

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

Vít Novotný  
witiko@mail.muni.cz

Version 2.16.0-5-g5bb83fb  
2022/08/26

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>	<b>3</b>	<b>Implementation</b>	<b>94</b>
1.1	Requirements . . . . .	2	3.1	Lua Implementation . . .	94
1.2	Feedback . . . . .	5	3.2	Plain T <sub>E</sub> X Implementation	211
1.3	Acknowledgements . . . .	5	3.3	L <sup>A</sup> T <sub>E</sub> X Implementation . .	227
<b>2</b>	<b>Interfaces</b>	<b>6</b>	3.4	ConT <sub>E</sub> Xt Implementation	250
2.1	Lua Interface . . . . .	6			
2.2	Plain T <sub>E</sub> X Interface . . . .	35	<b>References</b>		<b>255</b>
2.3	L <sup>A</sup> T <sub>E</sub> X Interface . . . . .	77			
2.4	ConT <sub>E</sub> Xt Interface . . . .	90	<b>Index</b>		<b>256</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 . .	32
3	A sequence diagram of typesetting a document using the Lua CLI . . . . .	32
4	Various formats of mathematical formulae . . . . .	83
5	The banner of the Markdown package . . . . .	84
6	A pushdown automaton that recognizes T <sub>E</sub> X comments . . . . .	156

## 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 convertors, 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 unicode
14 (function()
15   local ran_ok
16   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.

---

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

```

17   if not ran_ok then
18       unicode = {[ "utf8" ]={char=utf8.char}}
19   end
20 end()

```

**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).

```

21 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 packages:

**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 used to implement reflection capabilities that allow us to enumerate and inspect high-level concepts such as options, renderers, and renderer prototypes.

```

22 <@@=markdown>
23 \ifx\ExplSyntaxOn\undefined
24   \input expl3-generic\relax
25 \fi

```

**lt3luabridge** A package that allows us to execute Lua code with LuaTeX as well as with other TeX engines that provide the *shell escape* capability, which allows them to execute code with the system's shell.

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>TeX Requirements

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

```
26 \NeedsTeXFormat{LaTeX2e}%
```

a TeX engine that extends  $\varepsilon$ -TeX, and all the plain TeX prerequisites (see Section 1.1.2):

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>TeX 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>TeX 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` TeX command that is used in the default renderer prototype for citations. The package is included in TeXLive  $\geq$  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.

**soulutf8** A package that is used in the default renderer prototype for strike-throughs.

```
27 \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

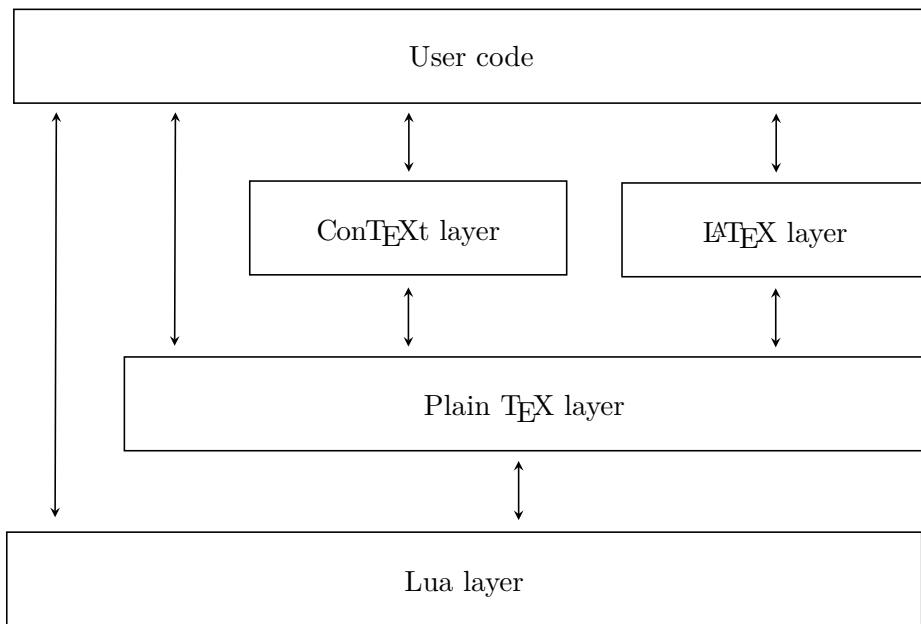
The Lua interface provides the conversion from UTF-8 encoded markdown to plain  $\text{T}_{\text{E}}\text{X}$ . This interface is used by the plain  $\text{T}_{\text{E}}\text{X}$  implementation (see Section 3.2) and will be of interest to the developers of other packages and Lua modules.

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

```
28 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.

```

29 local defaultOptions = {}

```

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

```

30 \ExplSyntaxOn
31 \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.

```

32 \prop_new:N \g_@@_lua_option_types_prop

```

```

33 \prop_new:N \g_@@_default_lua_options_prop
34 \seq_new:N \g_@@_option_layers_seq
35 \tl_const:Nn \c_@@_option_layer_lua_tl { lua }
36 \seq_put_right:NV \g_@@_option_layers_seq \c_@@_option_layer_lua_tl
37 \cs_new:Nn
38   \@@_add_lua_option:nnn
39   {
40     \@@_add_option:Vnnn
41     \c_@@_option_layer_lua_tl
42     { #1 }
43     { #2 }
44     { #3 }
45   }
46 \cs_new:Nn
47   \@@_add_option:nnnn
48   {
49     \seq_put_right:cn
50     { g_@@_ #1 _options_seq }
51     { #2 }
52     \prop_put:cnn
53     { g_@@_ #1 _option_types_prop }
54     { #2 }
55     { #3 }
56     \prop_put:cnn
57     { g_@@_default_ #1 _options_prop }
58     { #2 }
59     { #4 }
60     \@@_typecheck_option:n
61     { #2 }
62   }
63 \cs_generate_variant:Nn
64   \@@_add_option:nnnn
65   { Vnnn }
66 \tl_const:Nn \c_@@_option_value_true_tl { true }
67 \tl_const:Nn \c_@@_option_value_false_tl { false }
68 \cs_new:Nn \@@_typecheck_option:n
69   {
70     \@@_get_option_type:nN
71     { #1 }
72     \l_tmpa_tl
73     \str_case_e:Vn
74     \l_tmpa_tl
75     {
76       { \c_@@_option_type_boolean_tl }
77       {
78         \@@_get_option_value:nN
79         { #1 }

```



```

80         \l_tmpa_tl
81     \bool_if:nF
82     {
83         \str_if_eq_p:VV
84         \l_tmpa_tl
85         \c_@@_option_value_true_tl ||
86         \str_if_eq_p:VV
87         \l_tmpa_tl
88         \c_@@_option_value_false_tl
89     }
90     {
91         \msg_error:nnnV
92         { @@ }
93         { failed-typecheck-for-boolean-option }
94         { #1 }
95         \l_tmpa_tl
96     }
97 }
98 }
99 }
100 \msg_new:nnn
101 { @@ }
102 { failed-typecheck-for-boolean-option }
103 {
104     Option~#1~has~value~#2,~
105     but~a~boolean~(true~or~false)~was~expected.
106 }
107 \cs_generate_variant:Nn
108 \str_case_e:nn
109 { Vn }
110 \cs_generate_variant:Nn
111 \msg_error:nnnn
112 { nnnV }
113 \seq_new:N \g_@@_option_types_seq
114 \tl_const:Nn \c_@@_option_type_counter_tl { counter }
115 \seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_counter_tl
116 \tl_const:Nn \c_@@_option_type_boolean_tl { boolean }
117 \seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_boolean_tl
118 \tl_const:Nn \c_@@_option_type_number_tl { number }
119 \seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_number_tl
120 \tl_const:Nn \c_@@_option_type_path_tl { path }
121 \seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_path_tl
122 \tl_const:Nn \c_@@_option_type_slice_tl { slice }
123 \seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_slice_tl
124 \tl_const:Nn \c_@@_option_type_string_tl { string }
125 \seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_string_tl
126 \cs_new:Nn

```

```

127 \@@_get_option_type:nN
128 {
129   \bool_set_false:N
130   \l_tmpa_bool
131   \seq_map_inline:Nn
132   \g_@@_option_layers_seq
133   {
134     \prop_get:cnNT
135     { g_@@_ ##1 _option_types_prop }
136     { #1 }
137     \l_tmpa_tl
138     {
139       \bool_set_true:N
140       \l_tmpa_bool
141       \seq_map_break:
142     }
143   }
144   \bool_if:nF
145   \l_tmpa_bool
146   {
147     \msg_error:nnn
148     { @@ }
149     { undefined-option }
150     { #1 }
151   }
152   \seq_if_in:NVF
153   \g_@@_option_types_seq
154   \l_tmpa_tl
155   {
156     \msg_error:nnnV
157     { @@ }
158     { unknown-option-type }
159     { #1 }
160     \l_tmpa_tl
161   }
162   \tl_set_eq:NN
163   #2
164   \l_tmpa_tl
165 }
166 \msg_new:nnn
167 { @@ }
168 { unknown-option-type }
169 {
170   Option~#1~has~unknown~type~#2.
171 }
172 \msg_new:nnn
173 { @@ }

```

```

174 { undefined-option }
175 {
176     Option~#1~is~undefined.
177 }
178 \cs_new:Nn
179 \@@_get_default_option_value:nN
180 {
181     \bool_set_false:N
182     \l_tmpa_bool
183     \seq_map_inline:Nn
184     \g_@@_option_layers_seq
185     {
186         \prop_get:cnNT
187         { g_@@_default_ ##1 _options_prop }
188         { #1 }
189         #2
190         {
191             \bool_set_true:N
192             \l_tmpa_bool
193             \seq_map_break:
194         }
195     }
196     \bool_if:nF
197     \l_tmpa_bool
198     {
199         \msg_error:nnn
200         { @@ }
201         { undefined-option }
202         { #1 }
203     }
204 }
205 \cs_new:Nn
206 \@@_get_option_value:nN
207 {
208     \@@_option_tl_to_csname:nN
209     { #1 }
210     \l_tmpa_tl
211     \cs_if_free:cTF
212     { \l_tmpa_tl }
213     {
214         \@@_get_default_option_value:nN
215         { #1 }
216         #2
217     }
218     {
219         \@@_get_option_type:nN
220         { #1 }

```

```

221         \l_tmpa_tl
222     \str_if_eq:NNTF
223     \c_@@_option_type_counter_tl
224     \l_tmpa_tl
225     {
226         \@@_option_tl_to_csname:nN
227         { #1 }
228         \l_tmpa_tl
229         \tl_set:Nx
230         #2
231         { \the \cs:w \l_tmpa_tl \cs_end: }
232     }
233     {
234         \@@_option_tl_to_csname:nN
235         { #1 }
236         \l_tmpa_tl
237         \tl_set:Nv
238         #2
239         { \l_tmpa_tl }
240     }
241 }
242 }
243 \cs_new:Nn \@@_option_tl_to_csname:nN
244 {
245     \tl_set:Nn
246     \l_tmpa_tl
247     { \str_uppercase:n { #1 } }
248     \tl_set:Nx
249     #2
250     {
251         markdownOption
252         \tl_head:f { \l_tmpa_tl }
253         \tl_tail:n { #1 }
254     }
255 }

```

### 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.

```
256 \@@_add_lua_option:nnn
257   { cacheDir }
258   { path }
259   { \markdownOptionOutputDir / _markdown_\jobname }
260 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.

```
261 \@@_add_lua_option:nnn
262   { frozenCacheFileName }
263   { path }
264   { \markdownOptionCacheDir / frozenCache.tex }
265 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.

```
266 \@@_add_lua_option:nnn
267   { blankBeforeBlockquote }
268   { boolean }
269   { false }
270 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.

```
271 \@@_add_lua_option:nnn
272 { blankBeforeCodeFence }
273 { boolean }
274 { false }

275 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.

```
276 \@@_add_lua_option:nnn
277 { blankBeforeHeading }
278 { boolean }
279 { false }

280 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.

```
281 \@@_add_lua_option:nnn
282 { breakableBlockquotes }
283 { boolean }
284 { false }

285 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.

```
286 \@@_add_lua_option:nnn
287 { citationNbsps }
288 { boolean }
289 { true }

290 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.

```
291 \@@_add_lua_option:nnn
292 { citations }
293 { boolean }
294 { false }

295 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.``
```

```
296 \@@_add_lua_option:nnn  
297 { codeSpans }  
298 { boolean }  
299 { true }  
  
300 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.

```
301 \@@_add_lua_option:nnn  
302 { contentBlocks }  
303 { boolean }  
304 { false }  
  
305 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.

```
306 \@@_add_lua_option:nnn
307   { contentBlocksLanguageMap }
308   { path }
309   { markdown-languages.json }
310 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.

```
311 \@@_add_lua_option:nnn
312   { definitionLists }
313   { boolean }
314   { false }
315 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. Recursive nesting of markdown document fragments is undefined behavior when **eagerCache** is disabled.

```
316 \@@_add_lua_option:nnn
317   { eagerCache }
318   { boolean }
319   { true }
320 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}

```

```

321 \@@_add_lua_option:nnn
322   { expectJekyllData }
323   { boolean }
324   { false }

325 defaultOptions.expectJekyllData = false

```

`fancyLists=true, false`

default: false

`true` Enable the Pandoc fancy list extension:

```

a) first item
b) second item
c) third item

```

`false` Disable the Pandoc fancy list extension.

```

326 \@@_add_lua_option:nnn
327   { fancyLists }
328   { boolean }
329   { false }

330 defaultOptions.fancyLists = false

```

`fencedCode=true, false`

default: false

`true`

Enable the commonmark fenced code block extension:

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

``` html


```

<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.

```
331 \@@_add_lua_option:nnn
332 { fencedCode }
333 { boolean }
334 { false }
335 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  $\text{\TeX}$  document that contains markdown documents without invoking Lua using the `\markdownOptionFrozenCache` plain  $\text{\TeX}$  option. As a result, the plain  $\text{\TeX}$  document becomes more portable, but further changes in the order and the content of markdown documents will not be reflected.

```
336 \@@_add_lua_option:nnn
337 { finalizeCache }
338 { boolean }
339 { false }
```

```
340 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.

```
341 \@@_add_lua_option:nnn
```

```
342 { footnotes }
```

```
343 { boolean }
```

```
344 { false }
```

```
345 defaultOptions.footnotes = false
```

**frozenCacheCounter**=*<number>*

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***<number>* that will typeset markdown document number *<number>*.

```
346 \@@_add_lua_option:nnn
```

```
347 { frozenCacheCounter }
```

```
348 { counter }
```

```
349 { 0 }
```

```
350 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.

```
351 \@@_add_lua_option:nnn
352 { hardLineBreaks }
353 { boolean }
354 { false }

355 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.

```
356 \@@_add_lua_option:nnn
357 { hashEnumerators }
358 { boolean }
359 { false }

360 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.

```
361 \@@_add_lua_option:nnn
362   { headerAttributes }
363   { boolean }
364   { false }

365 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.

```
366 \@@_add_lua_option:nnn
367   { html }
368   { boolean }
369   { false }

370 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.

```
371 \@@_add_lua_option:nnn
372   { hybrid }
373   { boolean }
374   { false }

375 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.

```
376 \@@_add_lua_option:nnn
377   { inlineFootnotes }
378   { boolean }
379   { false }
380 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 `yml_metadata_block` syntax extension for entering metadata in YAML.

```
381 \@@_add_lua_option:nnn
382   { jeekyllData }
383   { boolean }
384   { false }
385 defaultOptions.jekyllData = false
```



`pipeTables=true, false`

default: false

**true** Enable the PHP Markdown pipe table syntax extension:

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

**false** Disable the PHP Markdown pipe table syntax extension.

```
386 \@@_add_lua_option:nnn
387 { pipeTables }
388 { boolean }
389 { false }
390 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.

```
391 \@@_add_lua_option:nnn
392 { preserveTabs }
393 { boolean }
394 { false }
395 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!
```

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

**false**      Disable relative references in autolinks.

```
396 \@@_add_lua_option:nnn
397   { relativeReferences }
398   { boolean }
399   { false }
400 defaultOptions.relativeReferences = false
```

**shiftHeadings**=*<shift amount>*      default: 0

All headings will be shifted by *<shift amount>*, 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 *<shift amount>* is positive, and to level 1, when *<shift amount>* is negative.

```
401 \@@_add_lua_option:nnn
402   { shiftHeadings }
403   { number }
404   { 0 }
405 defaultOptions.shiftHeadings = 0
```

**slice**=*<the beginning and the end of a slice>*      default: ^ \$

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 (^) selects the beginning of a document.
- The dollar sign (\$) selects the end of a document.
- ^*<identifier>* selects the beginning of a section with the HTML attribute #*<identifier>* (see the [headerAttributes](#) option).
- \$*<identifier>* selects the end of a section with the HTML attribute #*<identifier>*.
- *<identifier>* corresponds to ^*<identifier>* for the first selector and to \$*<identifier>* for the second selector.

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

```
406 \@@_add_lua_option:nnn
407   { slice }
408   { slice }
409   { ^~$ }
410 defaultOptions.slice = "^ $"
```

`smartEllipses=true, false` default: false

`true` Convert any ellipses in the input to the `\markdownRendererEllipsis` TeX macro.

`false` Preserve all ellipses in the input.

```
411 \@@_add_lua_option:nnn
412 { smartEllipses }
413 { boolean }
414 { false }
415 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 `\markdownRendererOListItemWithNumber` TeX macro.

`false` Ignore the numbers in the ordered list items. Each item will only produce a `\markdownRendererOListItem` TeX macro.

```
416 \@@_add_lua_option:nnn
417 { startNumber }
418 { boolean }
419 { true }
420 defaultOptions.startNumber = true
```

`strikeThrough=true, false` default: false

`true` Enable the Pandoc strike-through syntax extension:

This ~~is deleted text.~~

`false` Disable the Pandoc strike-through syntax extension.

```
421 \@@_add_lua_option:nnn
422 { strikeThrough }
423 { boolean }
424 { false }
425 defaultOptions.strikeThrough = false
```

`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 disabled:

```
\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.

```
426 \@@_add_lua_option:nnn
427   { stripIndent }
428   { boolean }
429   { false }
430 defaultOptions.stripIndent = false
```

`subscripts=true, false`

default: `false`

`true` Enable the Pandoc subscript syntax extension:

```
H~2~0 is a liquid.
```

`false` Disable the Pandoc subscript syntax extension.

```
431 \@@_add_lua_option:nnn
432   { subscripts }
433   { boolean }
434   { false }
435 defaultOptions.subscripts = false
```

`superscripts=true, false`

default: `false`

`true` Enable the Pandoc superscript syntax extension:

```
2^10^ is 1024.
```

`false` Disable the Pandoc superscript syntax extension.

```

436 \@@_add_lua_option:nnn
437   { superscripts }
438   { boolean }
439   { false }

440 defaultOptions.superscripts = 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.

```

441 \@@_add_lua_option:nnn
442   { tableCaptions }
443   { boolean }
444   { false }

445 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.

```

446 \@@_add_lua_option:nnn
447   { taskLists }
448   { boolean }
449   { false }

450 defaultOptions.taskLists = false

```

`texComments=true, false`

default: `false`

`true` Strip TeX-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 TeX-style comments.

```
451 \@@_add_lua_option:nnn
452   { texComments }
453   { boolean }
454   { false }
455 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 renderers 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.

```
456 \@@_add_lua_option:nnn
457   { tightLists }
458   { boolean }
459   { true }
```

```
460 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.

```
461 \@@_add_lua_option:nnn
462   { underscores }
463   { boolean }
464   { true }
465 \ExplSyntaxOff
466 defaultOptions.underscores = true
```

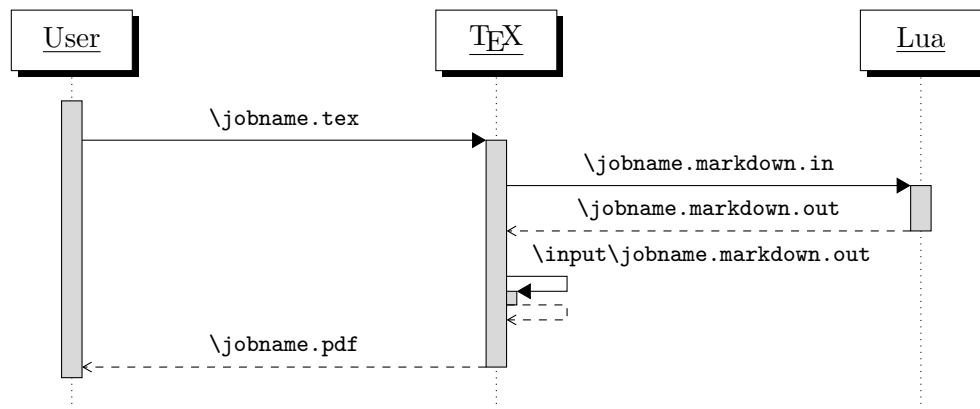
### 2.1.5 Command-Line Interface

The high-level operation of the Markdown package involves the communication between several programming layers: the plain  $\text{\TeX}$  layer hands markdown documents to the Lua layer. Lua converts the documents to  $\text{\TeX}$ , and hands the converted documents back to plain  $\text{\TeX}$  layer for typesetting, see Figure 2.

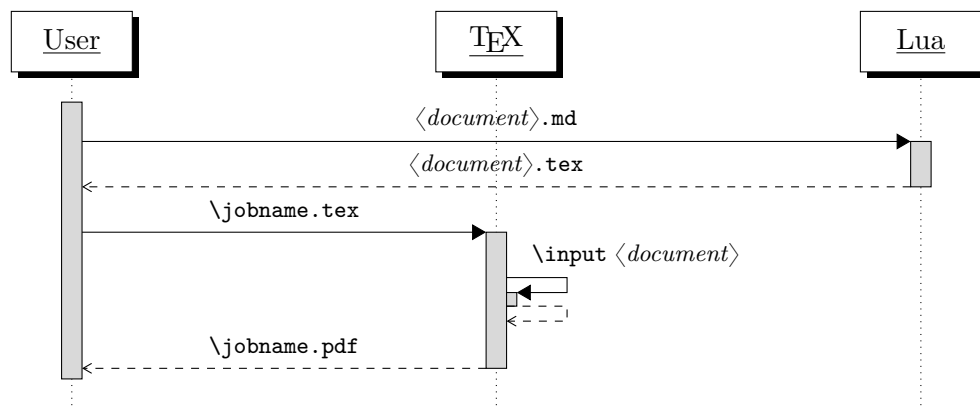
This procedure has the advantage of being fully automated. However, it also has several important disadvantages: The converted  $\text{\TeX}$  documents are cached on the file system, taking up increasing amount of space. Unless the  $\text{\TeX}$  engine includes a Lua interpreter, the package also 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  $\text{\TeX}$  is also provided, see Figure 3.

```
467
468 local HELP_STRING = [[
469 Usage: texlua ]] .. arg[0] .. [[ [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
470 where OPTIONS are documented in the Lua interface section of the
471 technical Markdown package documentation.
```



**Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the TeX interface**



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



```

472
473 When OUTPUT_FILE is unspecified, the result of the conversion will be
474 written to the standard output. When INPUT_FILE is also unspecified, the
475 result of the conversion will be read from the standard input.
476
477 Report bugs to: witiko@mail.muni.cz
478 Markdown package home page: <https://github.com/witiko/markdown>]]
479
480 local VERSION_STRING = [[
481 markdown-cli.lua (Markdown) ]] .. metadata.version .. [[
482
483 Copyright (C) ]] .. table.concat(metadata.copyright,
484                                     "\nCopyright (C) ") .. [[
485
486 License: ]] .. metadata.license
487
488 local function warn(s)
489     io.stderr:write("Warning: " .. s .. "\n") end
490
491 local function error(s)
492     io.stderr:write("Error: " .. s .. "\n")
493     os.exit(1) end
494
495 local process_options = true
496 local options = {}
497 local input_filename
498 local output_filename
499 for i = 1, #arg do
500     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.

```

501         if arg[i] == "--" then
502             process_options = false
503             goto continue

```

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

```

504         elseif arg[i]:match("=") then
505             local key, value = arg[i]:match("(.)=(.*)")

```

The `defaultOptions` table is consulted to identify whether `<value>` should be parsed as a string or as a boolean.

```

506             local default_type = type(defaultOptions[key])
507             if default_type == "boolean" then

```

```

508         options[key] = (value == "true")
509     elseif default_type == "number" then
510         options[key] = tonumber(value)
511     else
512         if default_type ~= "string" then
513             if default_type == "nil" then
514                 warn('Option "' .. key .. '" not recognized.')
515             else
516                 warn('Option "' .. key .. '" type not recognized, please file ' ..
517                     'a report to the package maintainer.')
518             end
519             warn('Parsing the ' .. 'value "' .. value ..'" of option "' ..
520                 key .. '" as a string.')
521         end
522         options[key] = value
523     end
524     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.

```

525     elseif arg[i] == "--help" or arg[i] == "-h" then
526         print(HELP_STRING)
527         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.

```

528     elseif arg[i] == "--version" or arg[i] == "-v" then
529         print(VERSION_STRING)
530         os.exit()
531     end
532 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  $\text{\TeX}$  document.

```

533 if input_filename == nil then
534     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.

```

535 elseif output_filename == nil then
536     output_filename = arg[i]
537 else
538     error('Unexpected argument: "' .. arg[i] .. '"')
539 end

```

```
540  ::continue::
541 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 T<sub>E</sub>X document `hello.tex`. After the Markdown package for our T<sub>E</sub>X format has been loaded, the converted document can be typeset as follows:

```
\input hello
```

## 2.2 Plain T<sub>E</sub>X Interface

The plain T<sub>E</sub>X interface provides macros for the typesetting of markdown input from within plain T<sub>E</sub>X, for setting the Lua interface options (see Section 2.1.2) used during the conversion from markdown to plain T<sub>E</sub>X and for changing the way markdown the tokens are rendered.

```
542 \def\markdownLastModified{(((LASTMODIFIED)))}%
543 \def\markdownVersion{(((VERSION)))}%
```

The plain T<sub>E</sub>X interface is implemented by the `markdown.tex` file that can be loaded as follows:

```
\input markdown
```

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

```
544 \let\markdownBegin\relax
545 \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 T<sub>E</sub>X.

546 `\let\markdownInput\relax`

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

The following example plain T<sub>E</sub>X 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.

To enable the enumeration of plain  $\TeX$  options, we will maintain the `\g_@@_plain_tex_options_seq` sequence.

```
547 \ExplSyntaxOn
548 \seq_new:N \g_@@_plain_tex_options_seq
```

To enable the reflection of default plain  $\TeX$  options and their types, we will maintain the `\g_@@_default_plain_tex_options_prop` and `\g_@@_plain_tex_option_types_prop` property lists, respectively.

```
549 \prop_new:N \g_@@_plain_tex_option_types_prop
550 \prop_new:N \g_@@_default_plain_tex_options_prop
551 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
552 \seq_put_right:NV \g_@@_option_layers_seq \c_@@_option_layer_plain_tex_tl
553 \cs_new:Nn
554   \@@_add_plain_tex_option:nnn
555   {
556     \@@_add_option:Vnnn
557     \c_@@_option_layer_plain_tex_tl
558     { #1 }
559     { #2 }
560     { #3 }
561   }
```

**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.

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`.

```
562 \@@_add_plain_tex_option:nnn
563   { frozenCache }
564   { boolean }
565   { 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 T<sub>E</sub>X document to populate and finalize the cache.
5. Enable the `\markdownOptionFrozenCache` option.
6. Publish the source code of the plain T<sub>E</sub>X 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 T<sub>E</sub>X in T<sub>E</sub>X 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 T<sub>E</sub>X engines tend to put quotation marks around `\jobname`, when it contains spaces.

```
566 \@@_add_plain_tex_option:nnn
567   { helperScriptFileName }
568   { path }
569   { \jobname.markdown.lua }
```

The `\markdownOptionHelperScriptFileName` macro has been deprecated and will be removed in Markdown 3.0.0. To control the filename of the helper Lua script file, use the `\g_luabridge_helper_script_filename_str` macro from the `lt3luabridge` package.

```
570 \str_new:N
571   \g_luabridge_helper_script_filename_str
572 \tl_gset:Nn
573   \g_luabridge_helper_script_filename_str
574   { \markdownOptionHelperScriptFileName }
```

The `\markdownOptionInputTempFileName` macro sets the filename of the temporary input file that is created during the buffering of markdown text from a T<sub>E</sub>X source. It defaults to `\jobname.markdown.in`. The same limitations as in the case of the `\markdownOptionHelperScriptFileName` macro apply here.

```
575 \@@_add_plain_tex_option:nnn
576   { inputTempFileName }
577   { path }
578   { \jobname.markdown.in }
```

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

```
579 \@@_add_plain_tex_option:nnn
580   { outputTempFileName }
581   { path }
582   { \jobname.markdown.out }
```

The `\markdownOptionOutputTempFileName` macro has been deprecated and will be removed in Markdown 3.0.0.

```
583 \str_new:N
584   \g_luabridge_standard_output_filename_str
585 \tl_gset:Nn
586   \g_luabridge_standard_output_filename_str
587   { \markdownOptionOutputTempFileName }
```

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.

```
588 \@@_add_plain_tex_option:nnn
589   { errorTempFileName }
590   { path }
591   { \jobname.markdown.err }
```

The `\markdownOptionErrorTempFileName` macro has been deprecated and will be removed in Markdown 3.0.0. To control the filename of the temporary file for Lua errors, use the `\g_luabridge_error_output_filename_str` macro from the `lt3luabridge` package.

```
592 \str_new:N
593   \g_luabridge_error_output_filename_str
594 \tl_gset:Nn
595   \g_luabridge_error_output_filename_str
596   { \markdownOptionErrorTempFileName }
```

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.

```
597 \@@_add_plain_tex_option:nnn
598   { outputDir }
599   { path }
600   { . }
```

Here, we automatically define plain TeX macros for the above plain TeX options.

Furthermore, we also define macros that map directly to the options recognized by the Lua interface, such as `\markdownOptionHybrid` for the `hybrid` Lua option (see Section 2.1.2), which are not processed by the plain TeX implementation, only passed along to Lua.

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.

```

601 \cs_new:Nn \@@_plain_tex_define_option_commands:
602 {
603   \seq_map_inline:Nn
604     \g_@@_option_layers_seq
605     {
606       \seq_map_inline:cn
607         { g_@@_ ##1 _options_seq }
608         {
609           \@@_plain_tex_define_option_command:n
610             { ####1 }
611         }
612     }
613 }
614 \cs_new:Nn \@@_plain_tex_define_option_command:n
615 {
616   \@@_get_default_option_value:nN
617     { #1 }
618     \l_tmpa_tl
619   \@@_set_option_value:nV
620     { #1 }
621     \l_tmpa_tl
622 }
623 \cs_new:Nn
624   \@@_set_option_value:nn
625   {
626     \@@_define_option:n
627       { #1 }
628     \@@_get_option_type:nN
629       { #1 }
630       \l_tmpa_tl
631     \str_if_eq:NNTF
632       \c_@@_option_type_counter_tl
633       \l_tmpa_tl
634       {
635         \@@_option_tl_to_csname:nN
636           { #1 }
637           \l_tmpa_tl
638         \int_gset:cn
639           { \l_tmpa_tl }
640           { #2 }
641       }
642     {
643       \@@_option_tl_to_csname:nN
644         { #1 }

```



```

645         \l_tmpa_tl
646         \cs_set:cpn
647         { \l_tmpa_tl }
648         { #2 }
649     }
650 }
651 \cs_generate_variant:Nn
652   \@@_set_option_value:nn
653   { nV }
654 \cs_new:Nn
655   \@@_define_option:n
656   {
657     \@@_option_tl_to_csname:nN
658     { #1 }
659     \l_tmpa_tl
660     \cs_if_free:cT
661     { \l_tmpa_tl }
662     {
663       \@@_get_option_type:nN
664       { #1 }
665       \l_tmpb_tl
666       \str_if_eq:NNT
667       \c_@@_option_type_counter_tl
668       \l_tmpb_tl
669       {
670         \@@_option_tl_to_csname:nN
671         { #1 }
672         \l_tmpa_tl
673         \int_new:c
674         { \l_tmpa_tl }
675       }
676     }
677   }
678 \@@_plain_tex_define_option_commands:

```

**2.2.2.3 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 3.2.4) 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`.

```

679 \seq_put_right:Nn
680   \g_@@_plain_tex_options_seq
681   { stripPercentSigns }
682 \prop_put:Nnn
683   \g_@@_plain_tex_option_types_prop

```

```

684 { stripPercentSigns }
685 { boolean }
686 \prop_put:Nnx
687 \g_@@_default_plain_tex_options_prop
688 { stripPercentSigns }
689 { false }
690 \ExplSyntaxOff

```

### 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.

```

691 \ExplSyntaxOn
692 \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.

```

693 \prop_new:N \g_@@_renderer_arities_prop
694 \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.

```

695 \def\markdownRendererTickedBox{%
696 \markdownRendererTickedBoxPrototype}%
697 \ExplSyntaxOn
698 \seq_put_right:Nn
699 \g_@@_renderers_seq
700 { tickedBox }
701 \prop_put:Nnn
702 \g_@@_renderer_arities_prop
703 { tickedBox }
704 { 0 }
705 \ExplSyntaxOff
706 \def\markdownRendererHalfTickedBox{%
707 \markdownRendererHalfTickedBoxPrototype}%
708 \ExplSyntaxOn
709 \seq_put_right:Nn

```

```

710 \g_@@_renderers_seq
711 { halfTickedBox }
712 \prop_put:Nnn
713 \g_@@_renderer_arities_prop
714 { halfTickedBox }
715 { 0 }
716 \ExplSyntaxOff
717 \def\markdownRendererUntickedBox{%
718 \markdownRendererUntickedBoxPrototype}%
719 \ExplSyntaxOn
720 \seq_put_right:Nn
721 \g_@@_renderers_seq
722 { untickedBox }
723 \prop_put:Nnn
724 \g_@@_renderer_arities_prop
725 { untickedBox }
726 { 0 }
727 \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.

```

728 \def\markdownRendererDocumentBegin{%
729 \markdownRendererDocumentBeginPrototype}%
730 \ExplSyntaxOn
731 \seq_put_right:Nn
732 \g_@@_renderers_seq
733 { documentBegin }
734 \prop_put:Nnn
735 \g_@@_renderer_arities_prop
736 { documentBegin }
737 { 0 }
738 \ExplSyntaxOff
739 \def\markdownRendererDocumentEnd{%
740 \markdownRendererDocumentEndPrototype}%
741 \ExplSyntaxOn
742 \seq_put_right:Nn
743 \g_@@_renderers_seq
744 { documentEnd }
745 \prop_put:Nnn
746 \g_@@_renderer_arities_prop

```

```

747 { documentEnd }
748 { 0 }
749 \ExplSyntaxOff

```

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

```

750 \def\markdownRendererInterblockSeparator{%
751   \markdownRendererInterblockSeparatorPrototype}%
752 \ExplSyntaxOn
753 \seq_put_right:Nn
754   \g_@@_renderers_seq
755   { interblockSeparator }
756 \prop_put:Nnn
757   \g_@@_renderer_arities_prop
758   { interblockSeparator }
759   { 0 }
760 \ExplSyntaxOff

```

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

```

761 \def\markdownRendererLineBreak{%
762   \markdownRendererLineBreakPrototype}%
763 \ExplSyntaxOn
764 \seq_put_right:Nn
765   \g_@@_renderers_seq
766   { lineBreak }
767 \prop_put:Nnn
768   \g_@@_renderer_arities_prop
769   { lineBreak }
770   { 0 }
771 \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.

