code>\markdownBegin</code>	35, 35, 36, 69, 71, 84
<code>\markdownEnd</code>	35, 35, 36, 69, 71, 84
<code>\markdownError</code>	69, 69
<code>\markdownExecute</code>	202
<code>\markdownExecuteDirect</code>	202, 202
<code>\markdownExecuteShellEscape</code>	202, 202
<code>\markdownFrozenCacheCounter</code>	189, 189, 205, 206
<code>\markdownIfOption</code>	198
<code>\markdownIfSnippetExists</code>	72
<code>\markdownInfo</code>	69
<code>\markdownInput</code>	35, 36, 70–72, 205, 207
<code>\markdownInputFileStream</code>	199
<code>\markdownInputPlainTeX</code>	207
<code>\markdownLuaExecute</code>	202, 203, 204, 205
<code>\markdownLuaOptions</code>	195, 198
<code>\markdownMakeOther</code>	69, 224, 225
<code>\markdownMode</code>	3, 37, 70, 70, 201, 204
<code>\markdownOptionCacheDir</code>	3, 37, 81, 198, 210
<code>\markdownOptionErrorTempFileName</code>	37, 203
<code>\markdownOptionFinalizeCache</code>	36, 36, 80, 189
<code>\markdownOptionFrozenCache</code>	8, 17, 36, 36, 75, 76, 80, 189, 208, 210
<code>\markdownOptionFrozenCacheFileName</code>	36, 37
<code>\markdownOptionHelperScriptFileName</code>	36, 37, 38, 203, 204
<code>\markdownOptionHybrid</code>	81
<code>\markdownOptionInputTempFileName</code>	37, 199–201
<code>\markdownOptionOutputDir</code>	37
<code>\markdownOptionOutputTempFileName</code>	37, 204
<code>\markdownOptionSmartEllipses</code>	81
<code>\markdownOptionStripPercentSigns</code>	39, 199, 200
<code>\markdownOutputFileStream</code>	199
<code>\markdownPrepare</code>	197
<code>\markdownPrepareLuaOptions</code>	195

<code>\markdownReadAndConvert</code>	69, 199, 207, 225
<code>\markdownReadAndConvertProcessLine</code>	200, 201
<code>\markdownReadAndConvertStripPercentSigns</code>	200
<code>\markdownReadAndConvertTab</code>	199
<code>\markdownRendererAttributeClassName</code>	65
<code>\markdownRendererAttributeIdentifier</code>	65
<code>\markdownRendererAttributeKeyValue</code>	65
<code>\markdownRendererBlockHtmlCommentBegin</code>	64
<code>\markdownRendererBlockHtmlCommentEnd</code>	64
<code>\markdownRendererBlockQuoteBegin</code>	55
<code>\markdownRendererBlockQuoteEnd</code>	55
<code>\markdownRendererCite</code>	62, 63
<code>\markdownRendererCodeSpan</code>	45
<code>\markdownRendererCodeSpanPrototype</code>	83
<code>\markdownRendererContentBlock</code>	46, 46
<code>\markdownRendererContentBlockCode</code>	46
<code>\markdownRendererContentBlockOnlineImage</code>	46
<code>\markdownRendererDlBegin</code>	51
<code>\markdownRendererDlBeginTight</code>	52
<code>\markdownRendererDlDefinitionBegin</code>	53
<code>\markdownRendererDlDefinitionEnd</code>	53
<code>\markdownRendererDlEnd</code>	53
<code>\markdownRendererDlEndTight</code>	54
<code>\markdownRendererDlItem</code>	52
<code>\markdownRendererDlItemEnd</code>	52
<code>\markdownRendererDocumentBegin</code>	40
<code>\markdownRendererDocumentEnd</code>	40
<code>\markdownRendererEllipsis</code>	26, 41
<code>\markdownRendererEmphasis</code>	54, 82
<code>\markdownRendererFootnote</code>	62
<code>\markdownRendererHalfTickedBox</code>	39
<code>\markdownRendererHeaderAttributeContextBegin</code>	66
<code>\markdownRendererHeaderAttributeContextEnd</code>	66
<code>\markdownRendererHeadingFive</code>	61
<code>\markdownRendererHeadingFour</code>	60
<code>\markdownRendererHeadingOne</code>	60
<code>\markdownRendererHeadingSix</code>	61
<code>\markdownRendererHeadingThree</code>	60
<code>\markdownRendererHeadingTwo</code>	60
<code>\markdownRendererHorizontalRule</code>	61
<code>\markdownRendererImage</code>	45
<code>\markdownRendererImagePrototype</code>	83