```

772 \def\markdownRendererEllipsis{%
773   \markdownRendererEllipsisPrototype}%
774 \ExplSyntaxOn
775 \seq_put_right:Nn
776   \g_@@_renderers_seq
777   { ellipsis }
778 \prop_put:Nnn
779   \g_@@_renderer_arities_prop

```

```

780 { ellipsis }
781 { 0 }
782 \ExplSyntaxOff

```

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

```

783 \def\markdownRendererNbsp{%
784   \markdownRendererNbspPrototype}%
785 \ExplSyntaxOn
786 \seq_put_right:Nn
787   \g_@@_renderers_seq
788   { nbsp }
789 \prop_put:Nnn
790   \g_@@_renderer_arities_prop
791   { nbsp }
792   { 0 }
793 \ExplSyntaxOff

```

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

```

794 \def\markdownRendererLeftBrace{%
795   \markdownRendererLeftBracePrototype}%
796 \ExplSyntaxOn
797 \seq_put_right:Nn
798   \g_@@_renderers_seq
799   { leftBrace }
800 \prop_put:Nnn
801   \g_@@_renderer_arities_prop
802   { leftBrace }
803   { 0 }
804 \ExplSyntaxOff
805 \def\markdownRendererRightBrace{%
806   \markdownRendererRightBracePrototype}%
807 \ExplSyntaxOn
808 \seq_put_right:Nn
809   \g_@@_renderers_seq
810   { rightBrace }
811 \prop_put:Nnn
812   \g_@@_renderer_arities_prop
813   { rightBrace }
814   { 0 }
815 \ExplSyntaxOff
816 \def\markdownRendererDollarSign{%
817   \markdownRendererDollarSignPrototype}%

```

```

818 \ExplSyntaxOn
819 \seq_put_right:Nn
820   \g_@@_renderers_seq
821   { dollarSign }
822 \prop_put:Nnn
823   \g_@@_renderer_arities_prop
824   { dollarSign }
825   { 0 }
826 \ExplSyntaxOff
827 \def\markdownRendererPercentSign{%
828   \markdownRendererPercentSignPrototype}%
829 \ExplSyntaxOn
830 \seq_put_right:Nn
831   \g_@@_renderers_seq
832   { percentSign }
833 \prop_put:Nnn
834   \g_@@_renderer_arities_prop
835   { percentSign }
836   { 0 }
837 \ExplSyntaxOff
838 \def\markdownRendererAmpersand{%
839   \markdownRendererAmpersandPrototype}%
840 \ExplSyntaxOn
841 \seq_put_right:Nn
842   \g_@@_renderers_seq
843   { ampersand }
844 \prop_put:Nnn
845   \g_@@_renderer_arities_prop
846   { ampersand }
847   { 0 }
848 \ExplSyntaxOff
849 \def\markdownRendererUnderscore{%
850   \markdownRendererUnderscorePrototype}%
851 \ExplSyntaxOn
852 \seq_put_right:Nn
853   \g_@@_renderers_seq
854   { underscore }
855 \prop_put:Nnn
856   \g_@@_renderer_arities_prop
857   { underscore }
858   { 0 }
859 \ExplSyntaxOff
860 \def\markdownRendererHash{%
861   \markdownRendererHashPrototype}%
862 \ExplSyntaxOn
863 \seq_put_right:Nn
864   \g_@@_renderers_seq

```

```

865 { hash }
866 \prop_put:Nnn
867 \g_@@_renderer_arities_prop
868 { hash }
869 { 0 }
870 \ExplSyntaxOff
871 \def\markdownRendererCircumflex{%
872 \markdownRendererCircumflexPrototype}%
873 \ExplSyntaxOn
874 \seq_put_right:Nn
875 \g_@@_renderers_seq
876 { circumflex }
877 \prop_put:Nnn
878 \g_@@_renderer_arities_prop
879 { circumflex }
880 { 0 }
881 \ExplSyntaxOff
882 \def\markdownRendererBackslash{%
883 \markdownRendererBackslashPrototype}%
884 \ExplSyntaxOn
885 \seq_put_right:Nn
886 \g_@@_renderers_seq
887 { backslash }
888 \prop_put:Nnn
889 \g_@@_renderer_arities_prop
890 { backslash }
891 { 0 }
892 \ExplSyntaxOff
893 \def\markdownRendererTilde{%
894 \markdownRendererTildePrototype}%
895 \ExplSyntaxOn
896 \seq_put_right:Nn
897 \g_@@_renderers_seq
898 { tilde }
899 \prop_put:Nnn
900 \g_@@_renderer_arities_prop
901 { tilde }
902 { 0 }
903 \ExplSyntaxOff
904 \def\markdownRendererPipe{%
905 \markdownRendererPipePrototype}%
906 \ExplSyntaxOn
907 \seq_put_right:Nn
908 \g_@@_renderers_seq
909 { pipe }
910 \prop_put:Nnn
911 \g_@@_renderer_arities_prop

```

```

912 { pipe }
913 { 0 }
914 \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.

```

915 \def\markdownRendererCodeSpan{%
916   \markdownRendererCodeSpanPrototype}%
917 \ExplSyntaxOn
918 \seq_put_right:Nn
919   \g_@@_renderers_seq
920   { codeSpan }
921 \prop_put:Nnn
922   \g_@@_renderer_arities_prop
923   { codeSpan }
924   { 1 }
925 \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.

```

926 \def\markdownRendererLink{%
927   \markdownRendererLinkPrototype}%
928 \ExplSyntaxOn
929 \seq_put_right:Nn
930   \g_@@_renderers_seq
931   { link }
932 \prop_put:Nnn
933   \g_@@_renderer_arities_prop
934   { link }
935   { 4 }
936 \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.

```

937 \def\markdownRendererImage{%
938   \markdownRendererImagePrototype}%
939 \ExplSyntaxOn
940 \seq_put_right:Nn
941   \g_@@_renderers_seq
942   { image }

```



```

943 \prop_put:Nnn
944   \g_@@_renderer_arities_prop
945   { image }
946   { 4 }
947 \ExplSyntaxOff

```

**2.2.3.11 Content Block Renderers** 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.

```

948 \def\markdownRendererContentBlock{%
949   \markdownRendererContentBlockPrototype}%
950 \ExplSyntaxOn
951 \seq_put_right:Nn
952   \g_@@_renderers_seq
953   { contentBlock }
954 \prop_put:Nnn
955   \g_@@_renderer_arities_prop
956   { contentBlock }
957   { 4 }
958 \ExplSyntaxOff

```

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

```

959 \def\markdownRendererContentBlockOnlineImage{%
960   \markdownRendererContentBlockOnlineImagePrototype}%
961 \ExplSyntaxOn
962 \seq_put_right:Nn
963   \g_@@_renderers_seq
964   { contentBlockOnlineImage }
965 \prop_put:Nnn
966   \g_@@_renderer_arities_prop
967   { contentBlockOnlineImage }
968   { 4 }
969 \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 extension  $s$ ,  $s:\text{lower}() = k$  is considered to be in a known programming language  $v$ .

---

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

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 T<sub>E</sub>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.

```

970 \def\markdownRendererContentBlockCode{%
971   \markdownRendererContentBlockCodePrototype}%
972 \ExplSyntaxOn
973 \seq_put_right:Nn
974   \g_@@_renderers_seq
975   { contentBlockCode }
976 \prop_put:Nnn
977   \g_@@_renderer_arities_prop
978   { contentBlockCode }
979   { 5 }
980 \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.

```

981 \def\markdownRendererUlBegin{%
982   \markdownRendererUlBeginPrototype}%
983 \ExplSyntaxOn
984 \seq_put_right:Nn
985   \g_@@_renderers_seq
986   { ulBegin }
987 \prop_put:Nnn
988   \g_@@_renderer_arities_prop
989   { ulBegin }
990   { 0 }
991 \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 disabled. The macro receives no arguments.

```

992 \def\markdownRendererUlBeginTight{%
993   \markdownRendererUlBeginTightPrototype}%
994 \ExplSyntaxOn
995 \seq_put_right:Nn

```

```

996 \g_@@_renderers_seq
997 { ulBeginTight }
998 \prop_put:Nnn
999 \g_@@_renderer_arities_prop
1000 { ulBeginTight }
1001 { 0 }
1002 \ExplSyntaxOff

```

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

```

1003 \def\markdownRendererUItem{%
1004 \markdownRendererUItemPrototype}%
1005 \ExplSyntaxOn
1006 \seq_put_right:Nn
1007 \g_@@_renderers_seq
1008 { ulItem }
1009 \prop_put:Nnn
1010 \g_@@_renderer_arities_prop
1011 { ulItem }
1012 { 0 }
1013 \ExplSyntaxOff

```

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

```

1014 \def\markdownRendererUItemEnd{%
1015 \markdownRendererUItemEndPrototype}%
1016 \ExplSyntaxOn
1017 \seq_put_right:Nn
1018 \g_@@_renderers_seq
1019 { ulItemEnd }
1020 \prop_put:Nnn
1021 \g_@@_renderer_arities_prop
1022 { ulItemEnd }
1023 { 0 }
1024 \ExplSyntaxOff

```

The `\markdownRendererUEnd` 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.

```

1025 \def\markdownRendererUEnd{%
1026 \markdownRendererUEndPrototype}%
1027 \ExplSyntaxOn
1028 \seq_put_right:Nn
1029 \g_@@_renderers_seq
1030 { ulEnd }
1031 \prop_put:Nnn

```

```

1032 \g_@@_renderer_arities_prop
1033 { ulEnd }
1034 { 0 }
1035 \ExplSyntaxOff

```

The `\markdownRendererUEndTight` 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 disabled. The macro receives no arguments.

```

1036 \def\markdownRendererUEndTight{%
1037 \markdownRendererUEndTightPrototype}%
1038 \ExplSyntaxOn
1039 \seq_put_right:Nn
1040 \g_@@_renderers_seq
1041 { ulEndTight }
1042 \prop_put:Nnn
1043 \g_@@_renderer_arities_prop
1044 { ulEndTight }
1045 { 0 }
1046 \ExplSyntaxOff

```

**2.2.3.13 Ordered List Renderers** The `\markdownRendererOlBegin` macro represents the beginning of an ordered list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the `fancyLists` option is disabled. The macro receives no arguments.

```

1047 \def\markdownRendererOlBegin{%
1048 \markdownRendererOlBeginPrototype}%
1049 \ExplSyntaxOn
1050 \seq_put_right:Nn
1051 \g_@@_renderers_seq
1052 { olBegin }
1053 \prop_put:Nnn
1054 \g_@@_renderer_arities_prop
1055 { olBegin }
1056 { 0 }
1057 \ExplSyntaxOff

```

The `\markdownRendererOlBeginTight` 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 enabled and the `fancyLists` option is disabled. The macro receives no arguments.

```

1058 \def\markdownRendererOlBeginTight{%
1059 \markdownRendererOlBeginTightPrototype}%
1060 \ExplSyntaxOn
1061 \seq_put_right:Nn

```

```

1062 \g_@@_renderers_seq
1063 { olBeginTight }
1064 \prop_put:Nnn
1065 \g_@@_renderer_arities_prop
1066 { olBeginTight }
1067 { 0 }
1068 \ExplSyntaxOff

```

The `\markdownRendererFancyOlBegin` macro represents the beginning of a fancy ordered list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the `fancyLists` option is enabled. The macro receives two arguments: the style of the list item labels (`Decimal`, `LowerRoman`, `UpperRoman`, `LowerAlpha`, and `UpperAlpha`), and the style of delimiters between list item labels and texts (`Default`, `OneParen`, and `Period`).

```

1069 \def\markdownRendererFancyOlBegin{%
1070 \markdownRendererFancyOlBeginPrototype}%
1071 \ExplSyntaxOn
1072 \seq_put_right:Nn
1073 \g_@@_renderers_seq
1074 { fancyOlBegin }
1075 \prop_put:Nnn
1076 \g_@@_renderer_arities_prop
1077 { fancyOlBegin }
1078 { 2 }
1079 \ExplSyntaxOff

```

The `\markdownRendererFancyOlBeginTight` macro represents the beginning of a fancy ordered list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `fancyLists` and `tightLists` options are enabled. The macro receives two arguments: the style of the list item labels, and the style of delimiters between list item labels and texts. See the `\markdownRendererFancyOlBegin` macro for the valid style values.

```

1080 \def\markdownRendererFancyOlBeginTight{%
1081 \markdownRendererFancyOlBeginTightPrototype}%
1082 \ExplSyntaxOn
1083 \seq_put_right:Nn
1084 \g_@@_renderers_seq
1085 { fancyOlBeginTight }
1086 \prop_put:Nnn
1087 \g_@@_renderer_arities_prop
1088 { fancyOlBeginTight }
1089 { 2 }
1090 \ExplSyntaxOff

```

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

```

1091 \def\markdownRendererOliItem{%
1092   \markdownRendererOliItemPrototype}%
1093 \ExplSyntaxOn
1094 \seq_put_right:Nn
1095   \g_@@_renderers_seq
1096   { oliItem }
1097 \prop_put:Nnn
1098   \g_@@_renderer_arities_prop
1099   { oliItem }
1100   { 0 }
1101 \ExplSyntaxOff

```

The `\markdownRendererOliItemEnd` macro represents the end of an item in an ordered list. This macro will only be produced, when the `fancyLists` option is disabled. The macro receives no arguments.

```

1102 \def\markdownRendererOliItemEnd{%
1103   \markdownRendererOliItemEndPrototype}%
1104 \ExplSyntaxOn
1105 \seq_put_right:Nn
1106   \g_@@_renderers_seq
1107   { oliItemEnd }
1108 \prop_put:Nnn
1109   \g_@@_renderer_arities_prop
1110   { oliItemEnd }
1111   { 0 }
1112 \ExplSyntaxOff

```

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

```

1113 \def\markdownRendererOliItemWithNumber{%
1114   \markdownRendererOliItemWithNumberPrototype}%
1115 \ExplSyntaxOn
1116 \seq_put_right:Nn
1117   \g_@@_renderers_seq
1118   { oliItemWithNumber }
1119 \prop_put:Nnn
1120   \g_@@_renderer_arities_prop
1121   { oliItemWithNumber }
1122   { 1 }
1123 \ExplSyntaxOff

```

The `\markdownRendererFancy01Item` macro represents an item in a fancy ordered list. This macro will only be produced, when the `startNumber` option is disabled and the `fancyLists` option is enabled. The macro receives no arguments.

```

1124 \def\markdownRendererFancy01Item{%
1125   \markdownRendererFancy01ItemPrototype}%
1126 \ExplSyntaxOn
1127 \seq_put_right:Nn
1128   \g_@@_renderers_seq
1129   { fancy01Item }
1130 \prop_put:Nnn
1131   \g_@@_renderer_arities_prop
1132   { fancy01Item }
1133   { 0 }
1134 \ExplSyntaxOff

```

The `\markdownRendererFancy01ItemEnd` macro represents the end of an item in a fancy ordered list. This macro will only be produced, when the `fancyLists` option is enabled. The macro receives no arguments.

```

1135 \def\markdownRendererFancy01ItemEnd{%
1136   \markdownRendererFancy01ItemEndPrototype}%
1137 \ExplSyntaxOn
1138 \seq_put_right:Nn
1139   \g_@@_renderers_seq
1140   { fancy01ItemEnd }
1141 \prop_put:Nnn
1142   \g_@@_renderer_arities_prop
1143   { fancy01ItemEnd }
1144   { 0 }
1145 \ExplSyntaxOff

```

The `\markdownRendererFancy01ItemWithNumber` macro represents an item in a fancy ordered list. This macro will only be produced, when the `startNumber` and `fancyLists` options are enabled. The macro receives a single numeric argument that corresponds to the item number.

```

1146 \def\markdownRendererFancy01ItemWithNumber{%
1147   \markdownRendererFancy01ItemWithNumberPrototype}%
1148 \ExplSyntaxOn
1149 \seq_put_right:Nn
1150   \g_@@_renderers_seq
1151   { fancy01ItemWithNumber }
1152 \prop_put:Nnn
1153   \g_@@_renderer_arities_prop
1154   { fancy01ItemWithNumber }
1155   { 1 }
1156 \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). This macro will only be produced, when the `fancyLists` option is disabled. The macro receives no arguments.

```

1157 \def\markdownRendererOlEnd{%
1158   \markdownRendererOlEndPrototype}%
1159 \ExplSyntaxOn
1160 \seq_put_right:Nn
1161   \g_@@_renderers_seq
1162   { olEnd }
1163 \prop_put:Nnn
1164   \g_@@_renderer_arities_prop
1165   { olEnd }
1166   { 0 }
1167 \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 enabled and the `fancyLists` option is disabled. The macro receives no arguments.

```

1168 \def\markdownRendererOlEndTight{%
1169   \markdownRendererOlEndTightPrototype}%
1170 \ExplSyntaxOn
1171 \seq_put_right:Nn
1172   \g_@@_renderers_seq
1173   { olEndTight }
1174 \prop_put:Nnn
1175   \g_@@_renderer_arities_prop
1176   { olEndTight }
1177   { 0 }
1178 \ExplSyntaxOff

```

The `\markdownRendererFancyOlEnd` macro represents the end of a fancy ordered list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the `fancyLists` option is enabled. The macro receives no arguments.

```

1179 \def\markdownRendererFancyOlEnd{%
1180   \markdownRendererFancyOlEndPrototype}%
1181 \ExplSyntaxOn
1182 \seq_put_right:Nn
1183   \g_@@_renderers_seq
1184   { fancyOlEnd }
1185 \prop_put:Nnn
1186   \g_@@_renderer_arities_prop
1187   { fancyOlEnd }

```



```

1188 { 0 }
1189 \ExplSyntaxOff

```

The `\markdownRendererFancyOlEndTight` macro represents the end of a fancy ordered list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `fancyLists` and `tightLists` options are enabled. The macro receives no arguments.

```

1190 \def\markdownRendererFancyOlEndTight{%
1191   \markdownRendererFancyOlEndTightPrototype}%
1192 \ExplSyntaxOn
1193 \seq_put_right:Nn
1194   \g_@@_renderers_seq
1195   { fancyOlEndTight }
1196 \prop_put:Nnn
1197   \g_@@_renderer_arities_prop
1198   { fancyOlEndTight }
1199   { 0 }
1200 \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.

```

1201 \def\markdownRendererDlBegin{%
1202   \markdownRendererDlBeginPrototype}%
1203 \ExplSyntaxOn
1204 \seq_put_right:Nn
1205   \g_@@_renderers_seq
1206   { dlBegin }
1207 \prop_put:Nnn
1208   \g_@@_renderer_arities_prop
1209   { dlBegin }
1210   { 0 }
1211 \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 disabled. The macro receives no arguments.

```

1212 \def\markdownRendererDlBeginTight{%
1213   \markdownRendererDlBeginTightPrototype}%
1214 \ExplSyntaxOn
1215 \seq_put_right:Nn
1216   \g_@@_renderers_seq

```

```

1217 { dlBeginTight }
1218 \prop_put:Nnn
1219 \g_@@_render_arity_prop
1220 { dlBeginTight }
1221 { 0 }
1222 \ExplSyntaxOff

```

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

```

1223 \def\markdownRendererDlItem{%
1224   \markdownRendererDlItemPrototype}%
1225 \ExplSyntaxOn
1226 \seq_put_right:Nn
1227   \g_@@_renderers_seq
1228   { dlItem }
1229 \prop_put:Nnn
1230   \g_@@_render_arity_prop
1231   { dlItem }
1232   { 1 }
1233 \ExplSyntaxOff

```

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

```

1234 \def\markdownRendererDlItemEnd{%
1235   \markdownRendererDlItemEndPrototype}%
1236 \ExplSyntaxOn
1237 \seq_put_right:Nn
1238   \g_@@_renderers_seq
1239   { dlItemEnd }
1240 \prop_put:Nnn
1241   \g_@@_render_arity_prop
1242   { dlItemEnd }
1243   { 0 }
1244 \ExplSyntaxOff

```

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

```

1245 \def\markdownRendererDlDefinitionBegin{%
1246   \markdownRendererDlDefinitionBeginPrototype}%
1247 \ExplSyntaxOn
1248 \seq_put_right:Nn
1249   \g_@@_renderers_seq
1250   { dlDefinitionBegin }
1251 \prop_put:Nnn
1252   \g_@@_render_arity_prop
1253   { dlDefinitionBegin }
1254   { 0 }

```

1255 \ExplSyntaxOff

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

```
1256 \def\markdownRendererDlDefinitionEnd{%
1257   \markdownRendererDlDefinitionEndPrototype}%
1258 \ExplSyntaxOn
1259 \seq_put_right:Nn
1260   \g_@@_renderers_seq
1261   { dlDefinitionEnd }
1262 \prop_put:Nnn
1263   \g_@@_renderer_arities_prop
1264   { dlDefinitionEnd }
1265   { 0 }
1266 \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.

```
1267 \def\markdownRendererDlEnd{%
1268   \markdownRendererDlEndPrototype}%
1269 \ExplSyntaxOn
1270 \seq_put_right:Nn
1271   \g_@@_renderers_seq
1272   { dlEnd }
1273 \prop_put:Nnn
1274   \g_@@_renderer_arities_prop
1275   { dlEnd }
1276   { 0 }
1277 \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 disabled. The macro receives no arguments.

```
1278 \def\markdownRendererDlEndTight{%
1279   \markdownRendererDlEndTightPrototype}%
1280 \ExplSyntaxOn
1281 \seq_put_right:Nn
1282   \g_@@_renderers_seq
1283   { dlEndTight }
1284 \prop_put:Nnn
1285   \g_@@_renderer_arities_prop
1286   { dlEndTight }
1287   { 0 }
1288 \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.

```

1289 \def\markdownRendererEmphasis{%
1290   \markdownRendererEmphasisPrototype}%
1291 \ExplSyntaxOn
1292 \seq_put_right:Nn
1293   \g_@@_renderers_seq
1294   { emphasis }
1295 \prop_put:Nnn
1296   \g_@@_renderer_arities_prop
1297   { emphasis }
1298   { 1 }
1299 \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.

```

1300 \def\markdownRendererStrongEmphasis{%
1301   \markdownRendererStrongEmphasisPrototype}%
1302 \ExplSyntaxOn
1303 \seq_put_right:Nn
1304   \g_@@_renderers_seq
1305   { strongEmphasis }
1306 \prop_put:Nnn
1307   \g_@@_renderer_arities_prop
1308   { strongEmphasis }
1309   { 1 }
1310 \ExplSyntaxOff

```

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

```

1311 \def\markdownRendererBlockQuoteBegin{%
1312   \markdownRendererBlockQuoteBeginPrototype}%
1313 \ExplSyntaxOn
1314 \seq_put_right:Nn
1315   \g_@@_renderers_seq
1316   { blockQuoteBegin }
1317 \prop_put:Nnn
1318   \g_@@_renderer_arities_prop
1319   { blockQuoteBegin }
1320   { 0 }
1321 \ExplSyntaxOff

```

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

```

1322 \def\markdownRendererBlockQuoteEnd{%
1323   \markdownRendererBlockQuoteEndPrototype}%
1324 \ExplSyntaxOn
1325 \seq_put_right:Nn
1326   \g_@@_renderers_seq
1327   { blockQuoteEnd }
1328 \prop_put:Nnn
1329   \g_@@_renderer_arities_prop
1330   { blockQuoteEnd }
1331   { 0 }
1332 \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.

```

1333 \def\markdownRendererInputVerbatim{%
1334   \markdownRendererInputVerbatimPrototype}%
1335 \ExplSyntaxOn
1336 \seq_put_right:Nn
1337   \g_@@_renderers_seq
1338   { inputVerbatim }
1339 \prop_put:Nnn
1340   \g_@@_renderer_arities_prop
1341   { inputVerbatim }
1342   { 1 }
1343 \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.

```

1344 \def\markdownRendererInputFencedCode{%
1345   \markdownRendererInputFencedCodePrototype}%
1346 \ExplSyntaxOn
1347 \seq_put_right:Nn
1348   \g_@@_renderers_seq
1349   { inputFencedCode }
1350 \prop_put:Nnn
1351   \g_@@_renderer_arities_prop
1352   { inputFencedCode }
1353   { 2 }
1354 \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.

```

1355 \def\markdownRendererJekyllDataBegin{%
1356   \markdownRendererJekyllDataBeginPrototype}%
1357 \ExplSyntaxOn
1358 \seq_put_right:Nn
1359   \g_@@_renderers_seq
1360   { jekyllDataBegin }
1361 \prop_put:Nnn
1362   \g_@@_renderer_arities_prop
1363   { jekyllDataBegin }
1364   { 0 }
1365 \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.

```

1366 \def\markdownRendererJekyllDataEnd{%
1367   \markdownRendererJekyllDataEndPrototype}%
1368 \ExplSyntaxOn
1369 \seq_put_right:Nn
1370   \g_@@_renderers_seq
1371   { jekyllDataEnd }
1372 \prop_put:Nnn
1373   \g_@@_renderer_arities_prop
1374   { jekyllDataEnd }
1375   { 0 }
1376 \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.

```

1377 \def\markdownRendererJekyllDataMappingBegin{%
1378   \markdownRendererJekyllDataMappingBeginPrototype}%
1379 \ExplSyntaxOn
1380 \seq_put_right:Nn
1381   \g_@@_renderers_seq
1382   { jekyllDataMappingBegin }
1383 \prop_put:Nnn
1384   \g_@@_renderer_arities_prop
1385   { jekyllDataMappingBegin }
1386   { 2 }
1387 \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.

```

1388 \def\markdownRendererJekyllDataMappingEnd{%
1389   \markdownRendererJekyllDataMappingEndPrototype}%
1390 \ExplSyntaxOn
1391 \seq_put_right:Nn
1392   \g_@@_renderers_seq
1393   { jekyllDataMappingEnd }
1394 \prop_put:Nnn
1395   \g_@@_renderer_arities_prop
1396   { jekyllDataMappingEnd }
1397   { 0 }
1398 \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.

```

1399 \def\markdownRendererJekyllDataSequenceBegin{%
1400   \markdownRendererJekyllDataSequenceBeginPrototype}%
1401 \ExplSyntaxOn
1402 \seq_put_right:Nn
1403   \g_@@_renderers_seq
1404   { jekyllDataSequenceBegin }
1405 \prop_put:Nnn
1406   \g_@@_renderer_arities_prop
1407   { jekyllDataSequenceBegin }
1408   { 2 }
1409 \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.

```

1410 \def\markdownRendererJekyllDataSequenceEnd{%
1411   \markdownRendererJekyllDataSequenceEndPrototype}%
1412 \ExplSyntaxOn
1413 \seq_put_right:Nn
1414   \g_@@_renderers_seq
1415   { jekyllDataSequenceEnd }
1416 \prop_put:Nnn
1417   \g_@@_renderer_arities_prop
1418   { jekyllDataSequenceEnd }
1419   { 0 }
1420 \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.

```

1421 \def\markdownRendererJekyllDataBoolean{%
1422   \markdownRendererJekyllDataBooleanPrototype}%
1423 \ExplSyntaxOn
1424 \seq_put_right:Nn
1425   \g_@@_renderers_seq
1426   { jekyllDataBoolean }
1427 \prop_put:Nnn
1428   \g_@@_renderer_arities_prop
1429   { jekyllDataBoolean }
1430   { 2 }
1431 \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.

```

1432 \def\markdownRendererJekyllDataNumber{%
1433   \markdownRendererJekyllDataNumberPrototype}%
1434 \ExplSyntaxOn
1435 \seq_put_right:Nn
1436   \g_@@_renderers_seq
1437   { jekyllDataNumber }
1438 \prop_put:Nnn
1439   \g_@@_renderer_arities_prop
1440   { jekyllDataNumber }
1441   { 2 }
1442 \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.

```

1443 \def\markdownRendererJekyllDataString{%
1444   \markdownRendererJekyllDataStringPrototype}%
1445 \ExplSyntaxOn
1446 \seq_put_right:Nn
1447   \g_@@_renderers_seq
1448   { jekyllDataString }
1449 \prop_put:Nnn

```



```

1450 \g_@@_renderer_arities_prop
1451 { jekyllDataString }
1452 { 2 }
1453 \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.

```

1454 \def\markdownRendererJekyllDataEmpty{%
1455 \markdownRendererJekyllDataEmptyPrototype}%
1456 \ExplSyntaxOn
1457 \seq_put_right:Nn
1458 \g_@@_renderers_seq
1459 { jekyllDataEmpty }
1460 \prop_put:Nnn
1461 \g_@@_renderer_arities_prop
1462 { jekyllDataEmpty }
1463 { 1 }
1464 \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.

```

1465 \def\markdownRendererHeadingOne{%
1466 \markdownRendererHeadingOnePrototype}%
1467 \ExplSyntaxOn
1468 \seq_put_right:Nn
1469 \g_@@_renderers_seq
1470 { headingOne }
1471 \prop_put:Nnn
1472 \g_@@_renderer_arities_prop
1473 { headingOne }
1474 { 1 }
1475 \ExplSyntaxOff

```

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

```

1476 \def\markdownRendererHeadingTwo{%
1477 \markdownRendererHeadingTwoPrototype}%
1478 \ExplSyntaxOn
1479 \seq_put_right:Nn
1480 \g_@@_renderers_seq

```

```

1481 { headingTwo }
1482 \prop_put:Nnn
1483 \g_@@_renderer_arities_prop
1484 { headingTwo }
1485 { 1 }
1486 \ExplSyntaxOff

```

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

```

1487 \def\markdownRendererHeadingThree{%
1488   \markdownRendererHeadingThreePrototype}%
1489 \ExplSyntaxOn
1490 \seq_put_right:Nn
1491   \g_@@_renderers_seq
1492   { headingThree }
1493 \prop_put:Nnn
1494   \g_@@_renderer_arities_prop
1495   { headingThree }
1496   { 1 }
1497 \ExplSyntaxOff

```

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

```

1498 \def\markdownRendererHeadingFour{%
1499   \markdownRendererHeadingFourPrototype}%
1500 \ExplSyntaxOn
1501 \seq_put_right:Nn
1502   \g_@@_renderers_seq
1503   { headingFour }
1504 \prop_put:Nnn
1505   \g_@@_renderer_arities_prop
1506   { headingFour }
1507   { 1 }
1508 \ExplSyntaxOff

```

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

```

1509 \def\markdownRendererHeadingFive{%
1510   \markdownRendererHeadingFivePrototype}%
1511 \ExplSyntaxOn
1512 \seq_put_right:Nn
1513   \g_@@_renderers_seq
1514   { headingFive }
1515 \prop_put:Nnn
1516   \g_@@_renderer_arities_prop
1517   { headingFive }
1518   { 1 }

```

```
1519 \ExplSyntaxOff
```

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

```
1520 \def\markdownRendererHeadingSix{%
1521   \markdownRendererHeadingSixPrototype}%
1522 \ExplSyntaxOn
1523 \seq_put_right:Nn
1524   \g_@@_renderers_seq
1525   { headingSix }
1526 \prop_put:Nnn
1527   \g_@@_renderer_arities_prop
1528   { headingSix }
1529   { 1 }
1530 \ExplSyntaxOff
```

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

```
1531 \def\markdownRendererHorizontalRule{%
1532   \markdownRendererHorizontalRulePrototype}%
1533 \ExplSyntaxOn
1534 \seq_put_right:Nn
1535   \g_@@_renderers_seq
1536   { horizontalRule }
1537 \prop_put:Nnn
1538   \g_@@_renderer_arities_prop
1539   { horizontalRule }
1540   { 0 }
1541 \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.

```
1542 \def\markdownRendererFootnote{%
1543   \markdownRendererFootnotePrototype}%
1544 \ExplSyntaxOn
1545 \seq_put_right:Nn
1546   \g_@@_renderers_seq
1547   { footnote }
1548 \prop_put:Nnn
1549   \g_@@_renderer_arities_prop
1550   { footnote }
1551   { 1 }
1552 \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.

```

1553 \def\markdownRendererCite{%
1554   \markdownRendererCitePrototype}%
1555 \ExplSyntaxOn
1556 \seq_put_right:Nn
1557   \g_@@_renderers_seq
1558   { cite }
1559 \prop_put:Nnn
1560   \g_@@_renderer_arities_prop
1561   { cite }
1562   { 1 }
1563 \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.

```

1564 \def\markdownRendererTextCite{%
1565   \markdownRendererTextCitePrototype}%
1566 \ExplSyntaxOn
1567 \seq_put_right:Nn
1568   \g_@@_renderers_seq
1569   { textCite }
1570 \prop_put:Nnn
1571   \g_@@_renderer_arities_prop
1572   { textCite }
1573   { 1 }
1574 \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.

```

1575 \def\markdownRendererTable{%
1576   \markdownRendererTablePrototype}%
1577 \ExplSyntaxOn
1578 \seq_put_right:Nn
1579   \g_@@_renderers_seq
1580   { table }
1581 \prop_put:Nnn
1582   \g_@@_renderer_arities_prop
1583   { table }
1584   { 3 }
1585 \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.

```

1586 \def\markdownRendererInlineHtmlComment{%
1587   \markdownRendererInlineHtmlCommentPrototype}%
1588 \ExplSyntaxOn
1589 \seq_put_right:Nn
1590   \g_@@_renderers_seq
1591   { inlineHtmlComment }
1592 \prop_put:Nnn
1593   \g_@@_renderer_arities_prop
1594   { inlineHtmlComment }
1595   { 1 }
1596 \ExplSyntaxOff
1597 \def\markdownRendererBlockHtmlCommentBegin{%
1598   \markdownRendererBlockHtmlCommentBeginPrototype}%
1599 \ExplSyntaxOn
1600 \seq_put_right:Nn
1601   \g_@@_renderers_seq
1602   { blockHtmlCommentBegin }
1603 \prop_put:Nnn
1604   \g_@@_renderer_arities_prop
1605   { blockHtmlCommentBegin }
1606   { 0 }
1607 \ExplSyntaxOff
1608 \def\markdownRendererBlockHtmlCommentEnd{%

```

```

1609 \markdownRendererBlockHtmlCommentEndPrototype}%
1610 \ExplSyntaxOn
1611 \seq_put_right:Nn
1612   \g_@@_renderers_seq
1613   { blockHtmlCommentEnd }
1614 \prop_put:Nnn
1615   \g_@@_renderer_arities_prop
1616   { blockHtmlCommentEnd }
1617   { 0 }
1618 \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.

```

1619 \def\markdownRendererInlineHtmlTag{%
1620   \markdownRendererInlineHtmlTagPrototype}%
1621 \ExplSyntaxOn
1622 \seq_put_right:Nn
1623   \g_@@_renderers_seq
1624   { inlineHtmlTag }
1625 \prop_put:Nnn
1626   \g_@@_renderer_arities_prop
1627   { inlineHtmlTag }
1628   { 1 }
1629 \ExplSyntaxOff
1630 \def\markdownRendererInputBlockHtmlElement{%
1631   \markdownRendererInputBlockHtmlElementPrototype}%
1632 \ExplSyntaxOn
1633 \seq_put_right:Nn
1634   \g_@@_renderers_seq
1635   { inputBlockHtmlElement }
1636 \prop_put:Nnn
1637   \g_@@_renderer_arities_prop
1638   { inputBlockHtmlElement }
1639   { 1 }
1640 \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  $\langle identifier \rangle$ .

`\markdownRendererAttributeClassName` 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.

```

1641 \def\markdownRendererAttributeIdentifier{%
1642   \markdownRendererAttributeIdentifierPrototype}%
1643 \ExplSyntaxOn
1644 \seq_put_right:Nn
1645   \g_@@_renderers_seq
1646   { attributeIdentifier }
1647 \prop_put:Nnn
1648   \g_@@_renderer_arities_prop
1649   { attributeIdentifier }
1650   { 1 }
1651 \ExplSyntaxOff
1652 \def\markdownRendererAttributeClassName{%
1653   \markdownRendererAttributeClassNamePrototype}%
1654 \ExplSyntaxOn
1655 \seq_put_right:Nn
1656   \g_@@_renderers_seq
1657   { attributeClassName }
1658 \prop_put:Nnn
1659   \g_@@_renderer_arities_prop
1660   { attributeClassName }
1661   { 1 }
1662 \ExplSyntaxOff
1663 \def\markdownRendererAttributeKeyValue{%
1664   \markdownRendererAttributeKeyValuePrototype}%
1665 \ExplSyntaxOn
1666 \seq_put_right:Nn
1667   \g_@@_renderers_seq
1668   { attributeKeyValue }
1669 \prop_put:Nnn
1670   \g_@@_renderer_arities_prop
1671   { attributeKeyValue }
1672   { 2 }
1673 \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.

```

1674 \def\markdownRendererHeaderAttributeContextBegin{%
1675   \markdownRendererHeaderAttributeContextBeginPrototype}%
1676 \ExplSyntaxOn
1677 \seq_put_right:Nn
1678   \g_@@_renderers_seq
1679   { headerAttributeContextBegin }
1680 \prop_put:Nnn
1681   \g_@@_renderer_arities_prop
1682   { headerAttributeContextBegin }
1683   { 0 }
1684 \ExplSyntaxOff
1685 \def\markdownRendererHeaderAttributeContextEnd{%
1686   \markdownRendererHeaderAttributeContextEndPrototype}%
1687 \ExplSyntaxOn
1688 \seq_put_right:Nn
1689   \g_@@_renderers_seq
1690   { headerAttributeContextEnd }
1691 \prop_put:Nnn
1692   \g_@@_renderer_arities_prop
1693   { headerAttributeContextEnd }
1694   { 0 }
1695 \ExplSyntaxOff

```

**2.2.3.29 Strike-Through Renderer** The `\markdownRendererStrikeThrough` macro represents a strike-through span of text. The macro receives a single argument that corresponds to the striked-out span of text. This macro will only be produced, when the `strikeThrough` option is enabled.

```

1696 \def\markdownRendererStrikeThrough{%
1697   \markdownRendererStrikeThroughPrototype}%
1698 \ExplSyntaxOn
1699 \seq_put_right:Nn
1700   \g_@@_renderers_seq
1701   { strikeThrough }
1702 \prop_put:Nnn
1703   \g_@@_renderer_arities_prop
1704   { strikeThrough }
1705   { 1 }
1706 \ExplSyntaxOff

```



**2.2.3.30 Superscript Renderer** The `\markdownRendererSuperscript` macro represents a superscript span of text. The macro receives a single argument that corresponds to the superscript span of text. This macro will only be produced, when the `superscripts` option is enabled.

```

1707 \def\markdownRendererSuperscript{%
1708   \markdownRendererSuperscriptPrototype}%
1709 \ExplSyntaxOn
1710 \seq_put_right:Nn
1711   \g_@@_renderers_seq
1712   { superscript }
1713 \prop_put:Nnn
1714   \g_@@_renderer_arities_prop
1715   { superscript }
1716   { 1 }
1717 \ExplSyntaxOff

```

**2.2.3.31 Subscript Renderer** The `\markdownRendererSubscript` macro represents a subscript span of text. The macro receives a single argument that corresponds to the subscript span of text. This macro will only be produced, when the `subscripts` option is enabled.

```

1718 \def\markdownRendererSubscript{%
1719   \markdownRendererSubscriptPrototype}%
1720 \ExplSyntaxOn
1721 \seq_put_right:Nn
1722   \g_@@_renderers_seq
1723   { subscript }
1724 \prop_put:Nnn
1725   \g_@@_renderer_arities_prop
1726   { subscript }
1727   { 1 }
1728 \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  $\text{\LaTeX}$ 3 kernel.

```

1729 \ExplSyntaxOn
1730 \keys_define:nn
1731   { markdown/jekyllData }
1732   { }
1733 \ExplSyntaxOff

```

The following  $\text{\TeX}$  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 L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt implementations (see sections 3.3 and 3.4).

```

1734 \ExplSyntaxOn
1735 \cs_new:Nn \@@_plaintex_define_renderer_prototypes:
1736 {
1737   \seq_map_function:NN
1738     \g_@@_renderers_seq
1739     \@@_plaintex_define_renderer_prototype:n
1740   \let\markdownRendererBlockHtmlCommentBeginPrototype=\iffalse
1741   \let\markdownRendererBlockHtmlCommentBegin=\iffalse
1742   \let\markdownRendererBlockHtmlCommentEndPrototype=\fi
1743   \let\markdownRendererBlockHtmlCommentEnd=\fi
1744 }
1745 \cs_new:Nn \@@_plaintex_define_renderer_prototype:n
1746 {
1747   \@@_renderer_prototype_tl_to_csname:nN
1748     { #1 }
1749     \l_tmpa_tl
1750   \prop_get:NnN
1751     \g_@@_renderer_arities_prop
1752     { #1 }
1753     \l_tmpb_tl
1754   \@@_plaintex_define_renderer_prototype:cV
1755     { \l_tmpa_tl }
1756     \l_tmpb_tl
1757 }
1758 \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
1759 {
1760   \tl_set:Nn
1761     \l_tmpa_tl
1762     { \str_uppercase:n { #1 } }
1763   \tl_set:Nx
1764     #2
1765     {
1766       markdownRenderer
1767       \tl_head:f { \l_tmpa_tl }
1768       \tl_tail:n { #1 }
1769       Prototype
1770     }
1771 }
1772 \cs_new:Nn \@@_plaintex_define_renderer_prototype:Nn
1773 {
1774   \cs_generate_from_arg_count:Nnn
1775     #1
1776     \cs_set:Npn
1777     { #2 }

```

```

1778     { }
1779 }
1780 \cs_generate_variant:Nn
1781   \@@_plaintex_define_renderer_prototype:Nn
1782   { cV }
1783 \@@_plaintex_define_renderer_prototypes:
1784 \ExplSyntaxOff

```

### 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 T<sub>E</sub>X engine that does not support direct Lua access is starting to buffer a text. The plain T<sub>E</sub>X implementation changes the category code of plain T<sub>E</sub>X 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*.

```

1785 \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 T<sub>E</sub>X 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.

```

1786 \let\markdownReadAndConvert\relax
1787 \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*.

```

1788   \catcode`\|=0\catcode`\=12%
1789   |gdef|markdownBegin{%
1790     |markdownReadAndConvert{\markdownEnd}%
1791                               {|markdownEnd}}%
1792 |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 3.2.6),

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
- `3` – The `lt3luabridge` Lua package

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.

The `\markdownMode` macro has been deprecated and will be removed in Markdown 3.0.0. The code that corresponds to `\markdownMode` value of `3` will be the only implementation.

```

1793 \ExplSyntaxOn
1794 \cs_if_exist:NF
1795   \markdownMode
1796   {
1797     \file_if_exist:nTF
1798       { lt3luabridge.tex }
1799       {
1800         \cs_new:Npn
1801           \markdownMode
1802           { 3 }
1803       }
1804     {
1805       \cs_if_exist:NTF
1806         \directlua
1807         {
1808           \cs_new:Npn
1809             \markdownMode
1810             { 2 }
1811         }
1812       {
1813         \cs_new:Npn
1814           \markdownMode
1815           { 0 }
1816       }
1817     }
1818   }

```

The `\markdownLuaRegisterIBCallback` and `\markdownLuaUnregisterIBCallback` macros have been deprecated and will be removed in Markdown 3.0.0:

```

1819 \def\markdownLuaRegisterIBCallback#1{\relax}%

```

```
1820 \def\markdownLuaUnregisterIBCallback#1{\relax}%
```

## 2.3 L<sup>A</sup>T<sub>E</sub>X Interface

The L<sup>A</sup>T<sub>E</sub>X interface provides L<sup>A</sup>T<sub>E</sub>X environments for the typesetting of markdown input from within L<sup>A</sup>T<sub>E</sub>X, facilities for setting Lua, plain T<sub>E</sub>X, and L<sup>A</sup>T<sub>E</sub>X options used during the conversion from markdown to plain T<sub>E</sub>X, and facilities for changing the way markdown tokens are rendered. The rest of the interface is inherited from the plain T<sub>E</sub>X interface (see Section 2.2).

The L<sup>A</sup>T<sub>E</sub>X implementation redefines the plain T<sub>E</sub>X logging macros (see Section 3.2.1) to use the L<sup>A</sup>T<sub>E</sub>X `\PackageInfo`, `\PackageWarning`, and `\PackageError` macros.

```
1821 \newcommand\markdownInfo[1]{\PackageInfo{markdown}{#1}}%
1822 \newcommand\markdownWarning[1]{\PackageWarning{markdown}{#1}}%
1823 \newcommand\markdownError[2]{\PackageError{markdown}{#1}{#2.}}%
1824 \input markdown/markdown
```

The L<sup>A</sup>T<sub>E</sub>X interface is implemented by the `markdown.sty` file, which can be loaded from the L<sup>A</sup>T<sub>E</sub>X document preamble as follows:

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

where `<options>` are the L<sup>A</sup>T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> parses package options.

### 2.3.1 Typesetting Markdown

The interface exposes the `markdown` and `markdown*` L<sup>A</sup>T<sub>E</sub>X 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.

```
1825 \newenvironment{markdown}\relax\relax
1826 \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_ **world** ... \end{markdown} % ... \end{document} </pre> | <pre> \documentclass{article} \usepackage{markdown} \begin{document} % ... \begin{markdown*}{smartEllipses} _Hello_ **world** ... \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  $\text{\TeX}$ . Unlike the `\markdownInput` macro provided by the plain  $\text{\TeX}$  interface, this macro also accepts  $\text{\LaTeX}$  interface options (see Section 2.3.2) as its optional argument. These options will only influence this markdown document.

The following example  $\text{\LaTeX}$  code showcases the usage of the `\markdownInput` macro:

```

\documentclass{article}
\usepackage{markdown}
\begin{document}
\markdownInput[smartEllipses]{hello.md}
\end{document}

```

### 2.3.2 Options

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

The  $\text{\LaTeX}$  options may be specified when loading the  $\text{\LaTeX}$  package, when using the `markdown*`  $\text{\LaTeX}$  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:

1827 `\ExplSyntaxOn`

```

1828 \cs_new:Nn
1829   \@@_setup:n
1830   {
1831     \keys_set:nn
1832       { markdown/latex-options }
1833       { #1 }
1834   }
1835 \let\markdownSetup=\@@_setup:n
1836 \ExplSyntaxOff

```

We may also store  $\text{\LaTeX}$  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:

```

1837 \newcommand\markdownSetupSnippet[2]{%
1838   \markdownIfSnippetExists{#1}%
1839   {%
1840     \markdownWarning
1841       {Redefined setup snippet \markdownLaTeXThemeName#1}%
1842     \csname markdownLaTeXSetupSnippet%
1843       \markdownLaTeXThemeName#1\endcsname={#2}%
1844   }{%
1845     \newtoks\next
1846     \next={#2}%
1847     \expandafter\let\csname markdownLaTeXSetupSnippet%
1848       \markdownLaTeXThemeName#1\endcsname=\next
1849   }}%

```

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

```

1850 \newcommand\markdownIfSnippetExists[3]{%
1851   \@ifundefined
1852     {markdownLaTeXSetupSnippet\markdownLaTeXThemeName#1}%
1853     {#3}{#2}}%

```

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

To enable the enumeration of  $\text{\LaTeX}$  options, we will maintain the `\g_@@_latex_options_seq` sequence.

```

1854 \ExplSyntaxOn
1855 \seq_new:N \g_@@_latex_options_seq

```

To enable the reflection of default  $\text{\LaTeX}$  options and their types, we will maintain the `\g_@@_default_latex_options_prop` and `\g_@@_latex_option_types_prop` property lists, respectively.

```

1856 \prop_new:N \g_@@_latex_option_types_prop
1857 \prop_new:N \g_@@_default_latex_options_prop
1858 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
1859 \seq_put_right:NV \g_@@_option_layers_seq \c_@@_option_layer_latex_tl

```

```

1860 \cs_new:Nn
1861   \@@_add_latex_option:nnn
1862   {
1863     \@@_add_option:Vnnn
1864     \c_@@_option_layer_latex_tl
1865     { #1 }
1866     { #2 }
1867     { #3 }
1868   }

```

**2.3.2.1 No default token renderer prototypes** Default token renderer prototypes require L<sup>A</sup>T<sub>E</sub>X 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 T<sub>E</sub>X implementation (see Section 3.2.2) and prevent the soft L<sup>A</sup>T<sub>E</sub>X prerequisites in Section 1.1.3 from being loaded: The plain option must be set before or when loading the package. Setting the option after loading the package will have no effect.

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

```

1869 \@@_add_latex_option:nnn
1870   { plain }
1871   { boolean }
1872   { false }
1873 \ExplSyntaxOff

```

**2.3.2.2 L<sup>A</sup>T<sub>E</sub>X themes** User-contributed L<sup>A</sup>T<sub>E</sub>X themes for the Markdown package provide a domain-specific interpretation of some Markdown tokens. Similarly to L<sup>A</sup>T<sub>E</sub>X packages, themes allow the authors to achieve a specific look and other high-level goals without low-level programming.

The L<sup>A</sup>T<sub>E</sub>X option with key `theme` loads a L<sup>A</sup>T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X document class or for a single L<sup>A</sup>T<sub>E</sub>X 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  $\text{\LaTeX}$  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  $\text{\LaTeX}$  package named `markdownthemewitiko_beamer_MU.sty`.

If the  $\text{\LaTeX}$  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  $\text{\LaTeX}$  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`  $\text{\LaTeX}$  package, and finally the `markdownthemewitiko_dot.sty`  $\text{\LaTeX}$  package:

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

```
1874 \newif\ifmarkdownLaTeXLoaded
1875 \markdownLaTeXLoadedfalse
1876 \AtEndOfPackage{\markdownLaTeXLoadedtrue}
1877 \ExplSyntaxOn
1878 \tl_new:N \markdownLaTeXThemePackageName
1879 \cs_new:Nn
1880 \@@_set_latex_theme:n
1881 {
1882   \str_if_in:NnF
1883     { #1 }
1884     { / }
1885     {
1886       \markdownError
1887       { Won't load theme with unqualified name #1 }
1888       { Theme names must contain at least one forward slash }
1889     }
1890   \tl_set:Nn \markdownLaTeXThemePackageName { #1 }
1891   \str_replace_all:Nnn
1892     \markdownLaTeXThemePackageName
1893     { / }
1894     { _ }
1895   \edef\markdownLaTeXThemePackageName{
```

```

1896     markdowntheme\markdownLaTeXThemePackageName}
1897     \expandafter\markdownLaTeXThemeLoad\expandafter{
1898     \markdownLaTeXThemePackageName}{#1/}
1899   }
1900 \keys_define:nn
1901   { markdown/latex-options }
1902   {
1903     theme .code:n = { \@@_set_latex_theme:n { #1 } },
1904   }
1905 \ExplSyntaxOff

```

The  $\text{\LaTeX}$  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  $\text{\LaTeX}$ , themes may not be loaded after the beginning of a  $\text{\LaTeX}$  document.

```

1906 \ExplSyntaxOn
1907 \@onlypreamble
1908   \@@_set_latex_theme:n
1909 \ExplSyntaxOff

```

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;

```

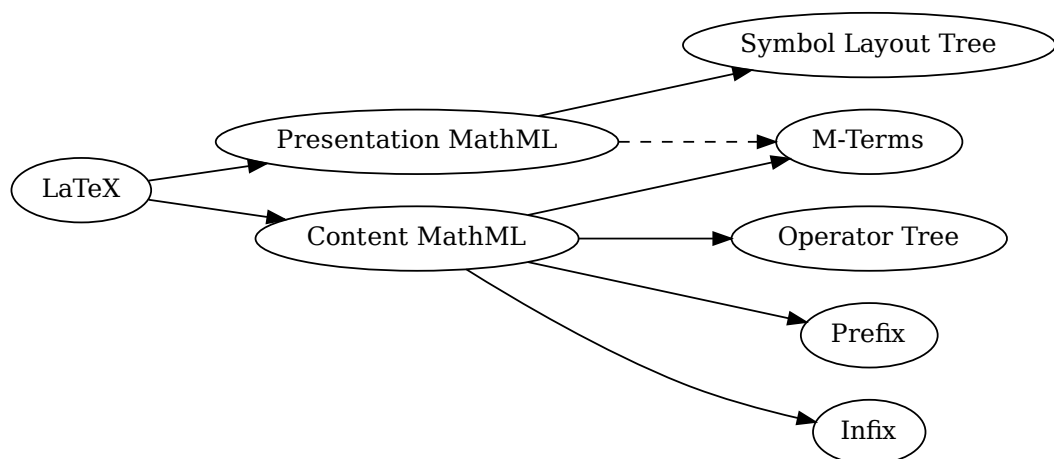
```

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.



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

The theme requires a Unix-like operating system with GNU Diffutils and Graphviz installed. The theme also requires shell access unless the `\markdownOptionFrozenCache` plain  $\TeX$  option is enabled.

1910 \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}
\begin{document}
\begin{markdown}

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

Typesetting the above document produces the output shown in Figure 5. The

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



# 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

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

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

theme requires the catchfile L<sup>A</sup>T<sub>E</sub>X 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 T<sub>E</sub>X option is enabled.

1911 \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 disabled.

```

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

```

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

```
1912 \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>*:

```

1913 \ExplSyntaxOn
1914 \keys_define:nn
1915   { markdown/latex-options }
1916   {
1917     snippet .code:n = {
1918       \markdownIfSnippetExists{#1}
1919       {
1920         \expandafter\markdownSetup\expandafter{
1921           \the\csname markdownLaTeXSetupSnippet
1922             \markdownLaTeXThemeName#1\endcsname}
1923       }{
1924         \markdownError
1925           {Can't~invoke~setup~snippet~#1}
1926           {The~setup~snippet~is~undefined}
1927       }
1928     }
1929   }
1930 \ExplSyntaxOff

```

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** Here, we automatically define plain T<sub>E</sub>X macros and the  $\langle key \rangle = \langle value \rangle$  interface for the above L<sup>A</sup>T<sub>E</sub>X options.

```
1931 \ExplSyntaxOn
1932 \cs_new:Nn \@@_latex_define_option_commands_and_keyvals:
1933 {
1934   \seq_map_inline:Nn
1935     \g_@@_latex_options_seq
1936     {
1937       \@@_plain_tex_define_option_command:n
1938       { ##1 }
1939     }

```

Furthermore, we also define the  $\langle key \rangle = \langle value \rangle$  interface for all option macros recognized by the Lua and plain T<sub>E</sub>X interfaces.

```
1940   \seq_map_inline:Nn
1941     \g_@@_option_layers_seq
1942     {
1943       \seq_map_inline:cn
1944         { g_@@_ ##1 _options_seq }
1945         {
1946           \@@_latex_define_option_keyval:nn
1947             { ##1 }
1948             { #####1 }
1949         }
1950     }
1951 }
1952 \cs_new:Nn \@@_latex_define_option_keyval:nn
1953 {

```

```

1954 \prop_get:cnN
1955 { g_@@_ #1 _option_types_prop }
1956 { #2 }
1957 \l_tmpa_tl
1958 \keys_define:nn
1959 { markdown/latex-options }
1960 {
1961     #2 .code:n = {
1962         \@@_set_option_value:nn
1963         { #2 }
1964         { ##1 }
1965     },
1966 }
1967 \str_if_eq:VVT
1968 \l_tmpa_tl
1969 \c_@@_option_type_boolean_tl
1970 {
1971     \keys_define:nn
1972     { markdown/latex-options }
1973     {
1974         #2 .default:n = { true },
1975     }
1976 }
1977 }
1978 \@@_latex_define_option_commands_and_keyvals:
1979 \ExplSyntaxOff

```

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.

```

1980 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
1981 \DeclareOption{frozenscache}{\markdownSetup{frozenCache}}

```

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

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

**2.3.2.5 Plain  $\text{\TeX}$  Markdown Token Renderers** The  $\text{\LaTeX}$  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  $\text{\TeX}$  interface (see Section 2.2.3).

```
1982 \ExplSyntaxOn
1983 \cs_new:Nn \@@_latex_define_renderers:
1984 {
1985   \seq_map_function:NN
1986     \g_@@_renderers_seq
1987     \@@_latex_define_renderer:n
1988 }
1989 \cs_new:Nn \@@_latex_define_renderer:n
1990 {
1991   \@@_renderer_tl_to_csname:nN
1992     { #1 }
1993     \l_tmpa_tl
1994   \prop_get:NnN
1995     \g_@@_renderer_arities_prop
1996     { #1 }
1997     \l_tmpb_tl
1998   \@@_latex_define_renderer:ncV
1999     { #1 }
2000     { \l_tmpa_tl }
2001     \l_tmpb_tl
2002 }
2003 \cs_new:Nn \@@_renderer_tl_to_csname:nN
2004 {
2005   \tl_set:Nn
2006     \l_tmpa_tl
2007     { \str_uppercase:n { #1 } }
2008   \tl_set:Nx
2009     #2
2010     {
2011       markdownRenderer
2012       \tl_head:f { \l_tmpa_tl }

```



```

2013     \tl_tail:n { #1 }
2014   }
2015 }
2016 \cs_new:Nn \@@_latex_define_renderer:nNn
2017 {
2018   \keys_define:nn
2019     { markdown/latex-options/renderers }
2020   {
2021     #1 .code:n = {
2022       \cs_generate_from_arg_count:NNnn
2023         #2
2024         \cs_set:Npn
2025           { #3 }
2026           { ##1 }
2027     },
2028   }
2029 }
2030 \cs_generate_variant:Nn
2031   \@@_latex_define_renderer:nNn
2032   { ncV }
2033 \ExplSyntaxOff

```

The following example  $\text{\LaTeX}$  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  $\text{\TeX}$  Markdown Token Renderer Prototypes** The  $\text{\LaTeX}$  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  $\text{\TeX}$  interface (see Section 2.2.4).

```

2034 \ExplSyntaxOn
2035 \cs_new:Nn \@@_latex_define_renderer_prototypes:
2036 {
2037   \seq_map_function:NN
2038     \g_@@_renderers_seq
2039     \@@_latex_define_renderer_prototype:n
2040 }
2041 \cs_new:Nn \@@_latex_define_renderer_prototype:n

```

```

2042 {
2043   \@@_renderer_prototype_tl_to_csname:nN
2044   { #1 }
2045   \l_tmpa_tl
2046   \prop_get:NnN
2047   \g_@@_renderer_arities_prop
2048   { #1 }
2049   \l_tmpb_tl
2050   \@@_latex_define_renderer_prototype:ncV
2051   { #1 }
2052   { \l_tmpa_tl }
2053   \l_tmpb_tl
2054 }
2055 \cs_new:Nn \@@_latex_define_renderer_prototype:nNn
2056 {
2057   \keys_define:nn
2058   { markdown/latex-options/renderer-prototypes }
2059   {
2060     #1 .code:n = {
2061       \cs_generate_from_arg_count:NNnn
2062       #2
2063       \cs_set:Npn
2064       { #3 }
2065       { ##1 }
2066     },
2067   }
2068 }
2069 \cs_generate_variant:Nn
2070 \@@_latex_define_renderer_prototype:nNn
2071 { ncV }
2072 \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 and facilities for setting Lua, plain T<sub>E</sub>X, and ConT<sub>E</sub>Xt options used during the conversion from markdown to plain T<sub>E</sub>X. The rest of the interface is inherited from the plain T<sub>E</sub>X interface (see Section 2.2).

```
2073 \writestatus{loading}{ConTEXt User Module / markdown}%
2074 \startmodule[markdown]
2075 \unprotect
```

The ConT<sub>E</sub>Xt implementation redefines the plain T<sub>E</sub>X logging macros (see Section 3.2.1) to use the ConT<sub>E</sub>Xt `\writestatus` macro.

```
2076 \def\markdownInfo#1{\writestatus{markdown}{#1.}}%
2077 \def\markdownWarning#1{\writestatus{markdown\space warn}{#1.}}%
2078 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
2079 \do\#\do\^\do\_ \do\% \do\~}%
2080 \input markdown/markdown
```

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, and defines the `\inputmarkdown` command.

```
2081 \let\startmarkdown\relax
2082 \let\stopmarkdown\relax
2083 \let\inputmarkdown\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
```

The `\inputmarkdown` 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  $\text{\TeX}$ . Unlike the `\markdownInput` macro provided by the plain  $\text{\TeX}$  interface, this macro also accepts  $\text{\ConTeXt}$  interface options (see Section 2.4.2) as its optional argument. These options will only influence this markdown document.

The following example  $\text{\LaTeX}$  code showcases the usage of the `\markdownInput` macro:

```
\usemodule[t][markdown]
\starttext
\inputmarkdown[smartEllipses]{hello.md}
\stoptext
```

### 2.4.2 Options

The  $\text{\ConTeXt}$  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  $\langle key \rangle = \text{true}$  (or, equivalently,  $\langle key \rangle = \text{yes}$ ) if the  $= \langle value \rangle$  part has been omitted.

$\text{\ConTeXt}$  options map directly to the options recognized by the plain  $\text{\TeX}$  interface (see Section 2.2.2).

The  $\text{\ConTeXt}$  options may be specified when using the `\inputmarkdown` macro (see Section 2.4), or via the `\setupmarkdown` macro. The `\setupmarkdown` macro receives the options to set up as its only argument:

```
2084 \ExplSyntaxOn
2085 \cs_new:Nn
2086   \@@_setup:n
2087   {
2088     \keys_set:nn
2089       { markdown/context-options }
2090       { #1 }
2091   }
2092 \long\def\setupmarkdown[#1]
2093   {
2094     \@@_setup:n
2095       { #1 }
2096   }
2097 \ExplSyntaxOff
```

**2.4.2.1 ConTeXt Interface Options** We define the  $\langle key \rangle = \langle value \rangle$  interface for all option macros recognized by the Lua and plain T<sub>E</sub>X interfaces.

```

2098 \ExplSyntaxOn
2099 \cs_new:Nn \@@_context_define_option_commands_and_keyvals:
2100 {
2101   \seq_map_inline:Nn
2102     \g_@@_option_layers_seq
2103     {
2104       \seq_map_inline:cn
2105         { g_@@_ ##1 _options_seq }
2106         {
2107           \@@_context_define_option_keyval:nn
2108             { ##1 }
2109             { ####1 }
2110         }
2111     }
2112 }
2113 \cs_new:Nn \@@_context_define_option_keyval:nn
2114 {
2115   \prop_get:cnN
2116     { g_@@_ #1 _option_types_prop }
2117     { #2 }
2118   \l_tmpa_tl
2119   \keys_define:nn
2120     { markdown/context-options }
2121     {
2122       #2 .code:n = {
2123         \tl_set:Nx
2124           \l_tmpa_tl
2125           {
2126             \str_case:nnF
2127               { ##1 }
2128               {
2129                 { yes } { true }
2130                 { no } { false }
2131               }
2132             { ##1 }
2133           }
2134         \@@_set_option_value:nV
2135           { #2 }
2136           \l_tmpa_tl
2137       },
2138     }
2139   \str_if_eq:VVT
2140     \l_tmpa_tl
2141     \c_@@_option_type_boolean_tl
2142     {

```

```

2143     \keys_define:nn
2144     { markdown/context-options }
2145     {
2146         #2 .default:n = { true },
2147     }
2148 }
2149 }
2150 \cs_generate_variant:Nn
2151 \@@_set_option_value:nn
2152 { nV }
2153 \@@_context_define_option_commands_and_keyvals:
2154 \ExplSyntaxOff

```

## 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  $\text{\TeX}$  *token renderers* is performed by the Lua layer. The plain  $\text{\TeX}$  layer provides default definitions for the token renderers. The  $\text{\LaTeX}$  and  $\text{Con}\text{\TeX}$ t layers correct idiosyncrasies of the respective  $\text{\TeX}$  formats, and provide format-specific default definitions for the token renderers.

### 3.1 Lua Implementation

The Lua implementation implements **writer** and **reader** objects, which provide the conversion from markdown to plain  $\text{\TeX}$ , and **extensions** objects, which provide syntax extensions for the **writer** and **reader** objects.

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  $\text{\TeX}$  writer were hidden behind the converter functions exposed by the Lua interface (see Section 2.1).

```

2155 local upper, gsub, format, length =
2156     string.upper, string.gsub, string.format, string.len
2157 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, any =
2158     lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
2159     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  $\text{\TeX}$  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.

```
2160 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.

```
2161 function util.err(msg, exit_code)
2162   io.stderr:write("markdown.lua: " .. msg .. "\n")
2163   os.exit(exit_code or 1)
2164 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.

```
2165 function util.cache(dir, string, salt, transform, suffix)
2166   local digest = md5.sumhexa(string .. (salt or ""))
2167   local name = util.pathname(dir, digest .. suffix)
2168   local file = io.open(name, "r")
2169   if file == nil then -- If no cache entry exists, then create a new one.
2170     file = assert(io.open(name, "w"),
2171       [[could not open file ]] .. name .. [[ for writing]])
2172     local result = string
2173     if transform ~= nil then
2174       result = transform(result)
2175     end
2176     assert(file:write(result))
2177     assert(file:close())
2178   end
2179   return name
2180 end
```

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

```
2181 function util.table_copy(t)
2182   local u = { }
2183   for k, v in pairs(t) do u[k] = v end
2184   return setmetatable(u, getmetatable(t))
2185 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 Ierusalimschy [9, Chapter 21].

```
2186 function util.expand_tabs_in_line(s, tabstop)
2187   local tab = tabstop or 4
2188   local corr = 0
2189   return (s:gsub("\t", function(p)
```

```

2190         local sp = tab - (p - 1 + corr) % tab
2191         corr = corr - 1 + sp
2192         return string.rep(" ", sp)
2193     end))
2194 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.

```

2195 function util.walk(t, f)
2196     local typ = type(t)
2197     if typ == "string" then
2198         f(t)
2199     elseif typ == "table" then
2200         local i = 1
2201         local n
2202         n = t[i]
2203         while n do
2204             util.walk(n, f)
2205             i = i + 1
2206             n = t[i]
2207         end
2208     elseif typ == "function" then
2209         local ok, val = pcall(t)
2210         if ok then
2211             util.walk(val, f)
2212         end
2213     else
2214         f(tostring(t))
2215     end
2216 end

```

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

```

2217 function util.flatten(ary)
2218     local new = {}
2219     for _,v in ipairs(ary) do
2220         if type(v) == "table" then
2221             for _,w in ipairs(util.flatten(v)) do
2222                 new[#new + 1] = w
2223             end
2224         else
2225             new[#new + 1] = v
2226         end
2227     end
2228     return new
2229 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.

```
2230 function util.rope_to_string(rope)
2231   local buffer = {}
2232   util.walk(rope, function(x) buffer[#buffer + 1] = x end)
2233   return table.concat(buffer)
2234 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.

```
2235 function util.rope_last(rope)
2236   if #rope == 0 then
2237     return nil
2238   else
2239     local l = rope[#rope]
2240     if type(l) == "table" then
2241       return util.rope_last(l)
2242     else
2243       return l
2244     end
2245   end
2246 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  $1 \leq i \leq \#ary$ .

```
2247 function util.intersperse(ary, x)
2248   local new = {}
2249   local l = #ary
2250   for i,v in ipairs(ary) do
2251     local n = #new
2252     new[n + 1] = v
2253     if i ~= l then
2254       new[n + 2] = x
2255     end
2256   end
2257   return new
2258 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  $1 \leq i \leq \#ary$ .

```
2259 function util.map(ary, f)
2260   local new = {}
2261   for i,v in ipairs(ary) do
2262     new[i] = f(v)
2263   end
2264   return new
2265 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.

```
2266 function util.escaper(char_escapes, string_escapes)
```

Build a string of escapable characters.

```
2267   local char_escapes_list = ""
2268   for i,_ in pairs(char_escapes) do
2269     char_escapes_list = char_escapes_list .. i
2270   end
```

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

```
2271   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.

```
2272   if string_escapes then
2273     for k,v in pairs(string_escapes) do
2274       escapable = P(k) / v + escapable
2275     end
2276   end
```

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

```
2277   local escape_string = Cs((escapable + any)^0)
```

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

```
2278   return function(s)
2279     return lpeg.match(escape_string, s)
2280   end
2281 end
```

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

```
2282 function util.pathname(dir, file)
2283   if #dir == 0 then
2284     return file
```

```

2285     else
2286         return dir .. "/" .. file
2287     end
2288 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.