<code>\markdownRendererInlineHtmlComment</code>	63
<code>\markdownRendererInlineHtmlTag</code>	64
<code>\markdownRendererInputBlockHtmlElement</code>	64
<code>\markdownRendererInputFencedCode</code>	56
<code>\markdownRendererInputVerbatim</code>	55
<code>\markdownRendererInterblockSeparator</code>	41
<code>\markdownRendererJekyllDataBegin</code>	56
<code>\markdownRendererJekyllDataBoolean</code>	58
<code>\markdownRendererJekyllDataEmpty</code>	59
<code>\markdownRendererJekyllDataEnd</code>	56
<code>\markdownRendererJekyllDataMappingBegin</code>	57
<code>\markdownRendererJekyllDataMappingEnd</code>	57
<code>\markdownRendererJekyllDataNumber</code>	58
<code>\markdownRendererJekyllDataSequenceBegin</code>	57
<code>\markdownRendererJekyllDataSequenceEnd</code>	58
<code>\markdownRendererJekyllDataString</code>	59
<code>\markdownRendererLineBreak</code>	41
<code>\markdownRendererLink</code>	45, 82
<code>\markdownRendererNbsp</code>	42
<code>\markdownRendererOlBegin</code>	49
<code>\markdownRendererOlBeginTight</code>	49
<code>\markdownRendererOlEnd</code>	51
<code>\markdownRendererOlEndTight</code>	51
<code>\markdownRendererOlItem</code>	26, 50
<code>\markdownRendererOlItemEnd</code>	50
<code>\markdownRendererOlItemWithNumber</code>	26, 50
<code>\markdownRendererStrongEmphasis</code>	54
<code>\markdownRendererTable</code>	63
<code>\markdownRendererTextCite</code>	62
<code>\markdownRendererTickedBox</code>	39
<code>\markdownRendererUlBegin</code>	47
<code>\markdownRendererUlBeginTight</code>	47
<code>\markdownRendererUlEnd</code>	48
<code>\markdownRendererUlEndTight</code>	49
<code>\markdownRendererUlItem</code>	48
<code>\markdownRendererUlItemEnd</code>	48
<code>\markdownRendererUntickedBox</code>	39
<code>\markdownSetup</code>	4, 72, 72, 206, 210
<code>\markdownSetupSnippet</code>	72, 72
<code>\markdownWarning</code>	69
<code>new</code>	6, 187

<code>normalize_tag</code>	166
<code>os.execute</code>	70, 202
<code>\PackageError</code>	206
<code>\PackageInfo</code>	206
<code>\PackageWarning</code>	206
<code>parsers</code>	149
<code>parsers.commented_line</code>	151
<code>\pdfshellescape</code>	202
<code>pipeTables</code>	6, 23, 27, 63
<code>preserveTabs</code>	24, 27, 167
<code>print</code>	203, 204
<code>reader</code>	85, 149, 166, 167
<code>reader-&gt;convert</code>	186, 187
<code>reader.new</code>	166, 166
<code>relativeReferences</code>	24
<code>\shellescape</code>	202
<code>shiftHeadings</code>	6, 25
<code>slice</code>	6, 20, 25, 134
<code>smartEllipses</code>	26, 41
<code>\startmarkdown</code>	84, 84, 225
<code>startNumber</code>	26, 50
<code>status.shell_escape</code>	202
<code>\stopmarkdown</code>	84, 84, 225
<code>stripIndent</code>	27, 167
<code>tableCaptions</code>	6, 27
<code>taskLists</code>	28, 39, 215
<code>\tex.print</code>	204
<code>tex.print</code>	203
<code>texComments</code>	29, 168
<code>\textcites</code>	216
<code>tightLists</code>	29, 47, 49, 51, 52, 54, 211
<code>\title</code>	223
<code>underscores</code>	30
<code>\url</code>	4
<code>\usepackage</code>	70, 73
<code>\usetheme</code>	73
<code>util.cache</code>	85

util.err	85
util.escaper	88
util.expand_tabs_in_line	86
util.flatten	87
util.intersperse	88
util.map	88
util.pathname	89
util.rope_last	87
util.rope_to_string	87
util.table_copy	86
util.walk	86, 87
\VerbatimInput	4
writer	85, 133
writer->active_attributes	145
writer->active_headings	145
writer->block_html_comment	140
writer->block_html_element	141
writer->blockquote	142
writer->bulletlist	139
writer->citation	136
writer->citations	148
writer->code	137
writer->codeFence	142
writer->contentblock	139
writer->defer_call	149, 149
writer->definitionlist	141
writer->document	143
writer->ellipsis	135
writer->emphasis	142
writer->escape	137, 137
writer->get_state	148
writer->heading	145
writer->hrule	135
writer->image	138
writer->inline_html_comment	140
writer->inline_html_tag	141
writer->interblocksep	135
writer->is_writing	134, 134
writer->jekyllData	143
writer->linebreak	135

<code>writer-&gt;link</code>	<i>137</i>
<code>writer-&gt;nbsp</code>	<i>134</i>
<code>writer-&gt;note</code>	<i>148</i>
<code>writer-&gt;ollist</code>	<i>140</i>
<code>writer-&gt;pack</code>	<i>135, 187</i>
<code>writer-&gt;paragraph</code>	<i>135</i>
<code>writer-&gt;plain</code>	<i>134</i>
<code>writer-&gt;set_state</code>	<i>148</i>
<code>writer-&gt;slice_begin</code>	<i>134</i>
<code>writer-&gt;slice_end</code>	<i>134</i>
<code>writer-&gt;space</code>	<i>134</i>
<code>writer-&gt;string</code>	<i>136, 137</i>
<code>writer-&gt;strong</code>	<i>142</i>
<code>writer-&gt;suffix</code>	<i>134</i>
<code>writer-&gt;table</code>	<i>137</i>
<code>writer-&gt;checkbox</code>	<i>142</i>
<code>writer-&gt;uri</code>	<i>136</i>
<code>writer-&gt;verbatim</code>	<i>142</i>
<code>writer.new</code>	<i>133, 133</i>
<code>\writestatus</code>	<i>225</i>