```

2289 local entities = {}
2290
2291 local character_entities = {
2292     ["Tab"] = 9,
2293     ["NewLine"] = 10,
2294     ["excl"] = 33,
2295     ["quot"] = 34,
2296     ["QUOT"] = 34,
2297     ["num"] = 35,
2298     ["dollar"] = 36,
2299     ["percnt"] = 37,
2300     ["amp"] = 38,
2301     ["AMP"] = 38,
2302     ["apos"] = 39,
2303     ["lpar"] = 40,
2304     ["rpar"] = 41,
2305     ["ast"] = 42,
2306     ["midast"] = 42,
2307     ["plus"] = 43,
2308     ["comma"] = 44,
2309     ["period"] = 46,
2310     ["sol"] = 47,
2311     ["colon"] = 58,
2312     ["semi"] = 59,
2313     ["lt"] = 60,
2314     ["LT"] = 60,
2315     ["equals"] = 61,
2316     ["gt"] = 62,
2317     ["GT"] = 62,
2318     ["quest"] = 63,
2319     ["commat"] = 64,
2320     ["lsqb"] = 91,
2321     ["lbrack"] = 91,
2322     ["bsol"] = 92,
2323     ["rsqb"] = 93,
2324     ["rbrack"] = 93,

```

2325 ["Hat"] = 94,  
 2326 ["lowbar"] = 95,  
 2327 ["grave"] = 96,  
 2328 ["DiacriticalGrave"] = 96,  
 2329 ["lcub"] = 123,  
 2330 ["lbrace"] = 123,  
 2331 ["verbar"] = 124,  
 2332 ["vert"] = 124,  
 2333 ["VerticalLine"] = 124,  
 2334 ["rcub"] = 125,  
 2335 ["rbrace"] = 125,  
 2336 ["nbsp"] = 160,  
 2337 ["NonBreakingSpace"] = 160,  
 2338 ["iexcl"] = 161,  
 2339 ["cent"] = 162,  
 2340 ["pound"] = 163,  
 2341 ["curren"] = 164,  
 2342 ["yen"] = 165,  
 2343 ["brvbar"] = 166,  
 2344 ["sect"] = 167,  
 2345 ["Dot"] = 168,  
 2346 ["die"] = 168,  
 2347 ["DoubleDot"] = 168,  
 2348 ["uml"] = 168,  
 2349 ["copy"] = 169,  
 2350 ["COPY"] = 169,  
 2351 ["ordf"] = 170,  
 2352 ["laquo"] = 171,  
 2353 ["not"] = 172,  
 2354 ["shy"] = 173,  
 2355 ["reg"] = 174,  
 2356 ["circledR"] = 174,  
 2357 ["REG"] = 174,  
 2358 ["macr"] = 175,  
 2359 ["OverBar"] = 175,  
 2360 ["strns"] = 175,  
 2361 ["deg"] = 176,  
 2362 ["plusmn"] = 177,  
 2363 ["pm"] = 177,  
 2364 ["PlusMinus"] = 177,  
 2365 ["sup2"] = 178,  
 2366 ["sup3"] = 179,  
 2367 ["acute"] = 180,  
 2368 ["DiacriticalAcute"] = 180,  
 2369 ["micro"] = 181,  
 2370 ["para"] = 182,  
 2371 ["middot"] = 183,

2372 ["centerdot"] = 183,  
 2373 ["CenterDot"] = 183,  
 2374 ["cedil"] = 184,  
 2375 ["Cedilla"] = 184,  
 2376 ["sup1"] = 185,  
 2377 ["ordm"] = 186,  
 2378 ["raquo"] = 187,  
 2379 ["frac14"] = 188,  
 2380 ["frac12"] = 189,  
 2381 ["half"] = 189,  
 2382 ["frac34"] = 190,  
 2383 ["iquest"] = 191,  
 2384 ["Agrave"] = 192,  
 2385 ["Aacute"] = 193,  
 2386 ["Acirc"] = 194,  
 2387 ["Atilde"] = 195,  
 2388 ["Auml"] = 196,  
 2389 ["Aring"] = 197,  
 2390 ["AElig"] = 198,  
 2391 ["Ccedil"] = 199,  
 2392 ["Egrave"] = 200,  
 2393 ["Eacute"] = 201,  
 2394 ["Ecirc"] = 202,  
 2395 ["Euml"] = 203,  
 2396 ["Igrave"] = 204,  
 2397 ["Iacute"] = 205,  
 2398 ["Icirc"] = 206,  
 2399 ["Iuml"] = 207,  
 2400 ["ETH"] = 208,  
 2401 ["Ntilde"] = 209,  
 2402 ["Ograve"] = 210,  
 2403 ["Oacute"] = 211,  
 2404 ["Ocirc"] = 212,  
 2405 ["Otilde"] = 213,  
 2406 ["Ouml"] = 214,  
 2407 ["times"] = 215,  
 2408 ["Oslash"] = 216,  
 2409 ["Ugrave"] = 217,  
 2410 ["Uacute"] = 218,  
 2411 ["Ucirc"] = 219,  
 2412 ["Uuml"] = 220,  
 2413 ["Yacute"] = 221,  
 2414 ["THORN"] = 222,  
 2415 ["szlig"] = 223,  
 2416 ["agrave"] = 224,  
 2417 ["aacute"] = 225,  
 2418 ["acirc"] = 226,

2419 ["atilde"] = 227,  
 2420 ["auml"] = 228,  
 2421 ["aring"] = 229,  
 2422 ["aelig"] = 230,  
 2423 ["ccedil"] = 231,  
 2424 ["egrave"] = 232,  
 2425 ["eacute"] = 233,  
 2426 ["ecirc"] = 234,  
 2427 ["euml"] = 235,  
 2428 ["igrave"] = 236,  
 2429 ["iacute"] = 237,  
 2430 ["icirc"] = 238,  
 2431 ["iuml"] = 239,  
 2432 ["eth"] = 240,  
 2433 ["ntilde"] = 241,  
 2434 ["ograve"] = 242,  
 2435 ["oacute"] = 243,  
 2436 ["ocirc"] = 244,  
 2437 ["otilde"] = 245,  
 2438 ["ouml"] = 246,  
 2439 ["divide"] = 247,  
 2440 ["div"] = 247,  
 2441 ["oslash"] = 248,  
 2442 ["ugrave"] = 249,  
 2443 ["uacute"] = 250,  
 2444 ["ucirc"] = 251,  
 2445 ["uuml"] = 252,  
 2446 ["yacute"] = 253,  
 2447 ["thorn"] = 254,  
 2448 ["yuml"] = 255,  
 2449 ["Amacr"] = 256,  
 2450 ["amacr"] = 257,  
 2451 ["Abreve"] = 258,  
 2452 ["abreve"] = 259,  
 2453 ["Aogon"] = 260,  
 2454 ["aogon"] = 261,  
 2455 ["Cacute"] = 262,  
 2456 ["cacute"] = 263,  
 2457 ["Ccirc"] = 264,  
 2458 ["ccirc"] = 265,  
 2459 ["Cdot"] = 266,  
 2460 ["cdot"] = 267,  
 2461 ["Ccaron"] = 268,  
 2462 ["ccaron"] = 269,  
 2463 ["Dcaron"] = 270,  
 2464 ["dcaron"] = 271,  
 2465 ["Dstrok"] = 272,

2466 ["dstrok"] = 273,  
 2467 ["Emacr"] = 274,  
 2468 ["emacr"] = 275,  
 2469 ["Edot"] = 278,  
 2470 ["edot"] = 279,  
 2471 ["Eogon"] = 280,  
 2472 ["eogon"] = 281,  
 2473 ["Ecaron"] = 282,  
 2474 ["ecaron"] = 283,  
 2475 ["Gcirc"] = 284,  
 2476 ["gcirc"] = 285,  
 2477 ["Gbreve"] = 286,  
 2478 ["gbreve"] = 287,  
 2479 ["Gdot"] = 288,  
 2480 ["gdot"] = 289,  
 2481 ["Gcedil"] = 290,  
 2482 ["Hcirc"] = 292,  
 2483 ["hcirc"] = 293,  
 2484 ["Hstrok"] = 294,  
 2485 ["hstrok"] = 295,  
 2486 ["Itilde"] = 296,  
 2487 ["itilde"] = 297,  
 2488 ["Imacr"] = 298,  
 2489 ["imacr"] = 299,  
 2490 ["Iogon"] = 302,  
 2491 ["iogon"] = 303,  
 2492 ["Idot"] = 304,  
 2493 ["imath"] = 305,  
 2494 ["inodot"] = 305,  
 2495 ["IJlig"] = 306,  
 2496 ["ijlig"] = 307,  
 2497 ["Jcirc"] = 308,  
 2498 ["jcirc"] = 309,  
 2499 ["Kcedil"] = 310,  
 2500 ["kcedil"] = 311,  
 2501 ["kgreen"] = 312,  
 2502 ["Lacute"] = 313,  
 2503 ["lacute"] = 314,  
 2504 ["Lcedil"] = 315,  
 2505 ["lcedil"] = 316,  
 2506 ["Lcaron"] = 317,  
 2507 ["lcaron"] = 318,  
 2508 ["Lmidot"] = 319,  
 2509 ["lmidot"] = 320,  
 2510 ["Lstrok"] = 321,  
 2511 ["lstrok"] = 322,  
 2512 ["Nacute"] = 323,

2513 ["nacute"] = 324,  
 2514 ["Ncedil"] = 325,  
 2515 ["ncedil"] = 326,  
 2516 ["Ncaron"] = 327,  
 2517 ["ncaron"] = 328,  
 2518 ["napos"] = 329,  
 2519 ["ENG"] = 330,  
 2520 ["eng"] = 331,  
 2521 ["Omacr"] = 332,  
 2522 ["omacr"] = 333,  
 2523 ["Odblac"] = 336,  
 2524 ["odblac"] = 337,  
 2525 ["OElig"] = 338,  
 2526 ["oelig"] = 339,  
 2527 ["Racute"] = 340,  
 2528 ["racute"] = 341,  
 2529 ["Rcedil"] = 342,  
 2530 ["rcedil"] = 343,  
 2531 ["Rcaron"] = 344,  
 2532 ["rcaron"] = 345,  
 2533 ["Sacute"] = 346,  
 2534 ["sacute"] = 347,  
 2535 ["Scirc"] = 348,  
 2536 ["scirc"] = 349,  
 2537 ["Scedil"] = 350,  
 2538 ["scedil"] = 351,  
 2539 ["Scaron"] = 352,  
 2540 ["scaron"] = 353,  
 2541 ["Tcedil"] = 354,  
 2542 ["tcedil"] = 355,  
 2543 ["Tcaron"] = 356,  
 2544 ["tcaron"] = 357,  
 2545 ["Tstrok"] = 358,  
 2546 ["tstrok"] = 359,  
 2547 ["Utilde"] = 360,  
 2548 ["utilde"] = 361,  
 2549 ["Umacr"] = 362,  
 2550 ["umacr"] = 363,  
 2551 ["Ubreve"] = 364,  
 2552 ["ubreve"] = 365,  
 2553 ["Uring"] = 366,  
 2554 ["uring"] = 367,  
 2555 ["Udblac"] = 368,  
 2556 ["udblac"] = 369,  
 2557 ["Uogon"] = 370,  
 2558 ["uogon"] = 371,  
 2559 ["Wcirc"] = 372,



```

2560 ["wcirc"] = 373,
2561 ["Ycirc"] = 374,
2562 ["ycirc"] = 375,
2563 ["Yuml"] = 376,
2564 ["Zacute"] = 377,
2565 ["zacute"] = 378,
2566 ["Zdot"] = 379,
2567 ["zdot"] = 380,
2568 ["Zcaron"] = 381,
2569 ["zcaron"] = 382,
2570 ["fnof"] = 402,
2571 ["imped"] = 437,
2572 ["gacute"] = 501,
2573 ["jmath"] = 567,
2574 ["circ"] = 710,
2575 ["caron"] = 711,
2576 ["Hacek"] = 711,
2577 ["breve"] = 728,
2578 ["Breve"] = 728,
2579 ["dot"] = 729,
2580 ["DiacriticalDot"] = 729,
2581 ["ring"] = 730,
2582 ["ogon"] = 731,
2583 ["tilde"] = 732,
2584 ["DiacriticalTilde"] = 732,
2585 ["dblac"] = 733,
2586 ["DiacriticalDoubleAcute"] = 733,
2587 ["DownBreve"] = 785,
2588 ["UnderBar"] = 818,
2589 ["Alpha"] = 913,
2590 ["Beta"] = 914,
2591 ["Gamma"] = 915,
2592 ["Delta"] = 916,
2593 ["Epsilon"] = 917,
2594 ["Zeta"] = 918,
2595 ["Eta"] = 919,
2596 ["Theta"] = 920,
2597 ["Iota"] = 921,
2598 ["Kappa"] = 922,
2599 ["Lambda"] = 923,
2600 ["Mu"] = 924,
2601 ["Nu"] = 925,
2602 ["Xi"] = 926,
2603 ["Omicron"] = 927,
2604 ["Pi"] = 928,
2605 ["Rho"] = 929,
2606 ["Sigma"] = 931,

```

```

2607 ["Tau"] = 932,
2608 ["Upsilon"] = 933,
2609 ["Phi"] = 934,
2610 ["Chi"] = 935,
2611 ["Psi"] = 936,
2612 ["Omega"] = 937,
2613 ["alpha"] = 945,
2614 ["beta"] = 946,
2615 ["gamma"] = 947,
2616 ["delta"] = 948,
2617 ["epsiv"] = 949,
2618 ["varepsilon"] = 949,
2619 ["epsilon"] = 949,
2620 ["zeta"] = 950,
2621 ["eta"] = 951,
2622 ["theta"] = 952,
2623 ["iota"] = 953,
2624 ["kappa"] = 954,
2625 ["lambda"] = 955,
2626 ["mu"] = 956,
2627 ["nu"] = 957,
2628 ["xi"] = 958,
2629 ["omicron"] = 959,
2630 ["pi"] = 960,
2631 ["rho"] = 961,
2632 ["sigmav"] = 962,
2633 ["varsigma"] = 962,
2634 ["sigmaf"] = 962,
2635 ["sigma"] = 963,
2636 ["tau"] = 964,
2637 ["upsi"] = 965,
2638 ["upsilon"] = 965,
2639 ["phi"] = 966,
2640 ["phiv"] = 966,
2641 ["varphi"] = 966,
2642 ["chi"] = 967,
2643 ["psi"] = 968,
2644 ["omega"] = 969,
2645 ["thetav"] = 977,
2646 ["vartheta"] = 977,
2647 ["thetasym"] = 977,
2648 ["Upsi"] = 978,
2649 ["upsih"] = 978,
2650 ["straightphi"] = 981,
2651 ["piv"] = 982,
2652 ["varpi"] = 982,
2653 ["Gammad"] = 988,

```

```

2654 ["gammad"] = 989,
2655 ["digamma"] = 989,
2656 ["kappav"] = 1008,
2657 ["varkappa"] = 1008,
2658 ["rhov"] = 1009,
2659 ["varrho"] = 1009,
2660 ["epsi"] = 1013,
2661 ["straightepsilon"] = 1013,
2662 ["bepsi"] = 1014,
2663 ["backepsilon"] = 1014,
2664 ["IOcy"] = 1025,
2665 ["DJcy"] = 1026,
2666 ["GJcy"] = 1027,
2667 ["Jukcy"] = 1028,
2668 ["DScy"] = 1029,
2669 ["Iukcy"] = 1030,
2670 ["YIcy"] = 1031,
2671 ["Jsercy"] = 1032,
2672 ["LJcy"] = 1033,
2673 ["NJcy"] = 1034,
2674 ["TSHcy"] = 1035,
2675 ["KJcy"] = 1036,
2676 ["Ubrcy"] = 1038,
2677 ["DZcy"] = 1039,
2678 ["Acy"] = 1040,
2679 ["Bcy"] = 1041,
2680 ["Vcy"] = 1042,
2681 ["Gcy"] = 1043,
2682 ["Dcy"] = 1044,
2683 ["IEcy"] = 1045,
2684 ["ZHcy"] = 1046,
2685 ["Zcy"] = 1047,
2686 ["Icy"] = 1048,
2687 ["Jcy"] = 1049,
2688 ["Kcy"] = 1050,
2689 ["Lcy"] = 1051,
2690 ["Mcy"] = 1052,
2691 ["Ncy"] = 1053,
2692 ["Ocy"] = 1054,
2693 ["Pcy"] = 1055,
2694 ["Rcy"] = 1056,
2695 ["Scy"] = 1057,
2696 ["Tcy"] = 1058,
2697 ["Ucy"] = 1059,
2698 ["Fcy"] = 1060,
2699 ["KHcy"] = 1061,
2700 ["TScy"] = 1062,

```

```

2701 ["CHcy"] = 1063,
2702 ["SHcy"] = 1064,
2703 ["SHCHcy"] = 1065,
2704 ["HARDcy"] = 1066,
2705 ["Ycy"] = 1067,
2706 ["SOFTcy"] = 1068,
2707 ["Ecy"] = 1069,
2708 ["YUcy"] = 1070,
2709 ["YAcy"] = 1071,
2710 ["acy"] = 1072,
2711 ["bcy"] = 1073,
2712 ["vcy"] = 1074,
2713 ["gcy"] = 1075,
2714 ["dcy"] = 1076,
2715 ["iecy"] = 1077,
2716 ["zhcy"] = 1078,
2717 ["zcy"] = 1079,
2718 ["icy"] = 1080,
2719 ["jcy"] = 1081,
2720 ["kcy"] = 1082,
2721 ["lcy"] = 1083,
2722 ["mcy"] = 1084,
2723 ["ncy"] = 1085,
2724 ["ocy"] = 1086,
2725 ["pcy"] = 1087,
2726 ["rcy"] = 1088,
2727 ["scy"] = 1089,
2728 ["tcy"] = 1090,
2729 ["ucy"] = 1091,
2730 ["fcy"] = 1092,
2731 ["khcy"] = 1093,
2732 ["tscy"] = 1094,
2733 ["chcy"] = 1095,
2734 ["shcy"] = 1096,
2735 ["shchcy"] = 1097,
2736 ["hardcy"] = 1098,
2737 ["ycy"] = 1099,
2738 ["softcy"] = 1100,
2739 ["ecy"] = 1101,
2740 ["yucy"] = 1102,
2741 ["yacy"] = 1103,
2742 ["iocy"] = 1105,
2743 ["djcy"] = 1106,
2744 ["gjcy"] = 1107,
2745 ["jukcy"] = 1108,
2746 ["dscy"] = 1109,
2747 ["iukcy"] = 1110,

```

```

2748 ["yicy"] = 1111,
2749 ["jsercy"] = 1112,
2750 ["ljcy"] = 1113,
2751 ["njcy"] = 1114,
2752 ["tshcy"] = 1115,
2753 ["kjcy"] = 1116,
2754 ["ubrky"] = 1118,
2755 ["dzcy"] = 1119,
2756 ["ensp"] = 8194,
2757 ["emsp"] = 8195,
2758 ["emsp13"] = 8196,
2759 ["emsp14"] = 8197,
2760 ["numsp"] = 8199,
2761 ["puncsp"] = 8200,
2762 ["thinsp"] = 8201,
2763 ["ThinSpace"] = 8201,
2764 ["hairsp"] = 8202,
2765 ["VeryThinSpace"] = 8202,
2766 ["ZeroWidthSpace"] = 8203,
2767 ["NegativeVeryThinSpace"] = 8203,
2768 ["NegativeThinSpace"] = 8203,
2769 ["NegativeMediumSpace"] = 8203,
2770 ["NegativeThickSpace"] = 8203,
2771 ["zwnj"] = 8204,
2772 ["zwj"] = 8205,
2773 ["lrm"] = 8206,
2774 ["rlm"] = 8207,
2775 ["hyphen"] = 8208,
2776 ["dash"] = 8208,
2777 ["ndash"] = 8211,
2778 ["mdash"] = 8212,
2779 ["horbar"] = 8213,
2780 ["Verbar"] = 8214,
2781 ["Vert"] = 8214,
2782 ["lsquo"] = 8216,
2783 ["OpenCurlyQuote"] = 8216,
2784 ["rsquo"] = 8217,
2785 ["rsquor"] = 8217,
2786 ["CloseCurlyQuote"] = 8217,
2787 ["lsquor"] = 8218,
2788 ["sbquo"] = 8218,
2789 ["ldquo"] = 8220,
2790 ["OpenCurlyDoubleQuote"] = 8220,
2791 ["rdquo"] = 8221,
2792 ["rdquor"] = 8221,
2793 ["CloseCurlyDoubleQuote"] = 8221,
2794 ["ldquor"] = 8222,

```

2795 ["bdquo"] = 8222,  
 2796 ["dagger"] = 8224,  
 2797 ["Dagger"] = 8225,  
 2798 ["ddagger"] = 8225,  
 2799 ["bull"] = 8226,  
 2800 ["bullet"] = 8226,  
 2801 ["nldr"] = 8229,  
 2802 ["hellip"] = 8230,  
 2803 ["mldr"] = 8230,  
 2804 ["permil"] = 8240,  
 2805 ["pertenk"] = 8241,  
 2806 ["prime"] = 8242,  
 2807 ["Prime"] = 8243,  
 2808 ["tprime"] = 8244,  
 2809 ["bprime"] = 8245,  
 2810 ["backprime"] = 8245,  
 2811 ["lsaquo"] = 8249,  
 2812 ["rsaquo"] = 8250,  
 2813 ["oline"] = 8254,  
 2814 ["caret"] = 8257,  
 2815 ["hybull"] = 8259,  
 2816 ["frasl"] = 8260,  
 2817 ["bsemi"] = 8271,  
 2818 ["qprime"] = 8279,  
 2819 ["MediumSpace"] = 8287,  
 2820 ["NoBreak"] = 8288,  
 2821 ["ApplyFunction"] = 8289,  
 2822 ["af"] = 8289,  
 2823 ["InvisibleTimes"] = 8290,  
 2824 ["it"] = 8290,  
 2825 ["InvisibleComma"] = 8291,  
 2826 ["ic"] = 8291,  
 2827 ["euro"] = 8364,  
 2828 ["tdot"] = 8411,  
 2829 ["TripleDot"] = 8411,  
 2830 ["DotDot"] = 8412,  
 2831 ["Copf"] = 8450,  
 2832 ["complexes"] = 8450,  
 2833 ["incare"] = 8453,  
 2834 ["gscr"] = 8458,  
 2835 ["hamilt"] = 8459,  
 2836 ["HilbertSpace"] = 8459,  
 2837 ["Hscr"] = 8459,  
 2838 ["Hfr"] = 8460,  
 2839 ["Poincareplane"] = 8460,  
 2840 ["quaternions"] = 8461,  
 2841 ["Hopf"] = 8461,

```

2842 ["planckh"] = 8462,
2843 ["planck"] = 8463,
2844 ["hbar"] = 8463,
2845 ["plankv"] = 8463,
2846 ["hslash"] = 8463,
2847 ["Iscr"] = 8464,
2848 ["imagline"] = 8464,
2849 ["image"] = 8465,
2850 ["Im"] = 8465,
2851 ["imagpart"] = 8465,
2852 ["Ifr"] = 8465,
2853 ["Lscr"] = 8466,
2854 ["lagran"] = 8466,
2855 ["Laplacetrif"] = 8466,
2856 ["ell"] = 8467,
2857 ["Nopf"] = 8469,
2858 ["naturals"] = 8469,
2859 ["numero"] = 8470,
2860 ["copysr"] = 8471,
2861 ["weierp"] = 8472,
2862 ["wp"] = 8472,
2863 ["Popf"] = 8473,
2864 ["primes"] = 8473,
2865 ["rationals"] = 8474,
2866 ["Qopf"] = 8474,
2867 ["Rscr"] = 8475,
2868 ["realine"] = 8475,
2869 ["real"] = 8476,
2870 ["Re"] = 8476,
2871 ["realpart"] = 8476,
2872 ["Rfr"] = 8476,
2873 ["reals"] = 8477,
2874 ["Ropf"] = 8477,
2875 ["rx"] = 8478,
2876 ["trade"] = 8482,
2877 ["TRADE"] = 8482,
2878 ["integers"] = 8484,
2879 ["Zopf"] = 8484,
2880 ["ohm"] = 8486,
2881 ["mho"] = 8487,
2882 ["Zfr"] = 8488,
2883 ["zeetrf"] = 8488,
2884 ["iiota"] = 8489,
2885 ["angst"] = 8491,
2886 ["bernou"] = 8492,
2887 ["Bernoullis"] = 8492,
2888 ["Bscr"] = 8492,

```

```

2889 ["Cfr"] = 8493,
2890 ["Cayleys"] = 8493,
2891 ["escr"] = 8495,
2892 ["Escr"] = 8496,
2893 ["expectation"] = 8496,
2894 ["Fscr"] = 8497,
2895 ["Fouriertrf"] = 8497,
2896 ["phmmat"] = 8499,
2897 ["Mellintrf"] = 8499,
2898 ["Mscr"] = 8499,
2899 ["order"] = 8500,
2900 ["orderof"] = 8500,
2901 ["oscr"] = 8500,
2902 ["alefsym"] = 8501,
2903 ["aleph"] = 8501,
2904 ["beth"] = 8502,
2905 ["gimel"] = 8503,
2906 ["daleth"] = 8504,
2907 ["CapitalDifferentialD"] = 8517,
2908 ["DD"] = 8517,
2909 ["DifferentialD"] = 8518,
2910 ["dd"] = 8518,
2911 ["ExponentialE"] = 8519,
2912 ["exponentiale"] = 8519,
2913 ["ee"] = 8519,
2914 ["ImaginaryI"] = 8520,
2915 ["ii"] = 8520,
2916 ["frac13"] = 8531,
2917 ["frac23"] = 8532,
2918 ["frac15"] = 8533,
2919 ["frac25"] = 8534,
2920 ["frac35"] = 8535,
2921 ["frac45"] = 8536,
2922 ["frac16"] = 8537,
2923 ["frac56"] = 8538,
2924 ["frac18"] = 8539,
2925 ["frac38"] = 8540,
2926 ["frac58"] = 8541,
2927 ["frac78"] = 8542,
2928 ["larr"] = 8592,
2929 ["leftarrow"] = 8592,
2930 ["LeftArrow"] = 8592,
2931 ["slarr"] = 8592,
2932 ["ShortLeftArrow"] = 8592,
2933 ["uarr"] = 8593,
2934 ["uparrow"] = 8593,
2935 ["UpArrow"] = 8593,

```



```

2936 ["ShortUpArrow"] = 8593,
2937 ["rarr"] = 8594,
2938 ["rightarrow"] = 8594,
2939 ["RightArrow"] = 8594,
2940 ["srarr"] = 8594,
2941 ["ShortRightArrow"] = 8594,
2942 ["darr"] = 8595,
2943 ["downarrow"] = 8595,
2944 ["DownArrow"] = 8595,
2945 ["ShortDownArrow"] = 8595,
2946 ["harr"] = 8596,
2947 ["leftrightarrow"] = 8596,
2948 ["LeftRightArrow"] = 8596,
2949 ["varr"] = 8597,
2950 ["updownarrow"] = 8597,
2951 ["UpDownArrow"] = 8597,
2952 ["nwarr"] = 8598,
2953 ["UpperLeftArrow"] = 8598,
2954 ["nwarrow"] = 8598,
2955 ["nearr"] = 8599,
2956 ["UpperRightArrow"] = 8599,
2957 ["nearrow"] = 8599,
2958 ["searr"] = 8600,
2959 ["searrow"] = 8600,
2960 ["LowerRightArrow"] = 8600,
2961 ["swarr"] = 8601,
2962 ["swarrow"] = 8601,
2963 ["LowerLeftArrow"] = 8601,
2964 ["nlarr"] = 8602,
2965 ["nleftarrow"] = 8602,
2966 ["nrarr"] = 8603,
2967 ["nrightarrow"] = 8603,
2968 ["rarrw"] = 8605,
2969 ["rightsquigarrow"] = 8605,
2970 ["Larr"] = 8606,
2971 ["twoheadleftarrow"] = 8606,
2972 ["Uarr"] = 8607,
2973 ["Rarr"] = 8608,
2974 ["twoheadrightarrow"] = 8608,
2975 ["Darr"] = 8609,
2976 ["larrrtl"] = 8610,
2977 ["leftarrowtail"] = 8610,
2978 ["rarrtl"] = 8611,
2979 ["rightarrowtail"] = 8611,
2980 ["LeftTeeArrow"] = 8612,
2981 ["mapstoleft"] = 8612,
2982 ["UpTeeArrow"] = 8613,

```

```

2983 ["mapstoup"] = 8613,
2984 ["map"] = 8614,
2985 ["RightTeeArrow"] = 8614,
2986 ["mapsto"] = 8614,
2987 ["DownTeeArrow"] = 8615,
2988 ["mapstodown"] = 8615,
2989 ["larrhk"] = 8617,
2990 ["hookleftarrow"] = 8617,
2991 ["rarrhk"] = 8618,
2992 ["hookrightarrow"] = 8618,
2993 ["larrlp"] = 8619,
2994 ["looparrowleft"] = 8619,
2995 ["rarrlp"] = 8620,
2996 ["looparrowright"] = 8620,
2997 ["harrrw"] = 8621,
2998 ["leftrightsquigarrow"] = 8621,
2999 ["nharr"] = 8622,
3000 ["nleftrightarrow"] = 8622,
3001 ["lsh"] = 8624,
3002 ["Lsh"] = 8624,
3003 ["rsh"] = 8625,
3004 ["Rsh"] = 8625,
3005 ["ldsh"] = 8626,
3006 ["rdsh"] = 8627,
3007 ["crarr"] = 8629,
3008 ["cularr"] = 8630,
3009 ["curvearrowleft"] = 8630,
3010 ["curarr"] = 8631,
3011 ["curvearrowright"] = 8631,
3012 ["olarr"] = 8634,
3013 ["circlearrowleft"] = 8634,
3014 ["orarr"] = 8635,
3015 ["circlearrowright"] = 8635,
3016 ["lharu"] = 8636,
3017 ["LeftVector"] = 8636,
3018 ["leftharpoonup"] = 8636,
3019 ["lhard"] = 8637,
3020 ["leftharpoondown"] = 8637,
3021 ["DownLeftVector"] = 8637,
3022 ["uharr"] = 8638,
3023 ["upharpoonright"] = 8638,
3024 ["RightUpVector"] = 8638,
3025 ["uharl"] = 8639,
3026 ["upharpoonleft"] = 8639,
3027 ["LeftUpVector"] = 8639,
3028 ["rharu"] = 8640,
3029 ["RightVector"] = 8640,

```

```

3030 ["rightharpoonup"] = 8640,
3031 ["rhard"] = 8641,
3032 ["rightharpoondown"] = 8641,
3033 ["DownRightVector"] = 8641,
3034 ["dharr"] = 8642,
3035 ["RightDownVector"] = 8642,
3036 ["downharpoonright"] = 8642,
3037 ["dharl"] = 8643,
3038 ["LeftDownVector"] = 8643,
3039 ["downharpoonleft"] = 8643,
3040 ["rlarr"] = 8644,
3041 ["rightleftarrows"] = 8644,
3042 ["RightArrowLeftArrow"] = 8644,
3043 ["udarr"] = 8645,
3044 ["UpArrowDownArrow"] = 8645,
3045 ["lrarr"] = 8646,
3046 ["leftrightarrows"] = 8646,
3047 ["LeftArrowRightArrow"] = 8646,
3048 ["llarr"] = 8647,
3049 ["leftleftarrows"] = 8647,
3050 ["uuarr"] = 8648,
3051 ["upuparrows"] = 8648,
3052 ["rrarr"] = 8649,
3053 ["rightrightarrows"] = 8649,
3054 ["ddarr"] = 8650,
3055 ["downdownarrows"] = 8650,
3056 ["lrhar"] = 8651,
3057 ["ReverseEquilibrium"] = 8651,
3058 ["leftrightharpoons"] = 8651,
3059 ["rlhar"] = 8652,
3060 ["rightleftharpoons"] = 8652,
3061 ["Equilibrium"] = 8652,
3062 ["nlArr"] = 8653,
3063 ["nLeftarrow"] = 8653,
3064 ["nhArr"] = 8654,
3065 ["nLeftrightarrow"] = 8654,
3066 ["nrArr"] = 8655,
3067 ["nrightarrow"] = 8655,
3068 ["lArr"] = 8656,
3069 ["Leftarrow"] = 8656,
3070 ["DoubleLeftArrow"] = 8656,
3071 ["uArr"] = 8657,
3072 ["Uparrow"] = 8657,
3073 ["DoubleUpArrow"] = 8657,
3074 ["rArr"] = 8658,
3075 ["Rightarrow"] = 8658,
3076 ["Implies"] = 8658,

```

```

3077 ["DoubleRightArrow"] = 8658,
3078 ["dArr"] = 8659,
3079 ["Downarrow"] = 8659,
3080 ["DoubleDownArrow"] = 8659,
3081 ["hArr"] = 8660,
3082 ["Leftrightarrow"] = 8660,
3083 ["DoubleLeftRightArrow"] = 8660,
3084 ["iff"] = 8660,
3085 ["vArr"] = 8661,
3086 ["Updownarrow"] = 8661,
3087 ["DoubleUpDownArrow"] = 8661,
3088 ["nwArr"] = 8662,
3089 ["neArr"] = 8663,
3090 ["seArr"] = 8664,
3091 ["swArr"] = 8665,
3092 ["lAarr"] = 8666,
3093 ["Lleftarrow"] = 8666,
3094 ["rAarr"] = 8667,
3095 ["Rrightarrow"] = 8667,
3096 ["zigrarr"] = 8669,
3097 ["larrb"] = 8676,
3098 ["LeftArrowBar"] = 8676,
3099 ["rarrb"] = 8677,
3100 ["RightArrowBar"] = 8677,
3101 ["duarr"] = 8693,
3102 ["DownArrowUpArrow"] = 8693,
3103 ["loarr"] = 8701,
3104 ["roarr"] = 8702,
3105 ["hoarr"] = 8703,
3106 ["forall"] = 8704,
3107 ["ForAll"] = 8704,
3108 ["comp"] = 8705,
3109 ["complement"] = 8705,
3110 ["part"] = 8706,
3111 ["PartialD"] = 8706,
3112 ["exist"] = 8707,
3113 ["Exists"] = 8707,
3114 ["nexist"] = 8708,
3115 ["NotExists"] = 8708,
3116 ["nexists"] = 8708,
3117 ["empty"] = 8709,
3118 ["emptyset"] = 8709,
3119 ["emptyv"] = 8709,
3120 ["varnothing"] = 8709,
3121 ["nabla"] = 8711,
3122 ["Del"] = 8711,
3123 ["isin"] = 8712,

```

```

3124 ["isinv"] = 8712,
3125 ["Element"] = 8712,
3126 ["in"] = 8712,
3127 ["notin"] = 8713,
3128 ["NotElement"] = 8713,
3129 ["notinva"] = 8713,
3130 ["niv"] = 8715,
3131 ["ReverseElement"] = 8715,
3132 ["ni"] = 8715,
3133 ["SuchThat"] = 8715,
3134 ["notni"] = 8716,
3135 ["notniva"] = 8716,
3136 ["NotReverseElement"] = 8716,
3137 ["prod"] = 8719,
3138 ["Product"] = 8719,
3139 ["coprod"] = 8720,
3140 ["Coproduct"] = 8720,
3141 ["sum"] = 8721,
3142 ["Sum"] = 8721,
3143 ["minus"] = 8722,
3144 ["mnplus"] = 8723,
3145 ["mp"] = 8723,
3146 ["MinusPlus"] = 8723,
3147 ["plusdo"] = 8724,
3148 ["dotplus"] = 8724,
3149 ["setmn"] = 8726,
3150 ["setminus"] = 8726,
3151 ["Backslash"] = 8726,
3152 ["ssetmn"] = 8726,
3153 ["smallsetminus"] = 8726,
3154 ["lowast"] = 8727,
3155 ["compfn"] = 8728,
3156 ["SmallCircle"] = 8728,
3157 ["radic"] = 8730,
3158 ["Sqrt"] = 8730,
3159 ["prop"] = 8733,
3160 ["propto"] = 8733,
3161 ["Proportional"] = 8733,
3162 ["vprop"] = 8733,
3163 ["varpropto"] = 8733,
3164 ["infin"] = 8734,
3165 ["angrt"] = 8735,
3166 ["ang"] = 8736,
3167 ["angle"] = 8736,
3168 ["angmsd"] = 8737,
3169 ["measuredangle"] = 8737,
3170 ["angsph"] = 8738,

```

```

3171 ["mid"] = 8739,
3172 ["VerticalBar"] = 8739,
3173 ["smid"] = 8739,
3174 ["shortmid"] = 8739,
3175 ["nmid"] = 8740,
3176 ["NotVerticalBar"] = 8740,
3177 ["nsmid"] = 8740,
3178 ["nshortmid"] = 8740,
3179 ["par"] = 8741,
3180 ["parallel"] = 8741,
3181 ["DoubleVerticalBar"] = 8741,
3182 ["spar"] = 8741,
3183 ["shortparallel"] = 8741,
3184 ["npar"] = 8742,
3185 ["nparallel"] = 8742,
3186 ["NotDoubleVerticalBar"] = 8742,
3187 ["nspar"] = 8742,
3188 ["nshortparallel"] = 8742,
3189 ["and"] = 8743,
3190 ["wedge"] = 8743,
3191 ["or"] = 8744,
3192 ["vee"] = 8744,
3193 ["cap"] = 8745,
3194 ["cup"] = 8746,
3195 ["int"] = 8747,
3196 ["Integral"] = 8747,
3197 ["Int"] = 8748,
3198 ["tint"] = 8749,
3199 ["iiint"] = 8749,
3200 ["conint"] = 8750,
3201 ["oint"] = 8750,
3202 ["ContourIntegral"] = 8750,
3203 ["Conint"] = 8751,
3204 ["DoubleContourIntegral"] = 8751,
3205 ["Cconint"] = 8752,
3206 ["cwint"] = 8753,
3207 ["cwconint"] = 8754,
3208 ["ClockwiseContourIntegral"] = 8754,
3209 ["awconint"] = 8755,
3210 ["CounterClockwiseContourIntegral"] = 8755,
3211 ["there4"] = 8756,
3212 ["therefore"] = 8756,
3213 ["Therefore"] = 8756,
3214 ["because"] = 8757,
3215 ["because"] = 8757,
3216 ["Because"] = 8757,
3217 ["ratio"] = 8758,

```

```

3218 ["Colon"] = 8759,
3219 ["Proportion"] = 8759,
3220 ["minusd"] = 8760,
3221 ["dotminus"] = 8760,
3222 ["mDDot"] = 8762,
3223 ["homtht"] = 8763,
3224 ["sim"] = 8764,
3225 ["Tilde"] = 8764,
3226 ["thksim"] = 8764,
3227 ["thicksim"] = 8764,
3228 ["bsim"] = 8765,
3229 ["backsim"] = 8765,
3230 ["ac"] = 8766,
3231 ["mstpos"] = 8766,
3232 ["acd"] = 8767,
3233 ["wreath"] = 8768,
3234 ["VerticalTilde"] = 8768,
3235 ["wr"] = 8768,
3236 ["nsim"] = 8769,
3237 ["NotTilde"] = 8769,
3238 ["esim"] = 8770,
3239 ["EqualTilde"] = 8770,
3240 ["eqsim"] = 8770,
3241 ["sime"] = 8771,
3242 ["TildeEqual"] = 8771,
3243 ["simeq"] = 8771,
3244 ["nsime"] = 8772,
3245 ["nsimeq"] = 8772,
3246 ["NotTildeEqual"] = 8772,
3247 ["cong"] = 8773,
3248 ["TildeFullEqual"] = 8773,
3249 ["simne"] = 8774,
3250 ["ncong"] = 8775,
3251 ["NotTildeFullEqual"] = 8775,
3252 ["asymp"] = 8776,
3253 ["ap"] = 8776,
3254 ["TildeTilde"] = 8776,
3255 ["approx"] = 8776,
3256 ["thkap"] = 8776,
3257 ["thickapprox"] = 8776,
3258 ["nap"] = 8777,
3259 ["NotTildeTilde"] = 8777,
3260 ["napprox"] = 8777,
3261 ["ape"] = 8778,
3262 ["approxeq"] = 8778,
3263 ["apid"] = 8779,
3264 ["bcong"] = 8780,

```

```

3265 ["backcong"] = 8780,
3266 ["asympeq"] = 8781,
3267 ["CupCap"] = 8781,
3268 ["bump"] = 8782,
3269 ["HumpDownHump"] = 8782,
3270 ["Bumpeq"] = 8782,
3271 ["bumpe"] = 8783,
3272 ["HumpEqual"] = 8783,
3273 ["bumpeq"] = 8783,
3274 ["esdot"] = 8784,
3275 ["DotEqual"] = 8784,
3276 ["doteq"] = 8784,
3277 ["eDot"] = 8785,
3278 ["doteqdot"] = 8785,
3279 ["efDot"] = 8786,
3280 ["fallingdotseq"] = 8786,
3281 ["erDot"] = 8787,
3282 ["risingdotseq"] = 8787,
3283 ["colone"] = 8788,
3284 ["coloneq"] = 8788,
3285 ["Assign"] = 8788,
3286 ["ecolon"] = 8789,
3287 ["eqcolon"] = 8789,
3288 ["ecir"] = 8790,
3289 ["eqcirc"] = 8790,
3290 ["cire"] = 8791,
3291 ["circeq"] = 8791,
3292 ["wedgeq"] = 8793,
3293 ["veeeq"] = 8794,
3294 ["trie"] = 8796,
3295 ["triangleq"] = 8796,
3296 ["equest"] = 8799,
3297 ["questeq"] = 8799,
3298 ["ne"] = 8800,
3299 ["NotEqual"] = 8800,
3300 ["equiv"] = 8801,
3301 ["Congruent"] = 8801,
3302 ["nequiv"] = 8802,
3303 ["NotCongruent"] = 8802,
3304 ["le"] = 8804,
3305 ["leq"] = 8804,
3306 ["ge"] = 8805,
3307 ["GreaterEqual"] = 8805,
3308 ["geq"] = 8805,
3309 ["lE"] = 8806,
3310 ["LessFullEqual"] = 8806,
3311 ["leqq"] = 8806,

```



```

3312 ["gE"] = 8807,
3313 ["GreaterFullEqual"] = 8807,
3314 ["geqq"] = 8807,
3315 ["lnE"] = 8808,
3316 ["lneqq"] = 8808,
3317 ["gnE"] = 8809,
3318 ["gneqq"] = 8809,
3319 ["Lt"] = 8810,
3320 ["NestedLessLess"] = 8810,
3321 ["ll"] = 8810,
3322 ["Gt"] = 8811,
3323 ["NestedGreaterGreater"] = 8811,
3324 ["gg"] = 8811,
3325 ["twixt"] = 8812,
3326 ["between"] = 8812,
3327 ["NotCupCap"] = 8813,
3328 ["nlt"] = 8814,
3329 ["NotLess"] = 8814,
3330 ["nless"] = 8814,
3331 ["ngt"] = 8815,
3332 ["NotGreater"] = 8815,
3333 ["ngtr"] = 8815,
3334 ["nle"] = 8816,
3335 ["NotLessEqual"] = 8816,
3336 ["nleq"] = 8816,
3337 ["nge"] = 8817,
3338 ["NotGreaterEqual"] = 8817,
3339 ["ngeq"] = 8817,
3340 ["lsim"] = 8818,
3341 ["LessTilde"] = 8818,
3342 ["lesssim"] = 8818,
3343 ["gsim"] = 8819,
3344 ["gtrsim"] = 8819,
3345 ["GreaterTilde"] = 8819,
3346 ["nlsim"] = 8820,
3347 ["NotLessTilde"] = 8820,
3348 ["ngsim"] = 8821,
3349 ["NotGreaterTilde"] = 8821,
3350 ["lg"] = 8822,
3351 ["lessgtr"] = 8822,
3352 ["LessGreater"] = 8822,
3353 ["gl"] = 8823,
3354 ["gtrless"] = 8823,
3355 ["GreaterLess"] = 8823,
3356 ["ntlg"] = 8824,
3357 ["NotLessGreater"] = 8824,
3358 ["ntgl"] = 8825,

```

```

3359 ["NotGreaterLess"] = 8825,
3360 ["pr"] = 8826,
3361 ["Precedes"] = 8826,
3362 ["prec"] = 8826,
3363 ["sc"] = 8827,
3364 ["Succeeds"] = 8827,
3365 ["succ"] = 8827,
3366 ["prcue"] = 8828,
3367 ["PrecedesSlantEqual"] = 8828,
3368 ["preccurlyeq"] = 8828,
3369 ["sccue"] = 8829,
3370 ["SucceedsSlantEqual"] = 8829,
3371 ["succcurlyeq"] = 8829,
3372 ["prsim"] = 8830,
3373 ["precsim"] = 8830,
3374 ["PrecedesTilde"] = 8830,
3375 ["scsim"] = 8831,
3376 ["succsim"] = 8831,
3377 ["SucceedsTilde"] = 8831,
3378 ["npr"] = 8832,
3379 ["nprec"] = 8832,
3380 ["NotPrecedes"] = 8832,
3381 ["nsc"] = 8833,
3382 ["nsucc"] = 8833,
3383 ["NotSucceeds"] = 8833,
3384 ["sub"] = 8834,
3385 ["subset"] = 8834,
3386 ["sup"] = 8835,
3387 ["supset"] = 8835,
3388 ["Superset"] = 8835,
3389 ["nsub"] = 8836,
3390 ["nsup"] = 8837,
3391 ["sube"] = 8838,
3392 ["SubsetEqual"] = 8838,
3393 ["subseteq"] = 8838,
3394 ["supe"] = 8839,
3395 ["supseteq"] = 8839,
3396 ["SupersetEqual"] = 8839,
3397 ["nsube"] = 8840,
3398 ["nsubseteq"] = 8840,
3399 ["NotSubsetEqual"] = 8840,
3400 ["nsupe"] = 8841,
3401 ["nsupseteq"] = 8841,
3402 ["NotSupersetEqual"] = 8841,
3403 ["subne"] = 8842,
3404 ["subsetneq"] = 8842,
3405 ["supne"] = 8843,

```

```

3406 ["supsetneq"] = 8843,
3407 ["cupdot"] = 8845,
3408 ["uplus"] = 8846,
3409 ["UnionPlus"] = 8846,
3410 ["sqsub"] = 8847,
3411 ["SquareSubset"] = 8847,
3412 ["sqsubset"] = 8847,
3413 ["sqsup"] = 8848,
3414 ["SquareSuperset"] = 8848,
3415 ["sqsupset"] = 8848,
3416 ["sqsube"] = 8849,
3417 ["SquareSubsetEqual"] = 8849,
3418 ["sqsubseteq"] = 8849,
3419 ["sqsupe"] = 8850,
3420 ["SquareSupersetEqual"] = 8850,
3421 ["sqsupseteq"] = 8850,
3422 ["sqcap"] = 8851,
3423 ["SquareIntersection"] = 8851,
3424 ["sqcup"] = 8852,
3425 ["SquareUnion"] = 8852,
3426 ["oplus"] = 8853,
3427 ["CirclePlus"] = 8853,
3428 ["ominus"] = 8854,
3429 ["CircleMinus"] = 8854,
3430 ["otimes"] = 8855,
3431 ["CircleTimes"] = 8855,
3432 ["osol"] = 8856,
3433 ["odot"] = 8857,
3434 ["CircleDot"] = 8857,
3435 ["ocir"] = 8858,
3436 ["circledcirc"] = 8858,
3437 ["oast"] = 8859,
3438 ["circledast"] = 8859,
3439 ["odash"] = 8861,
3440 ["circleddash"] = 8861,
3441 ["plusb"] = 8862,
3442 ["boxplus"] = 8862,
3443 ["minusb"] = 8863,
3444 ["boxminus"] = 8863,
3445 ["timesb"] = 8864,
3446 ["boxtimes"] = 8864,
3447 ["sdotb"] = 8865,
3448 ["dotsquare"] = 8865,
3449 ["vdash"] = 8866,
3450 ["RightTee"] = 8866,
3451 ["dashv"] = 8867,
3452 ["LeftTee"] = 8867,

```

```

3453 ["top"] = 8868,
3454 ["DownTee"] = 8868,
3455 ["bottom"] = 8869,
3456 ["bot"] = 8869,
3457 ["perp"] = 8869,
3458 ["UpTee"] = 8869,
3459 ["models"] = 8871,
3460 ["vDash"] = 8872,
3461 ["DoubleRightTee"] = 8872,
3462 ["Vdash"] = 8873,
3463 ["Vvdash"] = 8874,
3464 ["VDash"] = 8875,
3465 ["nvdash"] = 8876,
3466 ["nvDash"] = 8877,
3467 ["nVdash"] = 8878,
3468 ["nVDash"] = 8879,
3469 ["prurel"] = 8880,
3470 ["vltri"] = 8882,
3471 ["vartriangleleft"] = 8882,
3472 ["LeftTriangle"] = 8882,
3473 ["vrtri"] = 8883,
3474 ["vartriangleright"] = 8883,
3475 ["RightTriangle"] = 8883,
3476 ["ltrie"] = 8884,
3477 ["trianglelefteq"] = 8884,
3478 ["LeftTriangleEqual"] = 8884,
3479 ["rtrie"] = 8885,
3480 ["trianglerighteq"] = 8885,
3481 ["RightTriangleEqual"] = 8885,
3482 ["origof"] = 8886,
3483 ["imof"] = 8887,
3484 ["mumap"] = 8888,
3485 ["multimap"] = 8888,
3486 ["hercon"] = 8889,
3487 ["intcal"] = 8890,
3488 ["intercal"] = 8890,
3489 ["veebar"] = 8891,
3490 ["barvee"] = 8893,
3491 ["angrtvb"] = 8894,
3492 ["lrtri"] = 8895,
3493 ["xwedge"] = 8896,
3494 ["Wedge"] = 8896,
3495 ["bigwedge"] = 8896,
3496 ["xvee"] = 8897,
3497 ["Vee"] = 8897,
3498 ["bigvee"] = 8897,
3499 ["xcap"] = 8898,

```

```

3500 ["Intersection"] = 8898,
3501 ["bigcap"] = 8898,
3502 ["xcup"] = 8899,
3503 ["Union"] = 8899,
3504 ["bigcup"] = 8899,
3505 ["diam"] = 8900,
3506 ["diamond"] = 8900,
3507 ["Diamond"] = 8900,
3508 ["sdot"] = 8901,
3509 ["sstarf"] = 8902,
3510 ["Star"] = 8902,
3511 ["divonx"] = 8903,
3512 ["divideontimes"] = 8903,
3513 ["bowtie"] = 8904,
3514 ["ltimes"] = 8905,
3515 ["rtimes"] = 8906,
3516 ["lthree"] = 8907,
3517 ["leftthreetimes"] = 8907,
3518 ["rthree"] = 8908,
3519 ["rightthreetimes"] = 8908,
3520 ["bsime"] = 8909,
3521 ["backsimeq"] = 8909,
3522 ["cuvee"] = 8910,
3523 ["curlyvee"] = 8910,
3524 ["cuwed"] = 8911,
3525 ["curlywedge"] = 8911,
3526 ["Sub"] = 8912,
3527 ["Subset"] = 8912,
3528 ["Sup"] = 8913,
3529 ["Supset"] = 8913,
3530 ["Cap"] = 8914,
3531 ["Cup"] = 8915,
3532 ["fork"] = 8916,
3533 ["pitchfork"] = 8916,
3534 ["epar"] = 8917,
3535 ["ltdot"] = 8918,
3536 ["lessdot"] = 8918,
3537 ["gtdot"] = 8919,
3538 ["gtrdot"] = 8919,
3539 ["Ll"] = 8920,
3540 ["Gg"] = 8921,
3541 ["ggg"] = 8921,
3542 ["leg"] = 8922,
3543 ["LessEqualGreater"] = 8922,
3544 ["lesseqgtr"] = 8922,
3545 ["gel"] = 8923,
3546 ["gtreqless"] = 8923,

```

3547 ["GreaterEqualLess"] = 8923,  
 3548 ["cuepr"] = 8926,  
 3549 ["curlyeqprec"] = 8926,  
 3550 ["cuesc"] = 8927,  
 3551 ["curlyeqsucc"] = 8927,  
 3552 ["nprcue"] = 8928,  
 3553 ["NotPrecedesSlantEqual"] = 8928,  
 3554 ["nsccue"] = 8929,  
 3555 ["NotSucceedsSlantEqual"] = 8929,  
 3556 ["nsqsube"] = 8930,  
 3557 ["NotSquareSubsetEqual"] = 8930,  
 3558 ["nsqsupe"] = 8931,  
 3559 ["NotSquareSupersetEqual"] = 8931,  
 3560 ["lnsim"] = 8934,  
 3561 ["gnsim"] = 8935,  
 3562 ["prnsim"] = 8936,  
 3563 ["precnsim"] = 8936,  
 3564 ["scnsim"] = 8937,  
 3565 ["succnsim"] = 8937,  
 3566 ["nltri"] = 8938,  
 3567 ["ntriangleleft"] = 8938,  
 3568 ["NotLeftTriangle"] = 8938,  
 3569 ["nrtri"] = 8939,  
 3570 ["ntriangleright"] = 8939,  
 3571 ["NotRightTriangle"] = 8939,  
 3572 ["nltrie"] = 8940,  
 3573 ["ntrianglelefteq"] = 8940,  
 3574 ["NotLeftTriangleEqual"] = 8940,  
 3575 ["nrtrie"] = 8941,  
 3576 ["ntrianglerighteq"] = 8941,  
 3577 ["NotRightTriangleEqual"] = 8941,  
 3578 ["vellip"] = 8942,  
 3579 ["ctdot"] = 8943,  
 3580 ["utdot"] = 8944,  
 3581 ["dtdot"] = 8945,  
 3582 ["disin"] = 8946,  
 3583 ["isinsv"] = 8947,  
 3584 ["isins"] = 8948,  
 3585 ["isindot"] = 8949,  
 3586 ["notinvc"] = 8950,  
 3587 ["notinvb"] = 8951,  
 3588 ["isinE"] = 8953,  
 3589 ["nisd"] = 8954,  
 3590 ["xnis"] = 8955,  
 3591 ["nis"] = 8956,  
 3592 ["notnivc"] = 8957,  
 3593 ["notnivb"] = 8958,

```

3594 ["barwed"] = 8965,
3595 ["barwedge"] = 8965,
3596 ["Barwed"] = 8966,
3597 ["doublebarwedge"] = 8966,
3598 ["lceil"] = 8968,
3599 ["LeftCeiling"] = 8968,
3600 ["rceil"] = 8969,
3601 ["RightCeiling"] = 8969,
3602 ["lfloor"] = 8970,
3603 ["LeftFloor"] = 8970,
3604 ["rfloor"] = 8971,
3605 ["RightFloor"] = 8971,
3606 ["drcrop"] = 8972,
3607 ["dlcrop"] = 8973,
3608 ["urcrop"] = 8974,
3609 ["ulcrop"] = 8975,
3610 ["bnot"] = 8976,
3611 ["proflin"] = 8978,
3612 ["profsurf"] = 8979,
3613 ["telrec"] = 8981,
3614 ["target"] = 8982,
3615 ["ulcorn"] = 8988,
3616 ["ulcorner"] = 8988,
3617 ["urcorn"] = 8989,
3618 ["urcorner"] = 8989,
3619 ["dlcorn"] = 8990,
3620 ["llcorner"] = 8990,
3621 ["drcorn"] = 8991,
3622 ["lrcorn"] = 8991,
3623 ["frown"] = 8994,
3624 ["sfrown"] = 8994,
3625 ["smile"] = 8995,
3626 ["ssmile"] = 8995,
3627 ["cylcty"] = 9005,
3628 ["profalar"] = 9006,
3629 ["topbot"] = 9014,
3630 ["ovbar"] = 9021,
3631 ["solbar"] = 9023,
3632 ["angzarr"] = 9084,
3633 ["lmoust"] = 9136,
3634 ["lmoustache"] = 9136,
3635 ["rmoust"] = 9137,
3636 ["rmoustache"] = 9137,
3637 ["tbrk"] = 9140,
3638 ["OverBracket"] = 9140,
3639 ["bbrk"] = 9141,
3640 ["UnderBracket"] = 9141,

```

```

3641 ["bbrktbrk"] = 9142,
3642 ["OverParenthesis"] = 9180,
3643 ["UnderParenthesis"] = 9181,
3644 ["OverBrace"] = 9182,
3645 ["UnderBrace"] = 9183,
3646 ["trpezium"] = 9186,
3647 ["elinters"] = 9191,
3648 ["blank"] = 9251,
3649 ["oS"] = 9416,
3650 ["circledS"] = 9416,
3651 ["boxh"] = 9472,
3652 ["HorizontalLine"] = 9472,
3653 ["boxv"] = 9474,
3654 ["boxdr"] = 9484,
3655 ["boxdl"] = 9488,
3656 ["boxur"] = 9492,
3657 ["boxul"] = 9496,
3658 ["boxvr"] = 9500,
3659 ["boxvl"] = 9508,
3660 ["boxhd"] = 9516,
3661 ["boxhu"] = 9524,
3662 ["boxvh"] = 9532,
3663 ["boxH"] = 9552,
3664 ["boxV"] = 9553,
3665 ["boxdR"] = 9554,
3666 ["boxDr"] = 9555,
3667 ["boxDR"] = 9556,
3668 ["boxdL"] = 9557,
3669 ["boxDL"] = 9558,
3670 ["boxDL"] = 9559,
3671 ["boxuR"] = 9560,
3672 ["boxUr"] = 9561,
3673 ["boxUR"] = 9562,
3674 ["boxuL"] = 9563,
3675 ["boxU1"] = 9564,
3676 ["boxUL"] = 9565,
3677 ["boxvR"] = 9566,
3678 ["boxVr"] = 9567,
3679 ["boxVR"] = 9568,
3680 ["boxvL"] = 9569,
3681 ["boxV1"] = 9570,
3682 ["boxVL"] = 9571,
3683 ["boxHd"] = 9572,
3684 ["boxhD"] = 9573,
3685 ["boxHD"] = 9574,
3686 ["boxHu"] = 9575,
3687 ["boxhU"] = 9576,

```



```

3688 ["boxHU"] = 9577,
3689 ["boxvH"] = 9578,
3690 ["boxVh"] = 9579,
3691 ["boxVH"] = 9580,
3692 ["uhblk"] = 9600,
3693 ["lhblk"] = 9604,
3694 ["block"] = 9608,
3695 ["blk14"] = 9617,
3696 ["blk12"] = 9618,
3697 ["blk34"] = 9619,
3698 ["squ"] = 9633,
3699 ["square"] = 9633,
3700 ["Square"] = 9633,
3701 ["squf"] = 9642,
3702 ["squarf"] = 9642,
3703 ["blacksquare"] = 9642,
3704 ["FilledVerySmallSquare"] = 9642,
3705 ["EmptyVerySmallSquare"] = 9643,
3706 ["rect"] = 9645,
3707 ["marker"] = 9646,
3708 ["fltns"] = 9649,
3709 ["xutri"] = 9651,
3710 ["bigtriangleup"] = 9651,
3711 ["utrif"] = 9652,
3712 ["blacktriangle"] = 9652,
3713 ["utri"] = 9653,
3714 ["triangle"] = 9653,
3715 ["rtrif"] = 9656,
3716 ["blacktriangleright"] = 9656,
3717 ["rtri"] = 9657,
3718 ["triangleright"] = 9657,
3719 ["xdtri"] = 9661,
3720 ["bigtriangledown"] = 9661,
3721 ["dtrif"] = 9662,
3722 ["blacktriangledown"] = 9662,
3723 ["dtri"] = 9663,
3724 ["triangledown"] = 9663,
3725 ["ltrif"] = 9666,
3726 ["blacktriangleleft"] = 9666,
3727 ["ltri"] = 9667,
3728 ["triangleleft"] = 9667,
3729 ["loz"] = 9674,
3730 ["lozenge"] = 9674,
3731 ["cir"] = 9675,
3732 ["tridot"] = 9708,
3733 ["xcirc"] = 9711,
3734 ["bigcirc"] = 9711,

```

```

3735 ["ultri"] = 9720,
3736 ["urtri"] = 9721,
3737 ["lltri"] = 9722,
3738 ["EmptySmallSquare"] = 9723,
3739 ["FilledSmallSquare"] = 9724,
3740 ["starf"] = 9733,
3741 ["bigstar"] = 9733,
3742 ["star"] = 9734,
3743 ["phone"] = 9742,
3744 ["female"] = 9792,
3745 ["male"] = 9794,
3746 ["spades"] = 9824,
3747 ["spadesuit"] = 9824,
3748 ["clubs"] = 9827,
3749 ["clubsuit"] = 9827,
3750 ["hearts"] = 9829,
3751 ["heartsuit"] = 9829,
3752 ["diams"] = 9830,
3753 ["diamondsuit"] = 9830,
3754 ["sung"] = 9834,
3755 ["flat"] = 9837,
3756 ["natur"] = 9838,
3757 ["natural"] = 9838,
3758 ["sharp"] = 9839,
3759 ["check"] = 10003,
3760 ["checkmark"] = 10003,
3761 ["cross"] = 10007,
3762 ["malt"] = 10016,
3763 ["maltese"] = 10016,
3764 ["sext"] = 10038,
3765 ["VerticalSeparator"] = 10072,
3766 ["lbrk"] = 10098,
3767 ["rbrk"] = 10099,
3768 ["lobrk"] = 10214,
3769 ["LeftDoubleBracket"] = 10214,
3770 ["robrk"] = 10215,
3771 ["RightDoubleBracket"] = 10215,
3772 ["lang"] = 10216,
3773 ["LeftAngleBracket"] = 10216,
3774 ["langle"] = 10216,
3775 ["rang"] = 10217,
3776 ["RightAngleBracket"] = 10217,
3777 ["rangle"] = 10217,
3778 ["Lang"] = 10218,
3779 ["Rang"] = 10219,
3780 ["loang"] = 10220,
3781 ["roang"] = 10221,

```

```

3782 ["xlarr"] = 10229,
3783 ["longleftarrow"] = 10229,
3784 ["LongLeftArrow"] = 10229,
3785 ["xrarr"] = 10230,
3786 ["longrightarrow"] = 10230,
3787 ["LongRightArrow"] = 10230,
3788 ["xharr"] = 10231,
3789 ["longlefttrightarrow"] = 10231,
3790 ["LongLeftRightArrow"] = 10231,
3791 ["xlArr"] = 10232,
3792 ["Longleftarrow"] = 10232,
3793 ["DoubleLongLeftArrow"] = 10232,
3794 ["xrArr"] = 10233,
3795 ["Longrightarrow"] = 10233,
3796 ["DoubleLongRightArrow"] = 10233,
3797 ["xhArr"] = 10234,
3798 ["Longlefttrightarrow"] = 10234,
3799 ["DoubleLongLeftRightArrow"] = 10234,
3800 ["xmap"] = 10236,
3801 ["longmapsto"] = 10236,
3802 ["dzigrarr"] = 10239,
3803 ["nvlArr"] = 10498,
3804 ["nvrArr"] = 10499,
3805 ["nvHarr"] = 10500,
3806 ["Map"] = 10501,
3807 ["lbarr"] = 10508,
3808 ["rbarr"] = 10509,
3809 ["bkarow"] = 10509,
3810 ["lBarr"] = 10510,
3811 ["rBarr"] = 10511,
3812 ["dbkarow"] = 10511,
3813 ["RBarr"] = 10512,
3814 ["drbkarow"] = 10512,
3815 ["DDotrahd"] = 10513,
3816 ["UpArrowBar"] = 10514,
3817 ["DownArrowBar"] = 10515,
3818 ["Rarrtl"] = 10518,
3819 ["latail"] = 10521,
3820 ["ratail"] = 10522,
3821 ["lAtail"] = 10523,
3822 ["rAtail"] = 10524,
3823 ["larrfs"] = 10525,
3824 ["rarrfs"] = 10526,
3825 ["larrbfs"] = 10527,
3826 ["rarrbfs"] = 10528,
3827 ["nwarhk"] = 10531,
3828 ["nearhk"] = 10532,

```

```

3829 ["searhk"] = 10533,
3830 ["hksearow"] = 10533,
3831 ["swarhk"] = 10534,
3832 ["hkswarow"] = 10534,
3833 ["nwnear"] = 10535,
3834 ["nesear"] = 10536,
3835 ["toea"] = 10536,
3836 ["seswar"] = 10537,
3837 ["tosa"] = 10537,
3838 ["swnwar"] = 10538,
3839 ["rarrc"] = 10547,
3840 ["cudarr"] = 10549,
3841 ["ldca"] = 10550,
3842 ["rdca"] = 10551,
3843 ["cudarrl"] = 10552,
3844 ["larrpl"] = 10553,
3845 ["curarrm"] = 10556,
3846 ["cularrp"] = 10557,
3847 ["rarrpl"] = 10565,
3848 ["harrcir"] = 10568,
3849 ["Uarrocir"] = 10569,
3850 ["lurdshar"] = 10570,
3851 ["ldrushar"] = 10571,
3852 ["LeftRightVector"] = 10574,
3853 ["RightUpDownVector"] = 10575,
3854 ["DownLeftRightVector"] = 10576,
3855 ["LeftUpDownVector"] = 10577,
3856 ["LeftVectorBar"] = 10578,
3857 ["RightVectorBar"] = 10579,
3858 ["RightUpVectorBar"] = 10580,
3859 ["RightDownVectorBar"] = 10581,
3860 ["DownLeftVectorBar"] = 10582,
3861 ["DownRightVectorBar"] = 10583,
3862 ["LeftUpVectorBar"] = 10584,
3863 ["LeftDownVectorBar"] = 10585,
3864 ["LeftTeeVector"] = 10586,
3865 ["RightTeeVector"] = 10587,
3866 ["RightUpTeeVector"] = 10588,
3867 ["RightDownTeeVector"] = 10589,
3868 ["DownLeftTeeVector"] = 10590,
3869 ["DownRightTeeVector"] = 10591,
3870 ["LeftUpTeeVector"] = 10592,
3871 ["LeftDownTeeVector"] = 10593,
3872 ["lHar"] = 10594,
3873 ["uHar"] = 10595,
3874 ["rHar"] = 10596,
3875 ["dHar"] = 10597,

```

```

3876 ["luruhar"] = 10598,
3877 ["ldrdhar"] = 10599,
3878 ["ruluhar"] = 10600,
3879 ["rdldhar"] = 10601,
3880 ["lharul"] = 10602,
3881 ["llhard"] = 10603,
3882 ["rharul"] = 10604,
3883 ["lrhard"] = 10605,
3884 ["udhar"] = 10606,
3885 ["UpEquilibrium"] = 10606,
3886 ["duhar"] = 10607,
3887 ["ReverseUpEquilibrium"] = 10607,
3888 ["RoundImplies"] = 10608,
3889 ["erarr"] = 10609,
3890 ["simrarr"] = 10610,
3891 ["larrsim"] = 10611,
3892 ["rarrsim"] = 10612,
3893 ["rarrap"] = 10613,
3894 ["ltlarr"] = 10614,
3895 ["gtrarr"] = 10616,
3896 ["subrarr"] = 10617,
3897 ["suplarr"] = 10619,
3898 ["lfisht"] = 10620,
3899 ["rfisht"] = 10621,
3900 ["ufisht"] = 10622,
3901 ["dfisht"] = 10623,
3902 ["lopar"] = 10629,
3903 ["ropar"] = 10630,
3904 ["lbrke"] = 10635,
3905 ["rbrke"] = 10636,
3906 ["lbrkslu"] = 10637,
3907 ["rbrksld"] = 10638,
3908 ["lbrksld"] = 10639,
3909 ["rbrkslu"] = 10640,
3910 ["langd"] = 10641,
3911 ["rangd"] = 10642,
3912 ["lparlt"] = 10643,
3913 ["rpargt"] = 10644,
3914 ["gtlPar"] = 10645,
3915 ["ltrPar"] = 10646,
3916 ["vzigzag"] = 10650,
3917 ["vangrt"] = 10652,
3918 ["angrtvbd"] = 10653,
3919 ["ange"] = 10660,
3920 ["range"] = 10661,
3921 ["dwangle"] = 10662,
3922 ["uwangle"] = 10663,

```

```

3923 ["angmsdaa"] = 10664,
3924 ["angmsdab"] = 10665,
3925 ["angmsdac"] = 10666,
3926 ["angmsdad"] = 10667,
3927 ["angmsdae"] = 10668,
3928 ["angmsdaf"] = 10669,
3929 ["angmsdag"] = 10670,
3930 ["angmsdah"] = 10671,
3931 ["bemptyv"] = 10672,
3932 ["demptyv"] = 10673,
3933 ["cemptyv"] = 10674,
3934 ["raemptyv"] = 10675,
3935 ["laemptyv"] = 10676,
3936 ["ohbar"] = 10677,
3937 ["omid"] = 10678,
3938 ["opar"] = 10679,
3939 ["operp"] = 10681,
3940 ["olcross"] = 10683,
3941 ["odsold"] = 10684,
3942 ["olcir"] = 10686,
3943 ["ofcir"] = 10687,
3944 ["olt"] = 10688,
3945 ["ogt"] = 10689,
3946 ["cirscir"] = 10690,
3947 ["cirE"] = 10691,
3948 ["solb"] = 10692,
3949 ["bsolb"] = 10693,
3950 ["boxbox"] = 10697,
3951 ["trish"] = 10701,
3952 ["rtriltri"] = 10702,
3953 ["LeftTriangleBar"] = 10703,
3954 ["RightTriangleBar"] = 10704,
3955 ["race"] = 10714,
3956 ["iinfin"] = 10716,
3957 ["infintie"] = 10717,
3958 ["nvinfin"] = 10718,
3959 ["eparsl"] = 10723,
3960 ["smeparsl"] = 10724,
3961 ["eqvparsl"] = 10725,
3962 ["lozf"] = 10731,
3963 ["blacklozenge"] = 10731,
3964 ["RuleDelayed"] = 10740,
3965 ["dsol"] = 10742,
3966 ["xodot"] = 10752,
3967 ["bigodot"] = 10752,
3968 ["xoplus"] = 10753,
3969 ["bigoplus"] = 10753,

```

```

3970 ["xotime"] = 10754,
3971 ["bigotimes"] = 10754,
3972 ["xuplus"] = 10756,
3973 ["biguplus"] = 10756,
3974 ["xsqcup"] = 10758,
3975 ["bigsqcup"] = 10758,
3976 ["qint"] = 10764,
3977 ["iiiint"] = 10764,
3978 ["fpartint"] = 10765,
3979 ["cirfnint"] = 10768,
3980 ["awint"] = 10769,
3981 ["rppointint"] = 10770,
3982 ["scpointint"] = 10771,
3983 ["npointint"] = 10772,
3984 ["pointint"] = 10773,
3985 ["quatint"] = 10774,
3986 ["intlarhk"] = 10775,
3987 ["pluscir"] = 10786,
3988 ["plusacir"] = 10787,
3989 ["simplus"] = 10788,
3990 ["plusdu"] = 10789,
3991 ["plussim"] = 10790,
3992 ["plustwo"] = 10791,
3993 ["mcomma"] = 10793,
3994 ["minusdu"] = 10794,
3995 ["loplus"] = 10797,
3996 ["roplus"] = 10798,
3997 ["Cross"] = 10799,
3998 ["timesd"] = 10800,
3999 ["timesbar"] = 10801,
4000 ["smashp"] = 10803,
4001 ["lotimes"] = 10804,
4002 ["rotimes"] = 10805,
4003 ["otimesas"] = 10806,
4004 ["Otimes"] = 10807,
4005 ["odiv"] = 10808,
4006 ["triplus"] = 10809,
4007 ["triminus"] = 10810,
4008 ["tritime"] = 10811,
4009 ["iproduct"] = 10812,
4010 ["intproduct"] = 10812,
4011 ["amalg"] = 10815,
4012 ["capdot"] = 10816,
4013 ["ncup"] = 10818,
4014 ["ncap"] = 10819,
4015 ["capand"] = 10820,
4016 ["cupor"] = 10821,

```

```

4017 ["cupcap"] = 10822,
4018 ["capcup"] = 10823,
4019 ["cupbrcap"] = 10824,
4020 ["capbrcup"] = 10825,
4021 ["cupcup"] = 10826,
4022 ["capcap"] = 10827,
4023 ["ccups"] = 10828,
4024 ["ccaps"] = 10829,
4025 ["ccupssm"] = 10832,
4026 ["And"] = 10835,
4027 ["Or"] = 10836,
4028 ["andand"] = 10837,
4029 ["oror"] = 10838,
4030 ["orslope"] = 10839,
4031 ["andslope"] = 10840,
4032 ["andv"] = 10842,
4033 ["orv"] = 10843,
4034 ["andd"] = 10844,
4035 ["ord"] = 10845,
4036 ["wedbar"] = 10847,
4037 ["sdote"] = 10854,
4038 ["simdot"] = 10858,
4039 ["congdote"] = 10861,
4040 ["easter"] = 10862,
4041 ["apacir"] = 10863,
4042 ["apE"] = 10864,
4043 ["eplus"] = 10865,
4044 ["pluse"] = 10866,
4045 ["Esim"] = 10867,
4046 ["Colone"] = 10868,
4047 ["Equal"] = 10869,
4048 ["eDDot"] = 10871,
4049 ["ddotseq"] = 10871,
4050 ["equivDD"] = 10872,
4051 ["ltcir"] = 10873,
4052 ["gtcir"] = 10874,
4053 ["ltquest"] = 10875,
4054 ["gtquest"] = 10876,
4055 ["les"] = 10877,
4056 ["LessSlantEqual"] = 10877,
4057 ["leqslant"] = 10877,
4058 ["ges"] = 10878,
4059 ["GreaterSlantEqual"] = 10878,
4060 ["geqslant"] = 10878,
4061 ["lesdot"] = 10879,
4062 ["gesdot"] = 10880,
4063 ["lesdoto"] = 10881,

```



```

4064 ["gesdoto"] = 10882,
4065 ["lesdotor"] = 10883,
4066 ["gesdoto1"] = 10884,
4067 ["lap"] = 10885,
4068 ["lessapprox"] = 10885,
4069 ["gap"] = 10886,
4070 ["gtrapprox"] = 10886,
4071 ["lne"] = 10887,
4072 ["lneq"] = 10887,
4073 ["gne"] = 10888,
4074 ["gneq"] = 10888,
4075 ["lnap"] = 10889,
4076 ["lnapprox"] = 10889,
4077 ["gnap"] = 10890,
4078 ["gnapprox"] = 10890,
4079 ["lEg"] = 10891,
4080 ["lesseqqgtr"] = 10891,
4081 ["gEl"] = 10892,
4082 ["gtreqqless"] = 10892,
4083 ["lsime"] = 10893,
4084 ["gsime"] = 10894,
4085 ["lsimg"] = 10895,
4086 ["gsiml"] = 10896,
4087 ["lgE"] = 10897,
4088 ["glE"] = 10898,
4089 ["lesges"] = 10899,
4090 ["gesles"] = 10900,
4091 ["els"] = 10901,
4092 ["eqslantless"] = 10901,
4093 ["egs"] = 10902,
4094 ["eqslantgtr"] = 10902,
4095 ["elsdot"] = 10903,
4096 ["egsdot"] = 10904,
4097 ["el"] = 10905,
4098 ["eg"] = 10906,
4099 ["siml"] = 10909,
4100 ["sing"] = 10910,
4101 ["simlE"] = 10911,
4102 ["singE"] = 10912,
4103 ["LessLess"] = 10913,
4104 ["GreaterGreater"] = 10914,
4105 ["glj"] = 10916,
4106 ["gla"] = 10917,
4107 ["ltcc"] = 10918,
4108 ["gtcc"] = 10919,
4109 ["lescc"] = 10920,
4110 ["gescc"] = 10921,

```

```

4111 ["smt"] = 10922,
4112 ["lat"] = 10923,
4113 ["smtE"] = 10924,
4114 ["late"] = 10925,
4115 ["bumpE"] = 10926,
4116 ["pre"] = 10927,
4117 ["preceq"] = 10927,
4118 ["PrecedesEqual"] = 10927,
4119 ["sce"] = 10928,
4120 ["succeq"] = 10928,
4121 ["SucceedsEqual"] = 10928,
4122 ["prE"] = 10931,
4123 ["scE"] = 10932,
4124 ["prnE"] = 10933,
4125 ["precneqq"] = 10933,
4126 ["scnE"] = 10934,
4127 ["succneqq"] = 10934,
4128 ["prap"] = 10935,
4129 ["precapprox"] = 10935,
4130 ["scap"] = 10936,
4131 ["succapprox"] = 10936,
4132 ["prnap"] = 10937,
4133 ["precnapprox"] = 10937,
4134 ["scnap"] = 10938,
4135 ["succnapprox"] = 10938,
4136 ["Pr"] = 10939,
4137 ["Sc"] = 10940,
4138 ["subdot"] = 10941,
4139 ["supdot"] = 10942,
4140 ["subplus"] = 10943,
4141 ["supplus"] = 10944,
4142 ["submult"] = 10945,
4143 ["supmult"] = 10946,
4144 ["subedot"] = 10947,
4145 ["supedot"] = 10948,
4146 ["subE"] = 10949,
4147 ["subseteqq"] = 10949,
4148 ["supE"] = 10950,
4149 ["supseteqq"] = 10950,
4150 ["subsim"] = 10951,
4151 ["supsim"] = 10952,
4152 ["subnE"] = 10955,
4153 ["subsetneqq"] = 10955,
4154 ["supnE"] = 10956,
4155 ["supsetneqq"] = 10956,
4156 ["csub"] = 10959,
4157 ["csup"] = 10960,

```

```

4158 ["csube"] = 10961,
4159 ["csupe"] = 10962,
4160 ["subsup"] = 10963,
4161 ["supsub"] = 10964,
4162 ["subsub"] = 10965,
4163 ["supsup"] = 10966,
4164 ["suphsub"] = 10967,
4165 ["supdsub"] = 10968,
4166 ["forkv"] = 10969,
4167 ["topfork"] = 10970,
4168 ["mlcp"] = 10971,
4169 ["Dashv"] = 10980,
4170 ["DoubleLeftTee"] = 10980,
4171 ["Vdashl"] = 10982,
4172 ["Barv"] = 10983,
4173 ["vBar"] = 10984,
4174 ["vBarv"] = 10985,
4175 ["Vbar"] = 10987,
4176 ["Not"] = 10988,
4177 ["bNot"] = 10989,
4178 ["rnmid"] = 10990,
4179 ["cirmid"] = 10991,
4180 ["midcir"] = 10992,
4181 ["topcir"] = 10993,
4182 ["nhpar"] = 10994,
4183 ["parsim"] = 10995,
4184 ["parsl"] = 11005,
4185 ["fflig"] = 64256,
4186 ["filig"] = 64257,
4187 ["fllig"] = 64258,
4188 ["ffilig"] = 64259,
4189 ["ffllig"] = 64260,
4190 ["Ascr"] = 119964,
4191 ["Cscr"] = 119966,
4192 ["Dscr"] = 119967,
4193 ["Gscr"] = 119970,
4194 ["Jscr"] = 119973,
4195 ["Kscr"] = 119974,
4196 ["Nscr"] = 119977,
4197 ["Oscr"] = 119978,
4198 ["Pscr"] = 119979,
4199 ["Qscr"] = 119980,
4200 ["Sscr"] = 119982,
4201 ["Tscr"] = 119983,
4202 ["Uscr"] = 119984,
4203 ["Vscr"] = 119985,
4204 ["Wscr"] = 119986,

```

```

4205 ["Xscr"] = 119987,
4206 ["Yscr"] = 119988,
4207 ["Zscr"] = 119989,
4208 ["ascr"] = 119990,
4209 ["bscr"] = 119991,
4210 ["cscr"] = 119992,
4211 ["dscr"] = 119993,
4212 ["fscr"] = 119995,
4213 ["hscr"] = 119997,
4214 ["iscr"] = 119998,
4215 ["jscr"] = 119999,
4216 ["kscr"] = 120000,
4217 ["lscr"] = 120001,
4218 ["mscr"] = 120002,
4219 ["nscr"] = 120003,
4220 ["pscr"] = 120005,
4221 ["qscr"] = 120006,
4222 ["rscr"] = 120007,
4223 ["sscr"] = 120008,
4224 ["tscr"] = 120009,
4225 ["uscr"] = 120010,
4226 ["vscr"] = 120011,
4227 ["wscr"] = 120012,
4228 ["xscr"] = 120013,
4229 ["yscr"] = 120014,
4230 ["zscr"] = 120015,
4231 ["Afr"] = 120068,
4232 ["Bfr"] = 120069,
4233 ["Dfr"] = 120071,
4234 ["Efr"] = 120072,
4235 ["Ffr"] = 120073,
4236 ["Gfr"] = 120074,
4237 ["Jfr"] = 120077,
4238 ["Kfr"] = 120078,
4239 ["Lfr"] = 120079,
4240 ["Mfr"] = 120080,
4241 ["Nfr"] = 120081,
4242 ["Ofrr"] = 120082,
4243 ["Pfr"] = 120083,
4244 ["Qfr"] = 120084,
4245 ["Sfr"] = 120086,
4246 ["Tfr"] = 120087,
4247 ["Ufr"] = 120088,
4248 ["Vfr"] = 120089,
4249 ["Wfr"] = 120090,
4250 ["Xfr"] = 120091,
4251 ["Yfr"] = 120092,

```

```

4252 ["afr"] = 120094,
4253 ["bfr"] = 120095,
4254 ["cfr"] = 120096,
4255 ["dfr"] = 120097,
4256 ["efr"] = 120098,
4257 ["ffr"] = 120099,
4258 ["gfr"] = 120100,
4259 ["hfr"] = 120101,
4260 ["ifr"] = 120102,
4261 ["jfr"] = 120103,
4262 ["kfr"] = 120104,
4263 ["lfr"] = 120105,
4264 ["mfr"] = 120106,
4265 ["nfr"] = 120107,
4266 ["ofr"] = 120108,
4267 ["pfr"] = 120109,
4268 ["qfr"] = 120110,
4269 ["rfr"] = 120111,
4270 ["sfr"] = 120112,
4271 ["tfr"] = 120113,
4272 ["ufr"] = 120114,
4273 ["vfr"] = 120115,
4274 ["wfr"] = 120116,
4275 ["xfr"] = 120117,
4276 ["yfr"] = 120118,
4277 ["zfr"] = 120119,
4278 ["Aopf"] = 120120,
4279 ["Bopf"] = 120121,
4280 ["Dopf"] = 120123,
4281 ["Eopf"] = 120124,
4282 ["Fopf"] = 120125,
4283 ["Gopf"] = 120126,
4284 ["Iopf"] = 120128,
4285 ["Jopf"] = 120129,
4286 ["Kopf"] = 120130,
4287 ["Lopf"] = 120131,
4288 ["Mopf"] = 120132,
4289 ["Oopf"] = 120134,
4290 ["Sopf"] = 120138,
4291 ["Topf"] = 120139,
4292 ["Uopf"] = 120140,
4293 ["Vopf"] = 120141,
4294 ["Wopf"] = 120142,
4295 ["Xopf"] = 120143,
4296 ["Yopf"] = 120144,
4297 ["aopf"] = 120146,
4298 ["bopf"] = 120147,

```

```

4299 ["copf"] = 120148,
4300 ["dopf"] = 120149,
4301 ["eopf"] = 120150,
4302 ["fopf"] = 120151,
4303 ["gopf"] = 120152,
4304 ["hopf"] = 120153,
4305 ["iopf"] = 120154,
4306 ["jopf"] = 120155,
4307 ["kopf"] = 120156,
4308 ["lopf"] = 120157,
4309 ["mopf"] = 120158,
4310 ["nopf"] = 120159,
4311 ["oopf"] = 120160,
4312 ["popf"] = 120161,
4313 ["qopf"] = 120162,
4314 ["ropf"] = 120163,
4315 ["sopf"] = 120164,
4316 ["topf"] = 120165,
4317 ["uopf"] = 120166,
4318 ["vopf"] = 120167,
4319 ["wopf"] = 120168,
4320 ["xopf"] = 120169,
4321 ["yopf"] = 120170,
4322 ["zopf"] = 120171,
4323 }

```

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

```

4324 function entities.dec_entity(s)
4325     return unicode.utf8.char(tonumber(s))
4326 end

```

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

```

4327 function entities.hex_entity(s)
4328     return unicode.utf8.char(tonumber("0x"..s))
4329 end

```

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

```

4330 function entities.char_entity(s)
4331     local n = character_entities[s]
4332     if n == nil then
4333         return "&" .. s .. ";"
4334     end
4335     return unicode.utf8.char(n)
4336 end

```

### 3.1.3 Plain TeX Writer

This section documents the `writer` object, which implements the routines for producing the TeX output. The object is an amalgamate of the generic, TeX, LaTeX writer objects that were located in the `lunamark/writer/generic.lua`, `lunamark/writer/tex.lua`, and `lunamark/writer/latex.lua` files in the Lunamark 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.

```
4337 M.writer = {}
```

The `writer.new` method creates and returns a new TeX 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.

```
4338 function M.writer.new(options)
4339   local self = {}
```

Make `options` available as `writer->options`, so that it is accessible from extensions.

```
4340   self.options = options
```

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

```
4341   local slice_specifiers = {}
4342   for specifier in options.slice:gmatch("[^%s]+") do
4343     table.insert(slice_specifiers, specifier)
4344   end
4345
4346   if #slice_specifiers == 2 then
4347     self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
4348     local slice_begin_type = self.slice_begin:sub(1, 1)
4349     if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
4350       self.slice_begin = "^" .. self.slice_begin
4351     end
4352     local slice_end_type = self.slice_end:sub(1, 1)
4353     if slice_end_type ~= "^" and slice_end_type ~= "$" then
4354       self.slice_end = "$" .. self.slice_end
4355     end
4356   elseif #slice_specifiers == 1 then
4357     self.slice_begin = "^" .. slice_specifiers[1]
```

```

4358     self.slice_end = "$" .. slice_specifiers[1]
4359 end
4360
4361 if self.slice_begin == "^" and self.slice_end ~= "^" then
4362     self.is_writing = true
4363 else
4364     self.is_writing = false
4365 end

```

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

```

4366     self.suffix = ".tex"

```

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

```

4367     self.space = " "

```

Define **writer->nbsp** as the output format of a non-breaking space character.

```

4368     self.nbsp = "\\markdownRendererNbsp{}"

```

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

```

4369     function self.plain(s)
4370         return s
4371     end

```

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

```

4372     function self.paragraph(s)
4373         if not self.is_writing then return "" end
4374         return s
4375     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.

```

4376     function self.pack(name)
4377         return "[\\input ]" .. name .. "[\\relax]"
4378     end

```

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

```

4379     function self.interblocksep()
4380         if not self.is_writing then return "" end
4381         return "\\markdownRendererInterblockSeparator\\n{}"
4382     end

```

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

```

4383     self.linebreak = "\\markdownRendererLineBreak\\n{}"

```

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

```

4384     self.ellipsis = "\\markdownRendererEllipsis{}"

```



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

```
4385 function self.hrule()
4386     if not self.is_writing then return "" end
4387     return "\\markdownRendererHorizontalRule{}"
4388 end
```

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

```
4389 self.escaped_uri_chars = {
4390     ["{"] = "\\markdownRendererLeftBrace{}",
4391     ["}"] = "\\markdownRendererRightBrace{}",
4392     ["%"] = "\\markdownRendererPercentSign{}",
4393     ["\\"] = "\\markdownRendererBackslash{}",
4394 }
4395 self.escaped_minimal_strings = {
4396     ["^"] = "\\markdownRendererCircumflex\\markdownRendererCircumflex ",
4397     ["☒"] = "\\markdownRendererTickedBox{}",
4398     ["◻"] = "\\markdownRendererHalfTickedBox{}",
4399     ["□"] = "\\markdownRendererUntickedBox{}",
4400 }
```

Define a table `writer->escaped_chars` containing the mapping from special plain TeX characters (including the active pipe character (`|`) of ConTeXt) that need to be escaped for typeset content.

```
4401 self.escaped_chars = {
4402     ["{"] = "\\markdownRendererLeftBrace{}",
4403     ["}"] = "\\markdownRendererRightBrace{}",
4404     ["%"] = "\\markdownRendererPercentSign{}",
4405     ["\\"] = "\\markdownRendererBackslash{}",
4406     ["#"] = "\\markdownRendererHash{}",
4407     ["$"] = "\\markdownRendererDollarSign{}",
4408     ["&"] = "\\markdownRendererAmpersand{}",
4409     ["_"] = "\\markdownRendererUnderscore{}",
4410     ["^"] = "\\markdownRendererCircumflex{}",
4411     ["~"] = "\\markdownRendererTilde{}",
4412     ["|"] = "\\markdownRendererPipe{}",
4413 }
```

Use the `writer->escaped_chars`, `writer->escaped_uri_chars`, and `writer->escaped_minimal_strings` tables to create the `writer->escape`, `writer->escape_uri`, and `writer->escape_minimal` escaper functions.

```
4414 self.escape = util.escaper(self.escaped_chars, self.escaped_minimal_strings)
4415 self.escape_uri = util.escaper(self.escaped_uri_chars, self.escaped_minimal_strings)
4416 self.escape_minimal = util.escaper({}, self.escaped_minimal_strings)
```

Define `writer->string` as a function that will transform an input plain text span `s` to the output format and `writer->uri` as a function that will trans-

form an input URI `u` to the output format. If the `hybrid` option is enabled, use the `writer->escape_minimal`. Otherwise, use the `writer->escape`, and `writer->escape_uri` functions.

```
4417 if options.hybrid then
4418     self.string = self.escape_minimal
4419     self.uri = self.escape_minimal
4420 else
4421     self.string = self.escape
4422     self.uri = self.escape_uri
4423 end
```

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

```
4424 function self.code(s)
4425     return {"\\markdownRendererCodeSpan{",self.escape(s),"}"}
4426 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.

```
4427 function self.link(lab,src,tit)
4428     return {"\\markdownRendererLink{",lab,"}",
4429             "{",self.escape(src),"}",
4430             "{",self.uri(src),"}",
4431             "{",self.string(tit or ""),"}"}
4432 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.

```
4433 function self.image(lab,src,tit)
4434     return {"\\markdownRendererImage{",lab,"}",
4435             "{",self.string(src),"}",
4436             "{",self.uri(src),"}",
4437             "{",self.string(tit or ""),"}"}
4438 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.

```
4439 function self.bulletlist(items,tight)
4440     if not self.is_writing then return "" end
4441     local buffer = {}
4442     for _,item in ipairs(items) do
4443         buffer[#buffer + 1] = self.bulletitem(item)
4444     end
4445     local contents = util.intersperse(buffer,"\n")
```

```

4446     if tight and options.tightLists then
4447         return {"\\markdownRendererUlBeginTight\n",contents,
4448             "\n\\markdownRendererUlEndTight "}
4449     else
4450         return {"\\markdownRendererUlBegin\n",contents,
4451             "\n\\markdownRendererUlEnd "}
4452     end
4453 end

```

Define `writer->bulletitem` as a function that will transform an input bulleted list item to the output format, where `s` is the text of the list item.

```

4454 function self.bulletitem(s)
4455     return {"\\markdownRendererUlItem ",s,
4456         "\\markdownRendererUlItemEnd "}
4457 end

```

Define `writer->orderedlist` 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 is the number of the first list item.

```

4458 function self.orderedlist(items,tight,startnum)
4459     if not self.is_writing then return "" end
4460     local buffer = {}
4461     local num = startnum
4462     for _,item in ipairs(items) do
4463         buffer[#buffer + 1] = self.ordereditem(item,num)
4464         if num ~= nil then
4465             num = num + 1
4466         end
4467     end
4468     local contents = util.intersperse(buffer,"\n")
4469     if tight and options.tightLists then
4470         return {"\\markdownRendererOlBeginTight\n",contents,
4471             "\n\\markdownRendererOlEndTight "}
4472     else
4473         return {"\\markdownRendererOlBegin\n",contents,
4474             "\n\\markdownRendererOlEnd "}
4475     end
4476 end

```

Define `writer->ordereditem` as a function that will transform an input ordered list item to the output format, where `s` is the text of the list item. If the optional parameter `num` is present, it is the number of the list item.

```

4477 function self.ordereditem(s,num)
4478     if num ~= nil then
4479         return {"\\markdownRendererOlItemWithNumber{",num,"}",s,
4480             "\\markdownRendererOlItemEnd "}
4481     else

```

```

4482         return {"\\markdownRendererOlItem ",s,
4483                 "\\markdownRendererOlItemEnd "}
4484     end
4485 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.

```

4486 function self.inline_html_comment(contents)
4487     return {"\\markdownRendererInlineHtmlComment{",contents,""}
4488 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.

```

4489 function self.block_html_comment(contents)
4490     if not self.is_writing then return "" end
4491     return {"\\markdownRendererBlockHtmlCommentBegin\\n",contents,
4492           "\\n\\markdownRendererBlockHtmlCommentEnd "}
4493 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.

```

4494 function self.inline_html_tag(contents)
4495     return {"\\markdownRendererInlineHtmlTag{",self.string(contents),""}
4496 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.

```

4497 function self.block_html_element(s)
4498     if not self.is_writing then return "" end
4499     local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
4500     return {"\\markdownRendererInputBlockHtmlElement{",name,""}
4501 end

```

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

```

4502 function self.emphasis(s)
4503     return {"\\markdownRendererEmphasis{",s,""}
4504 end

```

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

```

4505 function self.tickbox(f)
4506     if f == 1.0 then
4507         return "☒ "

```

```

4508     elseif f == 0.0 then
4509         return "□ "
4510     else
4511         return "◻ "
4512     end
4513 end

```

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

```

4514 function self.strong(s)
4515     return {"\\markdownRendererStrongEmphasis{" ,s,"}"}
4516 end

```

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

```

4517 function self.blockquote(s)
4518     if #util.ropetostring(s) == 0 then return "" end
4519     return {"\\markdownRendererBlockQuoteBegin\\n",s,
4520         "\\n\\markdownRendererBlockQuoteEnd "}
4521 end

```

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

```

4522 function self.verbatim(s)
4523     if not self.is_writing then return "" end
4524     s = string.gsub(s, '[\\r\\n%s]*$', '')
4525     local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
4526     return {"\\markdownRendererInputVerbatim{" ,name,"}"}
4527 end

```

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

```

4528 function self.document(d)
4529     local active_attributes = self.active_attributes
4530     local buf = {"\\markdownRendererDocumentBegin\\n", d}
4531
4532     -- pop attributes for sections that have ended
4533     if options.headerAttributes and self.is_writing then
4534         while #active_attributes > 0 do
4535             local attributes = active_attributes[#active_attributes]
4536             if #attributes > 0 then
4537                 table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4538             end
4539             table.remove(active_attributes, #active_attributes)
4540         end
4541     end
4542
4543     table.insert(buf, "\\markdownRendererDocumentEnd")

```

```

4544
4545     return buf
4546 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.

```

4547 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.

```

4548 function self.heading(s, level, attributes)
4549     attributes = attributes or {}
4550     for i = 1, #attributes do
4551         attributes[attributes[i]] = true
4552     end
4553
4554     local active_attributes = self.active_attributes
4555     local slice_begin_type = self.slice_begin:sub(1, 1)
4556     local slice_begin_identifier = self.slice_begin:sub(2) or ""
4557     local slice_end_type = self.slice_end:sub(1, 1)
4558     local slice_end_identifier = self.slice_end:sub(2) or ""
4559
4560     local buf = {}
4561
4562     -- push empty attributes for implied sections
4563     while #active_attributes < level-1 do
4564         table.insert(active_attributes, {})
4565     end
4566
4567     -- pop attributes for sections that have ended
4568     while #active_attributes >= level do
4569         local active_identifiers = active_attributes[#active_attributes]
4570         -- tear down all active attributes at slice end
4571         if active_identifiers["#" .. slice_end_identifier] ~= nil
4572             and slice_end_type == "$" then
4573             for header_level = #active_attributes, 1, -1 do
4574                 if options.headerAttributes and #active_attributes[header_level] > 0 then
4575                     table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4576                 end
4577             end
4578             self.is_writing = false
4579         end
4580         table.remove(active_attributes, #active_attributes)
4581         if self.is_writing and options.headerAttributes and #active_identifiers > 0 then
4582             table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4583         end
4584         -- apply all active attributes at slice beginning
4585         if active_identifiers["#" .. slice_begin_identifier] ~= nil

```

```

4586         and slice_begin_type == "$" then
4587         for header_level = 1, #active_attributes do
4588             if options.headerAttributes and #active_attributes[header_level] > 0 then
4589                 table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
4590             end
4591         end
4592         self.is_writing = true
4593     end
4594 end
4595
4596 -- tear down all active attributes at slice end
4597 if attributes["#" .. slice_end_identfier] ~= nil
4598     and slice_end_type == "^" then
4599     for header_level = #active_attributes, 1, -1 do
4600         if options.headerAttributes and #active_attributes[header_level] > 0 then
4601             table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4602         end
4603     end
4604     self.is_writing = false
4605 end
4606
4607 -- push attributes for the new section
4608 table.insert(active_attributes, attributes)
4609 if self.is_writing and options.headerAttributes and #attributes > 0 then
4610     table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
4611 end
4612
4613 -- apply all active attributes at slice beginning
4614 if attributes["#" .. slice_begin_identfier] ~= nil
4615     and slice_begin_type == "^" then
4616     for header_level = 1, #active_attributes do
4617         if options.headerAttributes and #active_attributes[header_level] > 0 then
4618             table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
4619         end
4620     end
4621     self.is_writing = true
4622 end
4623
4624 if self.is_writing then
4625     table.sort(attributes)
4626     local key, value
4627     for i = 1, #attributes do
4628         if attributes[i]:sub(1, 1) == "#" then
4629             table.insert(buf, {"\\markdownRendererAttributeIdentifier{",
4630                               attributes[i]:sub(2), "}"})
4631         elseif attributes[i]:sub(1, 1) == "." then
4632             table.insert(buf, {"\\markdownRendererAttributeName{",

```

```

4633             attributes[i]:sub(2), "}")
4634         else
4635             key, value = attributes[i]:match("([~= ]+)%s*=%s*(.*)")
4636             table.insert(buf, {"\\markdownRendererAttributeKeyValue{",
4637                 key, "{", value, "}")
4638         end
4639     end
4640 end
4641
4642 local cmd
4643 level = level + options.shiftHeadings
4644 if level <= 1 then
4645     cmd = "\\markdownRendererHeadingOne"
4646 elseif level == 2 then
4647     cmd = "\\markdownRendererHeadingTwo"
4648 elseif level == 3 then
4649     cmd = "\\markdownRendererHeadingThree"
4650 elseif level == 4 then
4651     cmd = "\\markdownRendererHeadingFour"
4652 elseif level == 5 then
4653     cmd = "\\markdownRendererHeadingFive"
4654 elseif level >= 6 then
4655     cmd = "\\markdownRendererHeadingSix"
4656 else
4657     cmd = ""
4658 end
4659 if self.is_writing then
4660     table.insert(buf, {cmd, "{", s, "}"})
4661 end
4662
4663 return buf
4664 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.

```

4665 function self.get_state()
4666     return {
4667         is_writing=self.is_writing,
4668         active_attributes={table.unpack(self.active_attributes)},
4669     }
4670 end

```

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

```

4671 function self.set_state(s)
4672     local previous_state = self.get_state()
4673     for key, value in pairs(s) do
4674         self[key] = value

```



```

4675     end
4676     return previous_state
4677 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`.

```

4678 function self.defer_call(f)
4679     local previous_state = self.get_state()
4680     return function(...)
4681         local state = self.set_state(previous_state)
4682         local return_value = f(...)
4683         self.set_state(state)
4684         return return_value
4685     end
4686 end
4687
4688 return self
4689 end

```

### 3.1.4 Parsers

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

```

4690 local parsers = {}

```

#### 3.1.4.1 Basic Parsers

```

4691 parsers.percent = P("%")
4692 parsers.at = P("@")
4693 parsers.comma = P(",")
4694 parsers.asterisk = P("*")
4695 parsers.dash = P("-")
4696 parsers.plus = P("+")
4697 parsers.underscore = P("_")
4698 parsers.period = P(".")
4699 parsers.hash = P("#")
4700 parsers.ampersand = P("&")
4701 parsers.backtick = P("`")
4702 parsers.less = P("<")
4703 parsers.more = P(">")
4704 parsers.space = P(" ")
4705 parsers.squote = P("'")
4706 parsers.dquote = P('"')
4707 parsers.lparent = P("(")
4708 parsers.rparent = P(")")
4709 parsers.lbracket = P("[")

```

```

4710 parsers.rbracket          = P("]")
4711 parsers.lbrace             = P("{")
4712 parsers.rbrace             = P("}")
4713 parsers.circumflex         = P("^")
4714 parsers.slash              = P("/")
4715 parsers.equal              = P("=")
4716 parsers.colon              = P(":")
4717 parsers.semicolon          = P(";")
4718 parsers.exclamation        = P("!")
4719 parsers.pipe               = P("|")
4720 parsers.tilde              = P("~")
4721 parsers.backslash          = P("\\")
4722 parsers.tab                = P("\t")
4723 parsers.newline            = P("\n")
4724 parsers.tightblocksep      = P("\001")
4725
4726 parsers.digit              = R("09")
4727 parsers.hexdigit           = R("09","af","AF")
4728 parsers.letter             = R("AZ","az")
4729 parsers.alphanumeric       = R("AZ","az","09")
4730 parsers.keyword            = parsers.letter
4731                             * parsers.alphanumeric^0
4732 parsers.internal_punctuation = S(":;,.?")
4733
4734 parsers.doubleasterisks     = P("**")
4735 parsers.doubleunderscores   = P("__")
4736 parsers.doubletildes        = P("~")
4737 parsers.fourspace          = P("    ")
4738
4739 parsers.any                 = P(1)
4740 parsers.fail                = parsers.any - 1
4741
4742 parsers.escapable           = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")
4743 parsers.anyescaped          = parsers.backslash / " " * parsers.escapable
4744 + parsers.any
4745
4746 parsers.spacechar           = S("\t ")
4747 parsers.spacing             = S(" \n\r\t")
4748 parsers.nonspacechar        = parsers.any - parsers.spacing
4749 parsers.optionalspace       = parsers.spacechar^0
4750
4751 parsers.normalchar          = parsers.any - (V("SpecialChar")
4752   + parsers.spacing
4753   + parsers.tightblocksep)
4754 parsers.eof                 = -parsers.any
4755 parsers.nonindentSPACE      = parsers.space^-3 * - parsers.spacechar
4756 parsers.indent              = parsers.space^-3 * parsers.tab

```

```

4757                                     + parsers.fourspace / ""
4758 parsers.linechar                     = P(1 - parsers.newline)
4759
4760 parsers.blankline                     = parsers.optionalspace
4761                                     * parsers.newline / "\n"
4762 parsers.blanklines                     = parsers.blankline^0
4763 parsers.skipblanklines                 = (parsers.optionalspace * parsers.newline)^0
4764 parsers.indentedline                  = parsers.indent / ""
4765                                     * C(parsers.linechar^1 * parsers.newline^-
1)
4766 parsers.optionallyindentedline        = parsers.indent^-1 / ""
4767                                     * C(parsers.linechar^1 * parsers.newline^-
1)
4768 parsers.sp                             = parsers.spacing^0
4769 parsers.spnl                           = parsers.optionalspace
4770                                     * (parsers.newline * parsers.optionalspace)^-
1
4771 parsers.line                           = parsers.linechar^0 * parsers.newline
4772 parsers.nonemptyline                   = parsers.line - parsers.blankline

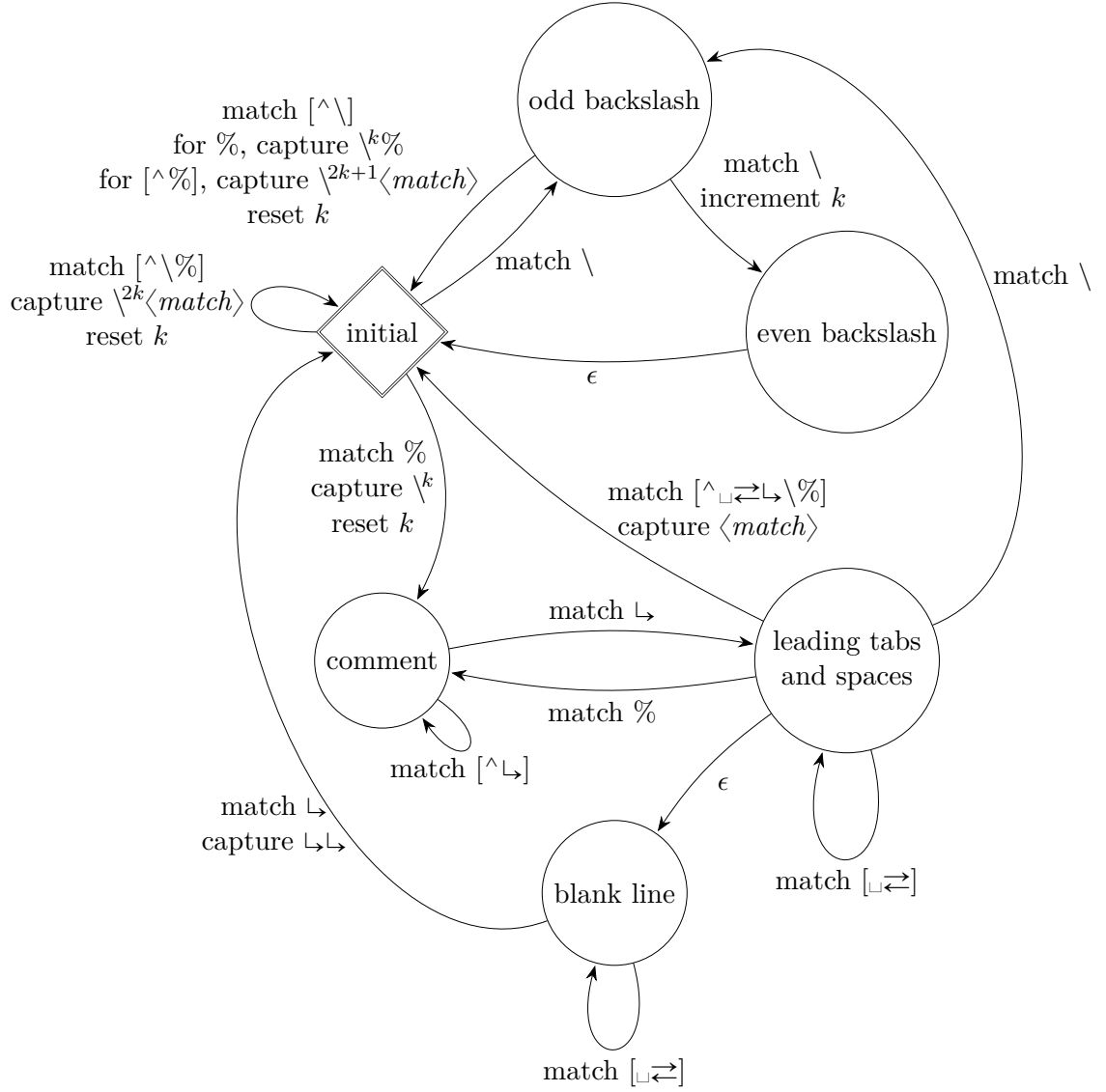
```

The `parsers.commented_line^1` parser recognizes the regular language of T<sub>E</sub>X comments, see an equivalent finite automaton in Figure 6.

```

4773 parsers.commented_line_letter        = parsers.linechar
4774                                     + parsers.newline
4775                                     - parsers.backslash
4776                                     - parsers.percent
4777 parsers.commented_line                = Cg(Cc(""), "backslashes")
4778                                     * ((#(parsers.commented_line_letter
4779                                     - parsers.newline)
4780                                     * Cb("backslashes")
4781                                     * Cs(parsers.commented_line_letter
4782                                     - parsers.newline)^1 -- initial
4783                                     * Cg(Cc(""), "backslashes"))
4784                                     + #(parsers.backslash * parsers.backslash)
4785                                     * Cg((parsers.backslash -- even backslash
4786                                     * parsers.backslash)^1, "backslashes")
4787                                     + (parsers.backslash
4788                                     * (#parsers.percent
4789                                     * Cb("backslashes")
4790                                     / function(backslashes)
4791                                     return string.rep("\\", #backslashes / 2)
4792                                     end
4793                                     * C(parsers.percent)
4794                                     + #parsers.commented_line_letter
4795                                     * Cb("backslashes")
4796                                     * Cc("\\")
4797                                     * C(parsers.commented_line_letter))

```



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

```

4798         * Cg(Cc(""), "backslashes"))))^0
4799     * (#parsers.percent
4800     * Cb("backslashes")
4801     / function(backslashes)
4802     return string.rep("\\", #backslashes / 2)
4803     end
4804     * ((parsers.percent -- comment
4805     * parsers.line
4806     * #parsers.blankline) -- blank line
4807     / "\\n"
4808     + parsers.percent -- comment
4809     * parsers.line
4810     * parsers.optionalspace) -- leading tabs and spaces
4811     + #(parsers.newline)
4812     * Cb("backslashes")
4813     * C(parsers.newline))
4814
4815 parsers.chunk = parsers.line * (parsers.optionallyindentedline
4816                             - parsers.blankline)^0
4817
4818 parsers.attribute_key_char = parsers.alphanumeric + S("_-")
4819 parsers.attribute_key = (parsers.attribute_key_char
4820                         - parsers.dash - parsers.digit)
4821                         * parsers.attribute_key_char^0
4822 parsers.attribute_value = ( (parsers.dquote / "\"")
4823                           * (parsers.anyescaped - parsers.dquote)^0
4824                           * (parsers.dquote / "\""))
4825                           + ( parsers.anyescaped - parsers.dquote - parsers.rbrace
4826                             - parsers.space)^0
4827
4828 -- block followed by 0 or more optionally
4829 -- indented blocks with first line indented.
4830 parsers.indented_blocks = function(bl)
4831     return Cs( bl
4832               * (parsers.blankline^1 * parsers.indent * -parsers.blankline * bl)^0
4833               * (parsers.blankline^1 + parsers.eof) )
4834 end

```

### 3.1.4.2 Parsers Used for Markdown Lists

```

4835 parsers.bulletchar = C(parsers.plus + parsers.asterisk + parsers.dash)
4836
4837 parsers.bullet = ( parsers.bulletchar * #parsers.spacing
4838                  * (parsers.tab + parsers.space^-3)
4839                  + parsers.space * parsers.bulletchar * #parsers.spacing
4840                  * (parsers.tab + parsers.space^-2)
4841                  + parsers.space * parsers.space * parsers.bulletchar

```

```

4842                                     * #parsers.spacing
4843                                     * (parsers.tab + parsers.space^-1)
4844             + parsers.space * parsers.space * parsers.space
4845                                     * parsers.bulletchar * #parsers.spacing
4846         )
4847
4848 local function tickbox(interior)
4849     return parsers.optionalspace * parsers.lbracket
4850         * interior * parsers.rbracket * parsers.spacechar^1
4851 end
4852
4853 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
4854 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
4855 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
4856

```

### 3.1.4.3 Parsers Used for Markdown Code Spans

```

4857 parsers.openticks    = Cg(parsers.backtick^1, "ticks")
4858
4859 local function captures_equal_length(_,i,a,b)
4860     return #a == #b and i
4861 end
4862
4863 parsers.closeticks   = parsers.space^-1
4864                       * Cmt(C(parsers.backtick^1)
4865                           * Cb("ticks"), captures_equal_length)
4866
4867 parsers.intickschar = (parsers.any - S("\n\r`"))
4868                       + (parsers.newline * -parsers.blankline)
4869                       + (parsers.space - parsers.closeticks)
4870                       + (parsers.backtick^1 - parsers.closeticks)
4871
4872 parsers.inticks      = parsers.openticks * parsers.space^-1
4873                       * C(parsers.intickschar^0) * parsers.closeticks

```

### 3.1.4.4 Parsers Used for Fenced Code Blocks

```

4874 local function captures_geq_length(_,i,a,b)
4875     return #a >= #b and i
4876 end
4877
4878 parsers.infostring   = (parsers.linechar - (parsers.backtick
4879       + parsers.space^1 * (parsers.newline + parsers.eof)))^0
4880
4881 local fenceindent
4882 parsers.fencehead    = function(char)
4883     return           C(parsers.nonindentospace) / function(s) fenceindent = #s end

```

```

4884             * Cg(char^3, "fencelength")
4885             * parsers.optionalspace * C(parsers.infostring)
4886             * parsers.optionalspace * (parsers.newline + parsers.eof)
4887     end
4888
4889     parsers.fencetail = function(char)
4890         return
4891             parsers.nonindentSPACE
4892             * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
4893             * parsers.optionalspace * (parsers.newline + parsers.eof)
4894             + parsers.eof
4895     end
4896
4897     parsers.fencedline = function(char)
4898         return
4899             C(parsers.line - parsers.fencetail(char))
4900             / function(s)
4901                 local i = 1
4902                 local remaining = fenceindent
4903                 while true do
4904                     local c = s:sub(i, i)
4905                     if c == " " and remaining > 0 then
4906                         remaining = remaining - 1
4907                         i = i + 1
4908                     elseif c == "\t" and remaining > 3 then
4909                         remaining = remaining - 4
4910                         i = i + 1
4911                     else
4912                         break
4913                     end
4914                 end
4915                 return s:sub(i)
4916             end
4917     end

```

### 3.1.4.5 Parsers Used for Markdown Tags and Links

```

4916 parsers.leader      = parsers.space^-3
4917
4918 -- content in balanced brackets, parentheses, or quotes:
4919 parsers.bracketed    = P{ parsers.lbracket
4920             * (( parsers.backslash / "\"" * parsers.rbracket
4921                 + parsers.any - (parsers.lbracket
4922                                     + parsers.rbracket
4923                                     + parsers.blankline^2)
4924                 ) + V(1))^0
4925             * parsers.rbracket }
4926
4927 parsers.inparens     = P{ parsers.lparent

```

```

4928             * ((parsers.anyescaped - (parsers.lparent
4929                                     + parsers.rparent
4930                                     + parsers.blankline^2)
4931             ) + V(1))^0
4932             * parsers.rparent }
4933
4934 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
4935             * ((parsers.anyescaped - (parsers.squote
4936                                     + parsers.blankline^2)
4937             ) + V(1))^0
4938             * parsers.squote }
4939
4940 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
4941             * ((parsers.anyescaped - (parsers.dquote
4942                                     + parsers.blankline^2)
4943             ) + V(1))^0
4944             * parsers.dquote }
4945
4946 -- bracketed tag for markdown links, allowing nested brackets:
4947 parsers.tag = parsers.lbracket
4948             * Cs((parsers.alphanumeric^1
4949                 + parsers.bracketed
4950                 + parsers.inticks
4951                 + ( parsers.backslash / "\"" * parsers.rbracket
4952                 + parsers.any
4953                 - (parsers.rbracket + parsers.blankline^2)))^0)
4954             * parsers.rbracket
4955
4956 -- url for markdown links, allowing nested brackets:
4957 parsers.url = parsers.less * Cs((parsers.anyescaped
4958                               - parsers.more)^0)
4959                               * parsers.more
4960             + Cs((parsers.inparens + (parsers.anyescaped
4961                               - parsers.spacing
4962                               - parsers.rparent))^1)
4963
4964 -- quoted text, possibly with nested quotes:
4965 parsers.title_s = parsers.squote * Cs(((parsers.anyescaped-parsers.squote)
4966                               + parsers.squoted)^0)
4967                               * parsers.squote
4968
4969 parsers.title_d = parsers.dquote * Cs(((parsers.anyescaped-parsers.dquote)
4970                               + parsers.dquoted)^0)
4971                               * parsers.dquote
4972
4973 parsers.title_p = parsers.lparent
4974             * Cs((parsers.inparens + (parsers.anyescaped-parsers.rparent))^0)

```



```

4975             * parsers.rparent
4976
4977 parsers.title      = parsers.title_d + parsers.title_s + parsers.title_p
4978
4979 parsers.optionaltitle
4980             = parsers.spnl * parsers.title * parsers.spacechar~0
4981             + Cc("")

```

### 3.1.4.6 Parsers Used for HTML

```

4982 -- case-insensitive match (we assume s is lowercase). must be single byte encoding
4983 parsers.keyword_exact = function(s)
4984     local parser = P(0)
4985     for i=1,#s do
4986         local c = s:sub(i,i)
4987         local m = c .. upper(c)
4988         parser = parser * S(m)
4989     end
4990     return parser
4991 end
4992
4993 parsers.block_keyword =
4994     parsers.keyword_exact("address") + parsers.keyword_exact("blockquote") +
4995     parsers.keyword_exact("center") + parsers.keyword_exact("del") +
4996     parsers.keyword_exact("div") + parsers.keyword_exact("div") +
4997     parsers.keyword_exact("p") + parsers.keyword_exact("pre") +
4998     parsers.keyword_exact("li") + parsers.keyword_exact("ol") +
4999     parsers.keyword_exact("ul") + parsers.keyword_exact("dl") +
5000     parsers.keyword_exact("dd") + parsers.keyword_exact("form") +
5001     parsers.keyword_exact("fieldset") + parsers.keyword_exact("isindex") +
5002     parsers.keyword_exact("ins") + parsers.keyword_exact("menu") +
5003     parsers.keyword_exact("noframes") + parsers.keyword_exact("frameset") +
5004     parsers.keyword_exact("h1") + parsers.keyword_exact("h2") +
5005     parsers.keyword_exact("h3") + parsers.keyword_exact("h4") +
5006     parsers.keyword_exact("h5") + parsers.keyword_exact("h6") +
5007     parsers.keyword_exact("hr") + parsers.keyword_exact("script") +
5008     parsers.keyword_exact("noscript") + parsers.keyword_exact("table") +
5009     parsers.keyword_exact("tbody") + parsers.keyword_exact("tfoot") +
5010     parsers.keyword_exact("thead") + parsers.keyword_exact("th") +
5011     parsers.keyword_exact("td") + parsers.keyword_exact("tr")
5012
5013 -- There is no reason to support bad html, so we expect quoted attributes
5014 parsers.htmlattributevalue
5015     = parsers.squote * (parsers.any - (parsers.blankline
5016                                     + parsers.squote))~0
5017                                     * parsers.squote
5018     + parsers.dquote * (parsers.any - (parsers.blankline

```

```

5019                                     + parsers.dquote))^0
5020                                     * parsers.dquote
5021
5022 parsers.htmlattribute      = parsers.spacing^1
5023                             * (parsers.alphanumeric + S("_-"))^1
5024                             * parsers.sp * parsers.equal * parsers.sp
5025                             * parsers.htmlattributevalue
5026
5027 parsers.htmlcomment       = P("<!--")
5028                             * parsers.optionalspace
5029                             * Cs((parsers.any - parsers.optionalspace * P("-->"))^0)
5030                             * parsers.optionalspace
5031                             * P("-->")
5032
5033 parsers.htmlinstruction    = P("<?") * (parsers.any - P("?>"))^0 * P("?>")
5034
5035 parsers.openelt_any = parsers.less * parsers.keyword * parsers.htmlattribute^0
5036                     * parsers.sp * parsers.more
5037
5038 parsers.openelt_exact = function(s)
5039   return parsers.less * parsers.sp * parsers.keyword_exact(s)
5040         * parsers.htmlattribute^0 * parsers.sp * parsers.more
5041 end
5042
5043 parsers.openelt_block = parsers.sp * parsers.block_keyword
5044                       * parsers.htmlattribute^0 * parsers.sp * parsers.more
5045
5046 parsers.closeelt_any = parsers.less * parsers.sp * parsers.slash
5047                     * parsers.keyword * parsers.sp * parsers.more
5048
5049 parsers.closeelt_exact = function(s)
5050   return parsers.less * parsers.sp * parsers.slash * parsers.keyword_exact(s)
5051         * parsers.sp * parsers.more
5052 end
5053
5054 parsers.emptyelt_any = parsers.less * parsers.sp * parsers.keyword
5055                     * parsers.htmlattribute^0 * parsers.sp * parsers.slash
5056                     * parsers.more
5057
5058 parsers.emptyelt_block = parsers.less * parsers.sp * parsers.block_keyword
5059                       * parsers.htmlattribute^0 * parsers.sp * parsers.slash
5060                       * parsers.more
5061
5062 parsers.displaytext = (parsers.any - parsers.less)^1
5063
5064 -- return content between two matched HTML tags
5065 parsers.in_matched = function(s)

```

```

5066 return { parsers.openelt_exact(s)
5067          * (V(1) + parsers.displaytext
5068            + (parsers.less - parsers.closeelt_exact(s)))^0
5069          * parsers.closeelt_exact(s) }
5070 end
5071
5072 local function parse_matched_tags(s,pos)
5073   local t = string.lower(lpeg.match(C(parsers.keyword),s,pos))
5074   return lpeg.match(parsers.in_matched(t),s,pos-1)
5075 end
5076
5077 parsers.in_matched_block_tags = parsers.less
5078                               * Cmt(#parsers.openelt_block, parse_matched_tags)
5079

```

#### 3.1.4.7 Parsers Used for HTML Entities

```

5080 parsers.hexentity = parsers.ampersand * parsers.hash * S("Xx")
5081                  * C(parsers.hexdigit^1) * parsers.semicolon
5082 parsers.decentity = parsers.ampersand * parsers.hash
5083                  * C(parsers.digit^1) * parsers.semicolon
5084 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
5085                  * parsers.semicolon

```

#### 3.1.4.8 Helpers for References

```

5086 -- parse a reference definition: [foo]: /bar "title"
5087 parsers.define_reference_parser = parsers.leader * parsers.tag * parsers.colon
5088                               * parsers.spacechar^0 * parsers.url
5089                               * parsers.optionaltitle * parsers.blankline^1

```

#### 3.1.4.9 Inline Elements

```

5090 parsers.Inline      = V("Inline")
5091 parsers.IndentedInline = V("IndentedInline")
5092
5093 -- parse many p between starter and ender
5094 parsers.between = function(p, starter, ender)
5095   local ender2 = B(parsers.nonspacechar) * ender
5096   return (starter * #parsers.nonspacechar * Ct(p * (p - ender2)^0) * ender2)
5097 end
5098
5099 parsers.urlchar      = parsers.anyescaped - parsers.newline - parsers.more

```

#### 3.1.4.10 Block Elements

```

5100 parsers.lineof = function(c)
5101   return (parsers.leader * (P(c) * parsers.optionalspace)^3
5102         * (parsers.newline * parsers.blankline^1

```

```

5103             + parsers.newline^-1 * parsers.eof))
5104 end

```

### 3.1.4.11 Lists

#### 3.1.4.12 Headings

```

5105 parsers.heading_attribute = (parsers.dash * Cc(".unnumbered"))
5106                             + C((parsers.hash + parsers.period)
5107                                * parsers.attribute_key)
5108                             + Cs( parsers.attribute_key
5109                                * parsers.optionalspace * parsers.equal * parsers.optionalspace
5110                                * parsers.attribute_value)
5111 parsers.HeadingAttributes = parsers.lbrace
5112                             * parsers.optionalspace
5113                             * parsers.heading_attribute
5114                             * (parsers.spacechar^1
5115                                * parsers.heading_attribute)^0
5116                             * parsers.optionalspace
5117                             * parsers.rbrace
5118
5119 -- parse Atx heading start and return level
5120 parsers.HeadingStart = #parsers.hash * C(parsers.hash^-6)
5121                       * -parsers.hash / length
5122
5123 -- parse setext header ending and return level
5124 parsers.HeadingLevel = parsers.equal^1 * Cc(1) + parsers.dash^1 * Cc(2)
5125
5126 local function strip_atx_end(s)
5127   return s:gsub("#%s*\n$", "")
5128 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>`.

```
5129 M.reader = {}
5130 function M.reader.new(writer, options, extensions)
5131     local self = {}
```

Make the `writer` and `options` parameters available as `reader->writer` and `reader->options`, respectively, so that they are accessible from extensions.

```
5132     self.writer = writer
5133     self.options = options
```

Create a `reader->parsers` hash table that stores PEG patterns that depend on the received `options`. Make `reader->parsers` inherit from the global `parsers` table.

```
5134     self.parsers = {}
5135     (function(parsers)
5136         setmetatable(self.parsers, {
5137             __index = function (_, key)
5138                 return parsers[key]
5139             end
5140         })
5141     end)(parsers)
```

Make `reader->parsers` available as a local `parsers` variable that will shadow the global `parsers` table and will make `reader->parsers` easier to type in the rest of the reader code.

```
5142     local parsers = self.parsers
```

**3.1.5.1 Top-Level Helper Functions** Define `reader->normalize_tag` as a function that normalizes a markdown reference tag by lowercasing it, and by collapsing any adjacent whitespace characters.

```
5143     function self.normalize_tag(tag)
5144         return string.lower(
5145             gsub(util.rope_to_string(tag), "[\n\r\t]+", " "))
5146     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.

```
5147     local function iterlines(s, f)
5148         local rope = lpeg.match(Ct((parsers.line / f)^1), s)
5149         return util.rope_to_string(rope)
5150     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.

```
5151     if options.preserveTabs then
```

```

5152     self.expandtabs = function(s) return s end
5153 else
5154     self.expandtabs = function(s)
5155         if s:find("\t") then
5156             return iterlines(s, util.expand_tabs_in_line)
5157         else
5158             return s
5159         end
5160     end
5161 end

```

**3.1.5.2 High-Level Parser Functions** Create a `reader->parser_functions` hash table that stores high-level parser functions. Define `reader->create_parser` as a function that will create a high-level parser function `reader->parser_functions.name`, that matches input using grammar `grammar`. If `toplevel` is true, the input is expected to come straight from the user, not from a recursive call, and will be preprocessed.

```

5162     self.parser_functions = {}
5163     self.create_parser = function(name, grammar, toplevel)
5164         self.parser_functions[name] = function(str)

```

If the parser function 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.

```

5165         if toplevel and options.stripIndent then
5166             local min_prefix_length, min_prefix = nil, ''
5167             str = iterlines(str, function(line)
5168                 if lpeg.match(parsers.nonemptyline, line) == nil then
5169                     return line
5170                 end
5171                 line = util.expand_tabs_in_line(line)
5172                 local prefix = lpeg.match(C(parsers.optionalspace), line)
5173                 local prefix_length = #prefix
5174                 local is_shorter = min_prefix_length == nil
5175                 is_shorter = is_shorter or prefix_length < min_prefix_length
5176                 if is_shorter then
5177                     min_prefix_length, min_prefix = prefix_length, prefix
5178                 end
5179                 return line
5180             end)
5181             str = str:gsub('^' .. min_prefix, '')
5182         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.

```

5183     if toplevel and (options.texComments or options.hybrid) then
5184         str = lpeg.match(Ct(parsers.commented_line^1), str)
5185         str = util.rope_to_string(str)
5186     end
5187     local res = lpeg.match(grammar(), str)
5188     if res == nil then
5189         error(format("%s failed on:\n%s", name, str:sub(1,20)))
5190     else
5191         return res
5192     end
5193 end
5194 end
5195
5196 self.create_parser("parse_blocks",
5197     function()
5198         return parsers.blocks
5199     end, true)
5200
5201 self.create_parser("parse_blocks_nested",
5202     function()
5203         return parsers.blocks_nested
5204     end, false)
5205
5206 self.create_parser("parse_inlines",
5207     function()
5208         return parsers.inlines
5209     end, false)
5210
5211 self.create_parser("parse_inlines_no_link",
5212     function()
5213         return parsers.inlines_no_link
5214     end, false)
5215
5216 self.create_parser("parse_inlines_no_inline_note",
5217     function()
5218         return parsers.inlines_no_inline_note
5219     end, false)
5220
5221 self.create_parser("parse_inlines_no_html",
5222     function()
5223         return parsers.inlines_no_html
5224     end, false)
5225
5226 self.create_parser("parse_inlines_nbsp",

```

```

5227         function()
5228             return parsers.inlines_nbsp
5229         end, false)

```

### 3.1.5.3 Parsers Used for Markdown Lists (local)

```

5230     if options.hashEnumerators then
5231         parsers.dig = parsers.digit + parsers.hash
5232     else
5233         parsers.dig = parsers.digit
5234     end
5235
5236     parsers.enumerator = C(parsers.dig^3 * parsers.period) * #parsers.spacing
5237                       + C(parsers.dig^2 * parsers.period) * #parsers.spacing
5238                       * (parsers.tab + parsers.space^1)
5239                       + C(parsers.dig * parsers.period) * #parsers.spacing
5240                       * (parsers.tab + parsers.space^2)
5241                       + parsers.space * C(parsers.dig^2 * parsers.period)
5242                       * #parsers.spacing
5243                       + parsers.space * C(parsers.dig * parsers.period)
5244                       * #parsers.spacing
5245                       * (parsers.tab + parsers.space^1)
5246                       + parsers.space * parsers.space * C(parsers.dig^1
5247                       * parsers.period) * #parsers.spacing

```

### 3.1.5.4 Parsers Used for Blockquotes (local)

```

5248     -- strip off leading > and indents, and run through blocks
5249     parsers.blockquote_body = ((parsers.leader * parsers.more * parsers.space^~
5250     1)/""
5251                               * parsers.linechar^0 * parsers.newline)^1
5252                               * (-(parsers.leader * parsers.more
5253                               + parsers.blankline) * parsers.linechar^1
5254                               * parsers.newline)^0
5255
5256     if not options.breakableBlockquotes then
5257         parsers.blockquote_body = parsers.blockquote_body
5258         * (parsers.blankline^0 / "")
5259     end

```

### 3.1.5.5 Parsers Used for Footnotes (local)

### 3.1.5.6 Helpers for Links and References (local)

```

5259     -- List of references defined in the document
5260     local references
5261
5262     -- add a reference to the list

```



```

5263 local function register_link(tag,url,title)
5264     references[self.normalize_tag(tag)] = { url = url, title = title }
5265     return ""
5266 end
5267
5268 -- lookup link reference and return either
5269 -- the link or nil and fallback text.
5270 local function lookup_reference(label,sps,tag)
5271     local tagpart
5272     if not tag then
5273         tag = label
5274         tagpart = ""
5275     elseif tag == "" then
5276         tag = label
5277         tagpart = "[]"
5278     else
5279         tagpart = {"[",
5280             self.parser_functions.parse_inlines(tag),
5281             "]" }
5282     end
5283     if sps then
5284         tagpart = {sps, tagpart}
5285     end
5286     local r = references[self.normalize_tag(tag)]
5287     if r then
5288         return r
5289     else
5290         return nil, {"[",
5291             self.parser_functions.parse_inlines(label),
5292             "]", tagpart}
5293     end
5294 end
5295
5296 -- lookup link reference and return a link, if the reference is found,
5297 -- or a bracketed label otherwise.
5298 local function indirect_link(label,sps,tag)
5299     return writer.defer_call(function()
5300         local r,fallback = lookup_reference(label,sps,tag)
5301         if r then
5302             return writer.link(
5303                 self.parser_functions.parse_inlines_no_link(label),
5304                 r.url, r.title)
5305         else
5306             return fallback
5307         end
5308     end)
5309 end

```

```

5310
5311 -- lookup image reference and return an image, if the reference is found,
5312 -- or a bracketed label otherwise.
5313 local function indirect_image(label,sps,tag)
5314     return writer.defer_call(function()
5315         local r,fallback = lookup_reference(label,sps,tag)
5316         if r then
5317             return writer.image(writer.string(label), r.url, r.title)
5318         else
5319             return {"!", fallback}
5320         end
5321     end)
5322 end

```

### 3.1.5.7 Inline Elements (local)

```

5323 parsers.Str      = (parsers.normalchar * (parsers.normalchar + parsers.at)^0)
5324                  / writer.string
5325
5326 parsers.Symbol   = (V("SpecialChar") - parsers.tightblocksep)
5327                  / writer.string
5328
5329 parsers.Ellipsis = P("...") / writer.ellipsis
5330
5331 parsers.Smart    = parsers.Ellipsis
5332
5333 parsers.Code     = parsers.inticks / writer.code
5334
5335 if options.blankBeforeBlockquote then
5336     parsers.bqstart = parsers.fail
5337 else
5338     parsers.bqstart = parsers.more
5339 end
5340
5341 if options.blankBeforeHeading then
5342     parsers.headerstart = parsers.fail
5343 else
5344     parsers.headerstart = parsers.hash
5345                         + (parsers.line * (parsers.equal^1 + parsers.dash^1)
5346                           * parsers.optionalspace * parsers.newline)
5347 end
5348
5349 parsers.EndlineExceptions
5350     = parsers.blankline -- paragraph break
5351     + parsers.tightblocksep -- nested list
5352     + parsers.eof          -- end of document
5353     + parsers.bqstart

```

```

5354         + parsers.headerstart
5355
5356 parsers.Endline = parsers.newline
5357                 * -V("EndlineExceptions")
5358                 * parsers.spacechar^0
5359                 / (options.hardLineBreaks and writer.linebreak
5360                   or writer.space)
5361
5362 parsers.OptionalIndent
5363                 = parsers.spacechar^1 / writer.space
5364
5365 parsers.Space = parsers.spacechar^2 * parsers.Endline / writer.linebreak
5366               + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
5367               + parsers.spacechar^1 * parsers.Endline
5368                 * parsers.optionalspace
5369                 / (options.hardLineBreaks
5370                   and writer.linebreak
5371                   or writer.space)
5372               + parsers.spacechar^1 * parsers.optionalspace
5373                 / writer.space
5374
5375 parsers.NonbreakingEndline
5376                 = parsers.newline
5377                 * -V("EndlineExceptions")
5378                 * parsers.spacechar^0
5379                 / (options.hardLineBreaks and writer.linebreak
5380                   or writer.nbsp)
5381
5382 parsers.NonbreakingSpace
5383                 = parsers.spacechar^2 * parsers.Endline / writer.linebreak
5384               + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
5385               + parsers.spacechar^1 * parsers.Endline
5386                 * parsers.optionalspace
5387                 / (options.hardLineBreaks
5388                   and writer.linebreak
5389                   or writer.nbsp)
5390               + parsers.spacechar^1 * parsers.optionalspace
5391                 / writer.nbsp
5392
5393 if options.underscores then
5394     parsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,
5395                                       parsers.doubleasterisks)
5396                     + parsers.between(parsers.Inline, parsers.doubleunderscores,
5397                                       parsers.doubleunderscores)
5398                     ) / writer.strong
5399
5400 parsers.Emph = ( parsers.between(parsers.Inline, parsers.asterisk,

```

```

5401                 parsers.asterisk)
5402         + parsers.between(parsers.Inline, parsers.underscore,
5403                 parsers.underscore)
5404         ) / writer.emphasis
5405     else
5406         parsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,
5407                 parsers.doubleasterisks)
5408         ) / writer.strong
5409
5410         parsers.Emph   = ( parsers.between(parsers.Inline, parsers.asterisk,
5411                 parsers.asterisk)
5412         ) / writer.emphasis
5413     end
5414
5415     parsers.AutoLinkUrl    = parsers.less
5416         * C(parsers.alphanumeric^1 * P("://") * parsers.urlchar^1)
5417         * parsers.more
5418         / function(url)
5419             return writer.link(writer.escape(url), url)
5420         end
5421
5422     parsers.AutoLinkEmail = parsers.less
5423         * C((parsers.alphanumeric + S("-._+"))^1
5424         * P("@") * parsers.urlchar^1)
5425         * parsers.more
5426         / function(email)
5427             return writer.link(writer.escape(email),
5428                 "mailto: "..email)
5429         end
5430
5431     parsers.AutoLinkRelativeReference
5432         = parsers.less
5433         * C(parsers.urlchar^1)
5434         * parsers.more
5435         / function(url)
5436             return writer.link(writer.escape(url), url)
5437         end
5438
5439     parsers.DirectLink    = (parsers.tag / self.parser_functions.parse_inlines_no_link)
5440         * parsers.spnl
5441         * parsers.lparent
5442         * (parsers.url + C("")) -- link can be empty [foo]()
5443         * parsers.optionaltitle
5444         * parsers.rparent
5445         / writer.link
5446

```

```

5447 parsers.IndirectLink = parsers.tag * (C(parsers.spnl) * parsers.tag)^-
5448                               / indirect_link
5449
5450 -- parse a link or image (direct or indirect)
5451 parsers.Link           = parsers.DirectLink + parsers.IndirectLink
5452
5453 parsers.DirectImage    = parsers.exclamation
5454                       * (parsers.tag / self.parser_functions.parse_inlines)
5455                       * parsers.spnl
5456                       * parsers.lparent
5457                       * (parsers.url + Cc("")) -- link can be empty [foo]()
5458                       * parsers.optionaltitle
5459                       * parsers.rparent
5460                       / writer.image
5461
5462 parsers.IndirectImage  = parsers.exclamation * parsers.tag
5463                       * (C(parsers.spnl) * parsers.tag)^-1 / indirect_image
5464
5465 parsers.Image          = parsers.DirectImage + parsers.IndirectImage
5466
5467 -- avoid parsing long strings of * or _ as emph/strong
5468 parsers.UlOrStarLine   = parsers.asterisk^4 + parsers.underscore^4
5469                       / writer.string
5470
5471 parsers.EscapedChar    = parsers.backslash * C(parsers.escapable) / writer.string
5472
5473 parsers.InlineHtml     = parsers.emptyelt_any / writer.inline_html_tag
5474                       + (parsers.htmlcomment / self.parser_functions.parse_inlines_no
5475                       / writer.inline_html_comment
5476                       + parsers.htmlinstruction
5477                       + parsers.openelt_any / writer.inline_html_tag
5478                       + parsers.closeelt_any / writer.inline_html_tag
5479
5480 parsers.HtmlEntity     = parsers.hexentity / entities.hex_entity / writer.string
5481                       + parsers.decentity / entities.dec_entity / writer.string
5482                       + parsers.tagentity / entities.char_entity / writer.string

```

### 3.1.5.8 Block Elements (local)

```

5483 parsers.DisplayHtml    = (parsers.htmlcomment / self.parser_functions.parse_blocks_nest
5484                       / writer.block_html_comment
5485                       + parsers.emptyelt_block / writer.block_html_element
5486                       + parsers.openelt_exact("hr") / writer.block_html_element
5487                       + parsers.in_matched_block_tags / writer.block_html_element
5488                       + parsers.htmlinstruction
5489

```

```

5490 parsers.Verbatim      = Cs( (parsers.blanklines
5491                               * ((parsers.indentedline - parsers.blankline))^1)^1
5492                               ) / self.expandtabs / writer.verbatim
5493
5494 parsers.Blockquote     = Cs(parsers.blockquote_body^1)
5495                           / self.parser_functions.parse_blocks_nested
5496                           / writer.blockquote
5497
5498 parsers.HorizontalRule = ( parsers.lineof(parsers.asterisk)
5499                           + parsers.lineof(parsers.dash)
5500                           + parsers.lineof(parsers.underscore)
5501                           ) / writer.hrule
5502
5503 parsers.Reference      = parsers.define_reference_parser / register_link
5504
5505 parsers.Paragraph      = parsers.nonindentspace * Ct(parsers.Inline^1)
5506                       * ( parsers.newline
5507                           * ( parsers.blankline^1
5508                               + #parsers.hash
5509                               + #(parsers.leader * parsers.more * parsers.space^-
5510                                   1)
5511                               + parsers.eof
5512                               )
5513                           + parsers.eof )
5514                       / writer.paragraph
5515
5516 parsers.Plain           = parsers.nonindentspace * Ct(parsers.Inline^1)
5517                       / writer.plain

```

### 3.1.5.9 Lists (local)

```

5517 parsers.starter = parsers.bullet + parsers.enumerator
5518
5519 if options.taskLists then
5520   parsers.tickbox = ( parsers.ticked_box
5521                       + parsers.halfchecked_box
5522                       + parsers.unticked_box
5523                       ) / writer.tickbox
5524 else
5525   parsers.tickbox = parsers.fail
5526 end
5527
5528 -- we use \001 as a separator between a tight list item and a
5529 -- nested list under it.
5530 parsers.NestedList      = Cs((parsers.optionallyindentedline
5531                               - parsers.starter)^1)
5532                           / function(a) return "\001"..a end

```

```

5533
5534 parsers.ListBlockLine      = parsers.optionallyindentedline
5535                             - parsers.blankline - (parsers.indent~-1
5536   * parsers.starter)
5537
5538 parsers.ListBlock           = parsers.line * parsers.ListBlockLine^0
5539
5540 parsers.ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
5541                             * parsers.ListBlock
5542
5543 parsers.TightListItem = function(starter)
5544     return -parsers.HorizontalRule
5545         * (Cs(starter / "" * parsers.tickbox~-1 * parsers.ListBlock * parsers.Nested
5546 1)
5547         / self.parser_functions.parse_blocks_nested)
5548         * -(parsers.blanklines * parsers.indent)
5549 end
5550
5551 parsers.LooseListItem = function(starter)
5552     return -parsers.HorizontalRule
5553         * Cs( starter / "" * parsers.tickbox~-1 * parsers.ListBlock * Cc("\n")
5554         * (parsers.NestedList + parsers.ListContinuationBlock^0)
5555         * (parsers.blanklines / "\n\n")
5556         ) / self.parser_functions.parse_blocks_nested
5557 end
5558
5559 parsers.BulletList = ( Ct(parsers.TightListItem(parsers.bullet)^1) * Cc(true)
5560                       * parsers.skipblanklines * -parsers.bullet
5561                       + Ct(parsers.LooseListItem(parsers.bullet)^1) * Cc(false)
5562                       * parsers.skipblanklines )
5563                       / writer.bulletlist
5564
5565 local function ordered_list(items,tight,startnum)
5566     if options.startNumber then
5567         startnum = tonumber(startnum) or 1 -- fallback for '#'
5568         if startnum ~= nil then
5569             startnum = math.floor(startnum)
5570         end
5571     else
5572         startnum = nil
5573     end
5574     return writer.orderedlist(items,tight,startnum)
5575 end
5576
5577 parsers.OrderedList = Cg(parsers.enumerator, "listtype") *
5578     ( Ct(parsers.TightListItem(Cb("listtype")))
5579       * parsers.TightListItem(parsers.enumerator)^0)

```

```

5579         * Cc(true) * parsers.skipblanklines * ~parsers.enumerator
5580     + Ct(parsers.LooseListItem(Cb("listtype")))
5581         * parsers.LooseListItem(parsers.enumerator)^0)
5582     * Cc(false) * parsers.skipblanklines
5583     ) * Cb("listtype") / ordered_list

```

### 3.1.5.10 Blank (local)

```

5584     parsers.Blank = parsers.blankline / ""
5585     + parsers.Reference
5586     + (parsers.tightblocksep / "\n")

```

### 3.1.5.11 Headings (local)

```

5587     -- parse atx header
5588     parsers.AtxHeading = Cg(parsers.HeadingStart,"level")
5589         * parsers.optionalspace
5590         * (C(parsers.line)
5591           / strip_atx_end
5592           / self.parser_functions.parse_inlines)
5593         * Cb("level")
5594         / writer.heading
5595
5596     parsers.SettextHeading = #(parsers.line * S("--"))
5597         * Ct(parsers.linechar^1
5598           / self.parser_functions.parse_inlines)
5599         * parsers.newline
5600         * parsers.HeadingLevel
5601         * parsers.optionalspace
5602         * parsers.newline
5603         / writer.heading
5604
5605     parsers.Heading = parsers.AtxHeading + parsers.SettextHeading

```

**3.1.5.12 Syntax Specification** Create a [reader->syntax](#) hash table that stores the PEG grammar.

```

5606     self.syntax =
5607         { "Blocks",
5608
5609           Blocks = ( V("ExpectedJekyllData")
5610             * (V("Blank")^0 / writer.interblocksep)
5611             )^~1
5612             * V("Blank")^0
5613             * V("Block")^~1
5614             * (V("Blank")^0 / writer.interblocksep
5615               * V("Block"))^0
5616             * V("Blank")^0 * parsers.eof,

```



```

5617
5618     Blank                = parsers.Blank,
5619
5620     UnexpectedJekyllData = parsers.fail,
5621     ExpectedJekyllData  = parsers.fail,
5622
5623     Block                = V("ContentBlock")
5624                          + V("UnexpectedJekyllData")
5625                          + V("Blockquote")
5626                          + V("PipeTable")
5627                          + V("Verbatim")
5628                          + V("FencedCode")
5629                          + V("HorizontalRule")
5630                          + V("BulletList")
5631                          + V("OrderedList")
5632                          + V("Heading")
5633                          + V("DefinitionList")
5634                          + V("DisplayHtml")
5635                          + V("Paragraph")
5636                          + V("Plain"),
5637
5638     ContentBlock          = parsers.fail,
5639     Blockquote            = parsers.Blockquote,
5640     Verbatim              = parsers.Verbatim,
5641     FencedCode            = parsers.fail,
5642     HorizontalRule        = parsers.HorizontalRule,
5643     BulletList            = parsers.BulletList,
5644     OrderedList          = parsers.OrderedList,
5645     Heading               = parsers.Heading,
5646     DefinitionList        = parsers.fail,
5647     DisplayHtml           = parsers.DisplayHtml,
5648     Paragraph             = parsers.Paragraph,
5649     PipeTable             = parsers.fail,
5650     Plain                 = parsers.Plain,
5651     EndlineExceptions     = parsers.EndlineExceptions,
5652
5653     Inline                = V("Str")
5654                          + V("Space")
5655                          + V("Endline")
5656                          + V("U1OrStarLine")
5657                          + V("Strong")
5658                          + V("Emph")
5659                          + V("StrikeThrough")
5660                          + V("Superscript")
5661                          + V("Subscript")
5662                          + V("InlineNote")
5663                          + V("NoteRef")

```

5664		+ V("Citations")
5665		+ V("Link")
5666		+ V("Image")
5667		+ V("Code")
5668		+ V("AutoLinkUrl")
5669		+ V("AutoLinkEmail")
5670		+ V("AutoLinkRelativeReference")
5671		+ V("InlineHtml")
5672		+ V("HtmlEntity")
5673		+ V("EscapedChar")
5674		+ V("Smart")
5675		+ V("Symbol"),
5676		
5677	IndentedInline	= V("Str")
5678		+ V("OptionalIndent")
5679		+ V("Endline")
5680		+ V("U1OrStarLine")
5681		+ V("Strong")
5682		+ V("Emph")
5683		+ V("StrikeThrough")
5684		+ V("Superscript")
5685		+ V("Subscript")
5686		+ V("InlineNote")
5687		+ V("NoteRef")
5688		+ V("Citations")
5689		+ V("Link")
5690		+ V("Image")
5691		+ V("Code")
5692		+ V("AutoLinkUrl")
5693		+ V("AutoLinkEmail")
5694		+ V("AutoLinkRelativeReference")
5695		+ V("InlineHtml")
5696		+ V("HtmlEntity")
5697		+ V("EscapedChar")
5698		+ V("Smart")
5699		+ V("Symbol"),
5700		
5701	Str	= parsers.Str,
5702	Space	= parsers.Space,
5703	OptionalIndent	= parsers.OptionalIndent,
5704	Endline	= parsers.Endline,
5705	U1OrStarLine	= parsers.U1OrStarLine,
5706	Strong	= parsers.Strong,
5707	Emph	= parsers.Emph,
5708	StrikeThrough	= parsers.fail,
5709	Superscript	= parsers.fail,
5710	Subscript	= parsers.fail,

```

5711     InlineNote           = parsers.fail,
5712     NoteRef              = parsers.fail,
5713     Citations            = parsers.fail,
5714     Link                  = parsers.Link,
5715     Image                 = parsers.Image,
5716     Code                  = parsers.Code,
5717     AutoLinkUrl           = parsers.AutoLinkUrl,
5718     AutoLinkEmail         = parsers.AutoLinkEmail,
5719     AutoLinkRelativeReference
5720                             = parsers.AutoLinkRelativeReference,
5721     InlineHtml             = parsers.InlineHtml,
5722     HtmlEntity            = parsers.HtmlEntity,
5723     EscapedChar           = parsers.EscapedChar,
5724     Smart                  = parsers.Smart,
5725     Symbol                = parsers.Symbol,
5726     SpecialChar           = parsers.fail,
5727 }

```

Define a hash table of all characters with special meaning and add method `reader->add_special_character` that extends the hash table and updates the PEG grammar.

```

5728 local special_characters = {"*", "~", "[", "]", "<", "!", "\\\""}
5729 self.add_special_character = function(c)
5730     table.insert(special_characters, c)
5731     self.syntax.SpecialChar = S(table.concat(special_characters, ""))
5732 end
5733 self.syntax.SpecialChar = S(table.concat(special_characters, ""))

```

Apply syntax extensions.

```

5734 for _, extension in ipairs(extensions) do
5735     extension.extend_writer(writer)
5736     extension.extend_reader(self)
5737 end
5738
5739 if options.underscores then
5740     self.add_special_character("_")
5741 end
5742
5743 if not options.codeSpans then
5744     self.syntax.Code = parsers.fail
5745 end
5746
5747 if not options.html then
5748     self.syntax.DisplayHtml = parsers.fail
5749     self.syntax.InlineHtml = parsers.fail
5750     self.syntax.HtmlEntity = parsers.fail
5751 else
5752     self.add_special_character("&")

```

```

5753 end
5754
5755 if options.preserveTabs then
5756     options.stripIndent = false
5757 end
5758
5759 if not options.smartEllipses then
5760     self.syntax.Smart = parsers.fail
5761 else
5762     self.add_special_character(".")
5763 end
5764
5765 if not options.relativeReferences then
5766     self.syntax.AutoLinkRelativeReference = parsers.fail
5767 end
5768
5769 local blocks_nested_t = util.table_copy(self.syntax)
5770 blocks_nested_t.ExpectedJekyllData = parsers.fail
5771 parsers.blocks_nested = Ct(blocks_nested_t)
5772
5773 parsers.blocks = Ct(self.syntax)
5774
5775 local inlines_t = util.table_copy(self.syntax)
5776 inlines_t[1] = "Inlines"
5777 inlines_t.Inlines = parsers.Inline^0 * (parsers.spacing^0 * parsers.eof / "")
5778 parsers.inlines = Ct(inlines_t)
5779
5780 local inlines_no_link_t = util.table_copy(inlines_t)
5781 inlines_no_link_t.Link = parsers.fail
5782 parsers.inlines_no_link = Ct(inlines_no_link_t)
5783
5784 local inlines_no_inline_note_t = util.table_copy(inlines_t)
5785 inlines_no_inline_note_t.InlineNote = parsers.fail
5786 parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
5787
5788 local inlines_no_html_t = util.table_copy(inlines_t)
5789 inlines_no_html_t.DisplayHtml = parsers.fail
5790 inlines_no_html_t.InlineHtml = parsers.fail
5791 inlines_no_html_t.HtmlEntity = parsers.fail
5792 parsers.inlines_no_html = Ct(inlines_no_html_t)
5793
5794 local inlines_nbsp_t = util.table_copy(inlines_t)
5795 inlines_nbsp_t.Endline = parsers.NonbreakingEndline
5796 inlines_nbsp_t.Space = parsers.NonbreakingSpace
5797 parsers.inlines_nbsp = Ct(inlines_nbsp_t)

```

### 3.1.5.13 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.

```
5798 function self.convert(input)
5799     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.

```
5800     local opt_string = {}
5801     for k,_ in pairs(defaultOptions) do
5802         local v = options[k]
5803         if k ~= "cacheDir" then
5804             opt_string[#opt_string+1] = k .. "=" .. tostring(v)
5805         end
5806     end
5807     table.sort(opt_string)
5808     local salt = table.concat(opt_string, ",") .. "," .. metadata.version
5809     local output
```

If we cache markdown documents, produce the cache file and transform its filename to plain TeX output via the `writer->pack` method.

```
5810     local function convert(input)
5811         local document = self.parser_functions.parse_blocks(input)
5812         return util.rope_to_string(writer.document(document))
5813     end
5814     if options.eagerCache or options.finalizeCache then
5815         local name = util.cache(options.cacheDir, input, salt, convert, ".md" .. writer.s
5816         output = writer.pack(name)
```

Otherwise, return the result of the conversion directly.

```
5817     else
5818         output = convert(input)
5819     end
```

If the `finalizeCache` option is enabled, populate the frozen cache in the file `frozenCacheFileName` with an entry for markdown document number `frozenCacheCounter`.

```
5820     if options.finalizeCache then
5821         local file, mode
5822         if options.frozenCacheCounter > 0 then
5823             mode = "a"
5824         else
5825             mode = "w"
5826         end
5827         file = assert(io.open(options.frozenCacheFileName, mode),
5828             [[could not open file ]] .. options.frozenCacheFileName
5829             .. [[ for writing]])
5830         assert(file:write([[\\expandafter\\global\\expandafter\\def\\csname ]])
```

```

5831         .. [[markdownFrozenCache]] .. options.frozenCacheCounter
5832         .. [[\endcsname{[]}] .. output .. [[]]] .. "\n"))
5833         assert(file:close())
5834     end
5835     return output
5836 end
5837 return self
5838 end

```

### 3.1.6 Syntax Extensions for Markdown

Create `extensions` hash table that contains syntax extensions. Syntax extensions are functions that produce objects with two methods: `extend_writer` and `extend_reader`. The `extend_writer` object takes a `writer` object as the only parameter and mutates it. Similarly, `extend_reader` takes a `reader` object as the only parameter and mutates it.

```

5839 M.extensions = {}

```

**3.1.6.1 Citations** The `extensions.citations` function implements the Pandoc citation syntax extension. When the `citation_nbsps` parameter is enabled, the syntax extension will replace regular spaces with non-breaking spaces inside the prenotes and postnotes of citations.

```

5840 M.extensions.citations = function(citation_nbsps)

```

Define table `escaped_citation_chars` containing the characters to escape in citations.

```

5841     local escaped_citation_chars = {
5842         ["{"] = "\\markdownRendererLeftBrace{}",
5843         ["}"] = "\\markdownRendererRightBrace{}",
5844         ["%"] = "\\markdownRendererPercentSign{}",
5845         ["\\"] = "\\markdownRendererBackslash{}",
5846         ["#"] = "\\markdownRendererHash{}",
5847     }
5848     return {
5849         extend_writer = function(self)
5850             local options = self.options
5851

```

Use the `escaped_citation_chars` to create the `escape_citation` escaper functions.

```

5852         local escape_citation = util.escaper(
5853             escaped_citation_chars,
5854             self.escaped_minimal_strings)

```

Define `writer->citation` as a function that will transform an input citation name `c` to the output format. If `writer->hybrid` is `true`, use the `writer->escape_minimal` function. Otherwise, use the `escape_citation` function.

```
5855     if options.hybrid then
5856         self.citation = self.escape_minimal
5857     else
5858         self.citation = escape_citation
5859     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.

```
5860     function self.citations(text_cites, cites)
5861         local buffer = {"\\markdownRenderer", text_cites and "TextCite" or "Cite",
5862             "{", #cites, "}"}
5863         for _,cite in ipairs(cites) do
5864             buffer[#buffer+1] = {cite.suppress_author and "-" or "+", "{",
5865                 cite.prenote or "", "}{" , cite.postnote or "", "}{" , cite.name, "}"}
5866         end
5867         return buffer
5868     end
5869 end, extend_reader = function(self)
5870     local parsers = self.parsers
5871     local syntax = self.syntax
5872     local writer = self.writer
5873
5874     local citation_chars
5875         = parsers.alphanumeric
5876         + S("#$%&-+<>~/_")
5877
5878     local citation_name
5879         = Cs(parsers.dash^-1) * parsers.at
5880         * Cs(citation_chars
5881             * (((citation_chars + parsers.internal_punctuation
```

```

5882         - parsers.comma - parsers.semicolon)
5883     * -#((parsers.internal_punctuation - parsers.comma
5884         - parsers.semicolon)^0
5885         * -(citation_chars + parsers.internal_punctuation
5886         - parsers.comma - parsers.semicolon)))^0
5887     * citation_chars)^-1)
5888
5889     local citation_body_prenote
5890         = Cs((parsers.alphanumeric^1
5891             + parsers.bracketed
5892             + parsers.inticks
5893             + (parsers.anyescaped
5894               - (parsers.rbracket + parsers.blankline^2))
5895             - (parsers.spnl * parsers.dash^-1 * parsers.at))^0)
5896
5897     local citation_body_postnote
5898         = Cs((parsers.alphanumeric^1
5899             + parsers.bracketed
5900             + parsers.inticks
5901             + (parsers.anyescaped
5902               - (parsers.rbracket + parsers.semicolon
5903                 + parsers.blankline^2))
5904             - (parsers.spnl * parsers.rbracket))^0)
5905
5906     local citation_body_chunk
5907         = citation_body_prenote
5908         * parsers.spnl * citation_name
5909         * (parsers.internal_punctuation - parsers.semicolon)^-
5910         1
5911         * parsers.spnl * citation_body_postnote
5912
5913     local citation_body
5914         = citation_body_chunk
5915         * (parsers.semicolon * parsers.spnl
5916           * citation_body_chunk)^0
5917
5918     local citation_headless_body_postnote
5919         = Cs((parsers.alphanumeric^1
5920             + parsers.bracketed
5921             + parsers.inticks
5922             + (parsers.anyescaped
5923               - (parsers.rbracket + parsers.at
5924                 + parsers.semicolon + parsers.blankline^2))
5925             - (parsers.spnl * parsers.rbracket))^0)
5926
5927     local citation_headless_body
5928         = citation_headless_body_postnote

```



```

5928             * (parsers.sp * parsers.semicolon * parsers.spnl
5929             * citation_body_chunk)^0
5930
5931     local citations
5932         = function(text_cites, raw_cites)
5933         local function normalize(str)
5934             if str == "" then
5935                 str = nil
5936             else
5937                 str = (citation_nbsps and
5938                     self.parser_functions.parse_inlines_nbsp or
5939                     self.parser_functions.parse_inlines)(str)
5940             end
5941             return str
5942         end
5943
5944         local cites = {}
5945         for i = 1,#raw_cites,4 do
5946             cites[#cites+1] = {
5947                 prenote = normalize(raw_cites[i]),
5948                 suppress_author = raw_cites[i+1] == "-",
5949                 name = writer.citation(raw_cites[i+2]),
5950                 postnote = normalize(raw_cites[i+3]),
5951             }
5952         end
5953         return writer.citations(text_cites, cites)
5954     end
5955
5956     local TextCitations
5957         = Ct((parsers.spnl
5958             * Cc("")
5959             * citation_name
5960             * ((parsers.spnl
5961                 * parsers.lbracket
5962                 * citation_headless_body
5963                 * parsers.rbracket) + Cc("")))^1)
5964         / function(raw_cites)
5965             return citations(true, raw_cites)
5966         end
5967
5968     local ParenthesizedCitations
5969         = Ct((parsers.spnl
5970             * parsers.lbracket
5971             * citation_body
5972             * parsers.rbracket)^1)
5973         / function(raw_cites)
5974             return citations(false, raw_cites)

```

```

5975             end
5976
5977     local Citations = TextCitations + ParenthesizedCitations
5978
5979     syntax.Citations = Citations
5980
5981     self.add_special_character("@")
5982     self.add_special_character("-")
5983 end
5984 }
5985 end

```

**3.1.6.2 Content Blocks** The `extensions.content_blocks` function implements the iA,Writer content blocks syntax extension. The `language_map` parameter specifies the filename of the JSON file that maps filename extensions to programming language names.

```

5986 M.extensions.content_blocks = function(language_map)

```

The `languages_json` table maps programming language filename extensions to fence infostrings. All `language_map` 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.

```

5987     local languages_json = (function()
5988         local ran_ok, kpse = pcall(require, "kpse")
5989         if ran_ok then
5990             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:

```

5991     else
5992         kpse = {lookup=function(filename, _) return filename end}
5993     end
5994     local base, prev, curr
5995     for _, filename in ipairs{kpse.lookup(language_map, { all=true })} do
5996         local file = io.open(filename, "r")
5997         if not file then goto continue end
5998         local json = file:read("*all"):gsub('^(\n)-'):gsub('["\n]-'):gsub('["\n]-')
5999         curr = (function()
6000             local _ENV={ json=json, load=load } -- luacheck: ignore _ENV
6001             return load("return "..json)()
6002         end)()
6003         if type(curr) == "table" then
6004             if base == nil then
6005                 base = curr
6006             else

```

```

6007         setmetatable(prev, { __index = curr })
6008     end
6009     prev = curr
6010 end
6011 ::continue::
6012 end
6013 return base or {}
6014 end()
6015
6016 return {
6017     extend_writer = function(self)

```

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.

```

6018     function self.contentblock(src,suf,type,tit)
6019         if not self.is_writing then return "" end
6020         src = src.." "..suf
6021         suf = suf:lower()
6022         if type == "onlineimage" then
6023             return {"\\markdownRendererContentBlockOnlineImage{" ,suf,"} ",
6024                 "{" ,self.string(src),"} ",
6025                 "{" ,self.uri(src),"} ",
6026                 "{" ,self.string(tit or ""),"} "}
6027         elseif languages_json[suf] then
6028             return {"\\markdownRendererContentBlockCode{" ,suf,"} ",
6029                 "{" ,self.string(languages_json[suf]),"} ",
6030                 "{" ,self.string(src),"} ",
6031                 "{" ,self.uri(src),"} ",
6032                 "{" ,self.string(tit or ""),"} "}
6033         else
6034             return {"\\markdownRendererContentBlock{" ,suf,"} ",
6035                 "{" ,self.string(src),"} ",
6036                 "{" ,self.uri(src),"} ",
6037                 "{" ,self.string(tit or ""),"} "}
6038         end
6039     end
6040 end, extend_reader = function(self)
6041     local parsers = self.parsers
6042     local syntax = self.syntax
6043     local writer = self.writer
6044
6045     local contentblock_tail
6046         = parsers.optionaltitle
6047         * (parsers.newline + parsers.eof)
6048

```

```

6049      -- case insensitive online image suffix:
6050      local onlineimagesuffix
6051          = (function(...)
6052              local parser = nil
6053              for _, suffix in ipairs({...}) do
6054                  local pattern=nil
6055                  for i=1,#suffix do
6056                      local char=suffix:sub(i,i)
6057                      char = S(char:lower()..char:upper())
6058                      if pattern == nil then
6059                          pattern = char
6060                      else
6061                          pattern = pattern * char
6062                      end
6063                  end
6064                  if parser == nil then
6065                      parser = pattern
6066                  else
6067                      parser = parser + pattern
6068                  end
6069              end
6070              return parser
6071          end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
6072
6073      -- online image url for iA Writer content blocks with mandatory suffix,
6074      -- allowing nested brackets:
6075      local onlineimageurl
6076          = (parsers.less
6077              * Cs((parsers.anyescaped
6078                  - parsers.more
6079                  - #(parsers.period
6080                      * onlineimagesuffix
6081                      * parsers.more
6082                      * contentblock_tail))^0)
6083              * parsers.period
6084              * Cs(onlineimagesuffix)
6085              * parsers.more
6086              + (Cs((parsers.inparens
6087                  + (parsers.anyescaped
6088                      - parsers.spacing
6089                      - parsers.rparent
6090                      - #(parsers.period
6091                          * onlineimagesuffix
6092                          * contentblock_tail))))^0)
6093              * parsers.period
6094              * Cs(onlineimagesuffix))
6095          ) * Cc("onlineimage")

```

```

6096
6097 -- filename for iA Writer content blocks with mandatory suffix:
6098 local localfilepath
6099     = parsers.slash
6100     * Cs((parsers.anyescaped
6101         - parsers.tab
6102         - parsers.newline
6103         - #(parsers.period
6104             * parsers.alphanumeric~1
6105             * contentblock_tail))^1)
6106     * parsers.period
6107     * Cs(parsers.alphanumeric~1)
6108     * Cc("localfile")
6109
6110 local ContentBlock
6111     = parsers.leader
6112     * (localfilepath + onlineimageurl)
6113     * contentblock_tail
6114     / writer.contentblock
6115
6116 syntax.ContentBlock = ContentBlock
6117 end
6118 }
6119 end

```

**3.1.6.3 Definition Lists** The `extensions.definition_lists` function implements the definition list syntax extension. If the `tight_lists` parameter is `true`, tight lists will produce special right item renderers.

```

6120 M.extensions.definition_lists = function(tight_lists)
6121     return {
6122         extend_writer = function(self)

```

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.

```

6123         local function dlitem(term, defs)
6124             local retVal = {"\\markdownRendererDlItem{",term,""}
6125             for _, def in ipairs(defs) do
6126                 retVal[#retVal+1] = {"\\markdownRendererDlDefinitionBegin ",def,
6127                                     "\\markdownRendererDlDefinitionEnd "}
6128             end
6129             retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
6130             return retVal
6131         end
6132

```

```

6133 function self.definitionlist(items,tight)
6134     if not self.is_writing then return "" end
6135     local buffer = {}
6136     for _,item in ipairs(items) do
6137         buffer[#buffer + 1] = dlitem(item.term, item.definitions)
6138     end
6139     if tight and tight_lists then
6140         return {"\\markdownRendererDlBeginTight\n", buffer,
6141             "\n\\markdownRendererDlEndTight"}
6142     else
6143         return {"\\markdownRendererDlBegin\n", buffer,
6144             "\n\\markdownRendererDlEnd"}
6145     end
6146 end
6147 end, extend_reader = function(self)
6148     local parsers = self.parsers
6149     local syntax = self.syntax
6150     local writer = self.writer
6151
6152     local defstartchar = S("~:")
6153
6154     local defstart = ( defstartchar * #parsers.spacing
6155                         * (parsers.tab + parsers.space~-3)
6156                         + parsers.space * defstartchar * #parsers.spacing
6157                         * (parsers.tab + parsers.space~-2)
6158                         + parsers.space * parsers.space * defstartchar
6159                         * #parsers.spacing
6160                         * (parsers.tab + parsers.space~-1)
6161                         + parsers.space * parsers.space * parsers.space
6162                         * defstartchar * #parsers.spacing
6163                     )
6164
6165     local dlchunk = Cs(parsers.line * (parsers.indentedline - parsers.blankline)^0)
6166
6167     local function definition_list_item(term, defs, _)
6168         return { term = self.parser_functions.parse_inlines(term),
6169             definitions = defs }
6170     end
6171
6172     local DefinitionListItemLoose
6173         = C(parsers.line) * parsers.skipblanklines
6174         * Ct((defstart
6175             * parsers.indented_blocks(dlchunk)
6176             / self.parser_functions.parse_blocks_nested)^1)
6177         * Cc(false) / definition_list_item
6178
6179     local DefinitionListItemTight

```

```

6180         = C(parsers.line)
6181         * Ct((defstart * dlchunk
6182             / self.parser_functions.parse_blocks_nested)^1)
6183         * Cc(true) / definition_list_item
6184
6185     local DefinitionList
6186         = ( Ct(DefinitionListItemLoose^1) * Cc(false)
6187           + Ct(DefinitionListItemTight^1)
6188           * (parsers.skipblanklines
6189             * -DefinitionListItemLoose * Cc(true))
6190           ) / writer.definitionlist
6191
6192     syntax.DefinitionList = DefinitionList
6193 end
6194 }
6195 end

```

**3.1.6.4 Fenced Code** The `extensions.fenced_code` function implements the commonmark fenced code block syntax extension. When the `blank_before_code_fence` parameter is `true`, the syntax extension requires between a paragraph and the following fenced code block.

```

6196 M.extensions.fenced_code = function(blank_before_code_fence)
6197     return {
6198         extend_writer = function(self)
6199             local options = self.options
6200

```

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

```

6201         function self.fencedCode(i, s)
6202             if not self.is_writing then return "" end
6203             s = string.gsub(s, '[\r\n%s]*$', '')
6204             local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
6205             return {"\\markdownRendererInputFencedCode{"..name.."}{"..i.."}"}
6206         end
6207     end, extend_reader = function(self)
6208         local parsers = self.parsers
6209         local syntax = self.syntax
6210         local writer = self.writer
6211
6212         local function captures_geq_length(_, i, a, b)
6213             return #a >= #b and i
6214         end
6215
6216         local infostring = (parsers.linechar - (parsers.backtick
6217             + parsers.space^1 * (parsers.newline + parsers.eof)))^0

```

```

6218
6219     local fenceindent
6220     local fencehead      = function(char)
6221         return           C(parsers.nonindentSPACE) / function(s) fenceindent = #s end
6222                         * Cg(char^3, "fencelength")
6223                         * parsers.optionalspace * C(infostring)
6224                         * parsers.optionalspace * (parsers.newline + parsers.eof)
6225     end
6226
6227     local fencetail      = function(char)
6228         return           parsers.nonindentSPACE
6229                         * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
6230                         * parsers.optionalspace * (parsers.newline + parsers.eof)
6231                         + parsers.eof
6232     end
6233
6234     local fencedline     = function(char)
6235         return           C(parsers.line - fencetail(char))
6236                         / function(s)
6237                             local i = 1
6238                             local remaining = fenceindent
6239                             while true do
6240                                 local c = s:sub(i, i)
6241                                 if c == " " and remaining > 0 then
6242                                     remaining = remaining - 1
6243                                     i = i + 1
6244                                 elseif c == "\t" and remaining > 3 then
6245                                     remaining = remaining - 4
6246                                     i = i + 1
6247                                 else
6248                                     break
6249                                 end
6250                             end
6251                             return s:sub(i)
6252                         end
6253     end
6254
6255     local TildeFencedCode
6256         = fencehead(parsers.tilde)
6257         * Cs(fencedline(parsers.tilde)^0)
6258         * fencetail(parsers.tilde)
6259
6260     local BacktickFencedCode
6261         = fencehead(parsers.backtick)
6262         * Cs(fencedline(parsers.backtick)^0)
6263         * fencetail(parsers.backtick)
6264

```



```

6265     local FencedCode = (TildeFencedCode
6266                          + BacktickFencedCode)
6267     / function(infostring, code)
6268         return writer.fencedCode(writer.string(infostring),
6269                                  self.expandtabs(code))
6270     end
6271
6272     syntax.FencedCode = FencedCode
6273
6274     local fencestart
6275     if blank_before_code_fence then
6276         fencestart = parsers.fail
6277     else
6278         fencestart = fencehead(parsers.backtick)
6279                     + fencehead(parsers.tilde)
6280     end
6281
6282     parsers.EndlineExceptions = parsers.EndlineExceptions + fencestart
6283     syntax.EndlineExceptions = parsers.EndlineExceptions
6284
6285     self.add_special_character("~")
6286 end
6287 }
6288 end

```

**3.1.6.5 Footnotes** The `extensions.footnotes` function implements the Pandoc footnote and inline footnote syntax extensions. When the `footnote` parameter is `true`, the Pandoc footnote syntax extension will be enabled. When the `inline_footnotes` parameter is `true`, the Pandoc inline footnote syntax extension will be enabled.

```

6289 M.extensions.footnotes = function(footnotes, inline_footnotes)
6290     assert(footnotes or inline_footnotes)
6291     return {
6292         extend_writer = function(self)

```

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

```

6293         function self.note(s)
6294             return {"\\markdownRendererFootnote{",s,""}
6295         end
6296     end, extend_reader = function(self)
6297         local parsers = self.parsers
6298         local syntax = self.syntax
6299         local writer = self.writer
6300
6301         if footnotes then

```

```

6302     local function strip_first_char(s)
6303         return s:sub(2)
6304     end
6305
6306     local RawNoteRef
6307         = #(parsers.lbracket * parsers.circumflex)
6308         * parsers.tag / strip_first_char
6309
6310     local rawnotes = {}
6311
6312     -- like indirect_link
6313     local function lookup_note(ref)
6314         return writer.defer_call(function()
6315             local found = rawnotes[self.normalize_tag(ref)]
6316             if found then
6317                 return writer.note(
6318                     self.parser_functions.parse_blocks_nested(found))
6319             else
6320                 return {"[",
6321                     self.parser_functions.parse_inlines("^" .. ref), "]" }
6322             end
6323         end)
6324     end
6325
6326     local function register_note(ref, rawnote)
6327         rawnotes[self.normalize_tag(ref)] = rawnote
6328         return ""
6329     end
6330
6331     local NoteRef = RawNoteRef / lookup_note
6332
6333     local NoteBlock
6334         = parsers.leader * RawNoteRef * parsers.colon
6335         * parsers.spnl * parsers.indented_blocks(parsers.chunk)
6336         / register_note
6337
6338     parsers.Blank = NoteBlock + parsers.Blank
6339     syntax.Blank = parsers.Blank
6340
6341     syntax.NoteRef = NoteRef
6342 end
6343 if inline_footnotes then
6344     local InlineNote
6345         = parsers.circumflex
6346         * (parsers.tag / self.parser_functions.parse_inlines_no_inline_note)
6347         / writer.note
6348     syntax.InlineNote = InlineNote

```

```

6349         end
6350
6351         self.add_special_character("^")
6352     end
6353 }
6354 end

```

**3.1.6.6 Header Attributes** The `extensions.header_attributes` function implements a syntax extension that enables the assignment of HTML attributes to headings.

```

6355 M.extensions.header_attributes = function()
6356     return {
6357         extend_writer = function()
6358         end, extend_reader = function(self)
6359             local parsers = self.parsers
6360             local syntax = self.syntax
6361             local writer = self.writer
6362
6363             parsers.AtxHeading = Cg(parsers.HeadingStart, "level")
6364                 * parsers.optionalspace
6365                 * (C(((parsers.linechar
6366                     - ((parsers.hash^1
6367                         * parsers.optionalspace
6368                         * parsers.HeadingAttributes^-1
6369                         + parsers.HeadingAttributes)
6370                         * parsers.optionalspace
6371                         * parsers.newline)))
6372                     * (parsers.linechar
6373                         - parsers.hash
6374                         - parsers.lbrace)^0)^1)
6375                 / self.parser_functions.parse_inlines)
6376             * Cg(Ct(parsers.newline
6377                 + (parsers.hash^1
6378                     * parsers.optionalspace
6379                     * parsers.HeadingAttributes^-1
6380                     + parsers.HeadingAttributes)
6381                     * parsers.optionalspace
6382                     * parsers.newline), "attributes")
6383             * Cb("level")
6384             * Cb("attributes")
6385             / writer.heading
6386
6387             parsers.SettextHeading = #(parsers.line * S("--"))
6388                 * (C(((parsers.linechar
6389                     - (parsers.HeadingAttributes
6390                         * parsers.optionalspace
6391                         * parsers.newline)))

```

```

6392         * (parsers.linechar
6393           - parsers.lbrace)^0^1)
6394       / self.parser_functions.parse_inlines)
6395     * Cg(Ct(parsers.newline
6396         + (parsers.HeadingAttributes
6397           * parsers.optionalspace
6398           * parsers.newline)), "attributes")
6399     * parsers.HeadingLevel
6400     * Cb("attributes")
6401     * parsers.optionalspace
6402     * parsers.newline
6403     / writer.heading
6404
6405     parsers.Heading = parsers.AtxHeading + parsers.SettextHeading
6406     syntax.Heading = parsers.Heading
6407   end
6408 }
6409 end

```

**3.1.6.7 YAML Metadata** The `extensions.jekyll_data` function implements the Pandoc `yaml_metadata_block` syntax extension for entering metadata in YAML. When the `expect_jekyll_data` parameter is `true`, then a markdown document may begin directly with YAML metadata and may contain nothing but YAML metadata

```

6410 M.extensions.jekyll_data = function(expect_jekyll_data)
6411   return {
6412     extend_writer = function(self)

```

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`.

```

6413     function self.jekyllData(d, t, p)
6414       if not self.is_writing then return "" end
6415
6416       local buf = {}
6417
6418       local keys = {}
6419       for k, _ in pairs(d) do
6420         table.insert(keys, k)
6421       end
6422       table.sort(keys)
6423
6424       if not p then
6425         table.insert(buf, "\\markdownRendererJekyllDataBegin")
6426       end

```

```

6427
6428     if #d > 0 then
6429         table.insert(buf, "\\markdownRendererJekyllDataSequenceBegin{")
6430         table.insert(buf, self.uri(p or "null"))
6431         table.insert(buf, "{")
6432         table.insert(buf, #keys)
6433         table.insert(buf, "}")
6434     else
6435         table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
6436         table.insert(buf, self.uri(p or "null"))
6437         table.insert(buf, "{")
6438         table.insert(buf, #keys)
6439         table.insert(buf, "}")
6440     end
6441
6442     for _, k in ipairs(keys) do
6443         local v = d[k]
6444         local typ = type(v)
6445         k = tostring(k or "null")
6446         if typ == "table" and next(v) ~= nil then
6447             table.insert(
6448                 buf,
6449                 self.jekyllData(v, t, k)
6450             )
6451         else
6452             k = self.uri(k)
6453             v = tostring(v)
6454             if typ == "boolean" then
6455                 table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
6456                 table.insert(buf, k)
6457                 table.insert(buf, "{")
6458                 table.insert(buf, v)
6459                 table.insert(buf, "}")
6460             elseif typ == "number" then
6461                 table.insert(buf, "\\markdownRendererJekyllDataNumber{")
6462                 table.insert(buf, k)
6463                 table.insert(buf, "{")
6464                 table.insert(buf, v)
6465                 table.insert(buf, "}")
6466             elseif typ == "string" then
6467                 table.insert(buf, "\\markdownRendererJekyllDataString{")
6468                 table.insert(buf, k)
6469                 table.insert(buf, "{")
6470                 table.insert(buf, t(v))
6471                 table.insert(buf, "}")
6472             elseif typ == "table" then
6473                 table.insert(buf, "\\markdownRendererJekyllDataEmpty{")

```

```

6474         table.insert(buf, k)
6475         table.insert(buf, "}")
6476     else
6477         error(format("Unexpected type %s for value of " ..
6478             "YAML key %s", typ, k))
6479     end
6480 end
6481 end
6482
6483 if #d > 0 then
6484     table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
6485 else
6486     table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
6487 end
6488
6489 if not p then
6490     table.insert(buf, "\\markdownRendererJekyllDataEnd")
6491 end
6492
6493 return buf
6494 end
6495 end, extend_reader = function(self)
6496     local parsers = self.parsers
6497     local syntax = self.syntax
6498     local writer = self.writer
6499
6500     local JekyllData
6501         = Cmt( C((parsers.line - P("---") - P("..."))^0)
6502             , function(s, i, text) -- luacheck: ignore s i
6503                 local data
6504                 local ran_ok, _ = pcall(function()
6505                     local tinyyaml = require("markdown-tinyyaml")
6506                     data = tinyyaml.parse(text, {timestamps=false})
6507                 end)
6508                 if ran_ok and data ~= nil then
6509                     return true, writer.jekyllData(data, function(s)
6510                         return self.parser_functions.parse_blocks_nested(s)
6511                     end, nil)
6512                 else
6513                     return false
6514                 end
6515             end
6516         )
6517
6518     local UnexpectedJekyllData
6519         = P("---")
6520         * parsers.blankline / 0

```

```

6521             * #(-parsers.blankline) -- if followed by blank, it's an hrule
6522             * JekyllData
6523             * (P("----") + P("..."))
6524
6525     local ExpectedJekyllData
6526         = ( P("----")
6527             * parsers.blankline / 0
6528             * #(-parsers.blankline) -- if followed by blank, it's an hrule
6529             )^-1
6530             * JekyllData
6531             * (P("----") + P("..."))^-1
6532
6533     syntax.UnexpectedJekyllData = UnexpectedJekyllData
6534     if expect_jekyll_data then
6535         syntax.ExpectedJekyllData = ExpectedJekyllData
6536     end
6537 end
6538 }
6539 end

```

**3.1.6.8 Pipe Tables** The `extensions.pipe_table` function implements the PHP Markdown table syntax extension (affectionately known as pipe tables). When the `table_captions` parameter is `true`, the function also implements the Pandoc `table_captions` syntax extension for table captions.

```

6540 M.extensions.pipe_tables = function(table_captions)
6541
6542     local function make_pipe_table_rectangular(rows)
6543         local num_columns = #rows[2]
6544         local rectangular_rows = {}
6545         for i = 1, #rows do
6546             local row = rows[i]
6547             local rectangular_row = {}
6548             for j = 1, num_columns do
6549                 rectangular_row[j] = row[j] or ""
6550             end
6551             table.insert(rectangular_rows, rectangular_row)
6552         end
6553         return rectangular_rows
6554     end
6555
6556     local function pipe_table_row(allow_empty_first_column
6557                                   , nonempty_column
6558                                   , column_separator
6559                                   , column)
6560         local row_beginning
6561         if allow_empty_first_column then

```

```

6562         row_beginning = -- empty first column
6563                         #(parsers.spacechar^4
6564                         * column_separator)
6565                         * parsers.optionalspace
6566                         * column
6567                         * parsers.optionalspace
6568         -- non-empty first column
6569         + parsers.nonindentspace
6570         * nonempty_column^-1
6571         * parsers.optionalspace
6572     else
6573         row_beginning = parsers.nonindentspace
6574                         * nonempty_column^-1
6575                         * parsers.optionalspace
6576     end
6577
6578     return Ct(row_beginning
6579             * (-- single column with no leading pipes
6580             #(column_separator
6581             * parsers.optionalspace
6582             * parsers.newline)
6583             * column_separator
6584             * parsers.optionalspace
6585             -- single column with leading pipes or
6586             -- more than a single column
6587             + (column_separator
6588             * parsers.optionalspace
6589             * column
6590             * parsers.optionalspace)^1
6591             * (column_separator
6592             * parsers.optionalspace)^-1))
6593 end
6594
6595 return {
6596     extend_writer = function(self)

```

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.

```

6597     function self.table(rows, caption)
6598         if not self.is_writing then return "" end
6599         local buffer = {"\\markdownRendererTable{",
6600             caption or "", "{", #rows - 1, "{", #rows[1], "}"}
6601         local temp = rows[2] -- put alignments on the first row
6602         rows[2] = rows[1]
6603         rows[1] = temp
6604         for i, row in ipairs(rows) do

```



```

6605         table.insert(buffer, "{")
6606         for _, column in ipairs(row) do
6607             if i > 1 then -- do not use braces for alignments
6608                 table.insert(buffer, "{")
6609             end
6610             table.insert(buffer, column)
6611             if i > 1 then
6612                 table.insert(buffer, "}")
6613             end
6614         end
6615         table.insert(buffer, "}")
6616     end
6617     return buffer
6618 end
6619 end, extend_reader = function(self)
6620     local parsers = self.parsers
6621     local syntax = self.syntax
6622     local writer = self.writer
6623
6624     local table_hline_separator = parsers.pipe + parsers.plus
6625
6626     local table_hline_column = (parsers.dash
6627                                - #(parsers.dash
6628                                   * (parsers.spacechar
6629                                      + table_hline_separator
6630                                      + parsers.newline)))^1
6631                                * (parsers.colon * Cc("r")
6632                                   + parsers.dash * Cc("d"))
6633                                + parsers.colon
6634                                * (parsers.dash
6635                                   - #(parsers.dash
6636                                      * (parsers.spacechar
6637   + table_hline_separator
6638   + parsers.newline)))^1
6639                                * (parsers.colon * Cc("c")
6640                                   + parsers.dash * Cc("l"))
6641
6642     local table_hline = pipe_table_row(false
6643   , table_hline_column
6644   , table_hline_separator
6645   , table_hline_column)
6646
6647     local table_caption_beginning = parsers.skipblanklines
6648                                     * parsers.nonindentspace
6649                                     * (P("Table")^-1 * parsers.colon)
6650                                     * parsers.optionalspace
6651

```

```

6652     local table_row = pipe_table_row(true
6653                                     , (C((parsers.linechar - parsers.pipe)^1)
6654                                     / self.parser_functions.parse_inlines)
6655                                     , parsers.pipe
6656                                     , (C((parsers.linechar - parsers.pipe)^0)
6657                                     / self.parser_functions.parse_inlines))
6658
6659     local table_caption
6660     if table_captions then
6661         table_caption = #table_caption_beginning
6662                       * table_caption_beginning
6663                       * Ct(parsers.IndentedInline^1)
6664                       * parsers.newline
6665     else
6666         table_caption = parsers.fail
6667     end
6668
6669     local PipeTable = Ct(table_row * parsers.newline
6670                       * table_hline
6671                       * (parsers.newline * table_row)^0)
6672                       / make_pipe_table_rectangular
6673                       * table_caption^-1
6674                       / writer.table
6675
6676     syntax.PipeTable = PipeTable
6677 end
6678 }
6679 end

```

**3.1.6.9 Strike-Through** The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```

6680 M.extensions.strike_through = function()
6681     return {
6682         extend_writer = function(self)

```

Define `writer->strike_through` as a function that will transform a strike-through span `s` of input text to the output format.

```

6683         function self.strike_through(s)
6684             return {"\\markdownRendererStrikeThrough{" ,s,"}"}
6685         end
6686     end, extend_reader = function(self)
6687         local parsers = self.parsers
6688         local syntax = self.syntax
6689         local writer = self.writer
6690
6691         local StrikeThrough = (
6692             parsers.between(parsers.Inline, parsers.doubletildes,

```

```

6693             parsers.doubletildes)
6694         ) / writer.strike_through
6695
6696         syntax.StrikeThrough = StrikeThrough
6697
6698         self.add_special_character("~")
6699     end
6700 }
6701 end

```

**3.1.6.10 Superscripts** The `extensions.superscripts` function implements the Pandoc superscript syntax extension.

```

6702 M.extensions.superscripts = function()
6703     return {
6704         extend_writer = function(self)

```

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

```

6705             function self.superscript(s)
6706                 return {"\\markdownRendererSuperscript{",s,""}
6707             end
6708         end, extend_reader = function(self)
6709             local parsers = self.parsers
6710             local syntax = self.syntax
6711             local writer = self.writer
6712
6713             local Superscript = (
6714                 parsers.between(parsers.Str, parsers.circumflex, parsers.circumflex)
6715             ) / writer.superscript
6716
6717             syntax.Superscript = Superscript
6718
6719             self.add_special_character("^")
6720         end
6721     }
6722 end

```

**3.1.6.11 Subscripts** The `extensions.subscripts` function implements the Pandoc subscript syntax extension.

```

6723 M.extensions.subscripts = function()
6724     return {
6725         extend_writer = function(self)

```

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

```

6726             function self.subscript(s)

```

```

6727         return {"\\markdownRendererSubscript{"s,"}"}
6728     end
6729 end, extend_reader = function(self)
6730     local parsers = self.parsers
6731     local syntax = self.syntax
6732     local writer = self.writer
6733
6734     local Subscript = (
6735         parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
6736     ) / writer.subscript
6737
6738     syntax.Subscript = Subscript
6739
6740     self.add_special_character("~")
6741 end
6742 }
6743 end

```

**3.1.6.12 Fancy Lists** The `extensions.fancy_lists` function implements the Pandoc fancy list extension.

```

6744 M.extensions.fancy_lists = function()
6745     return {
6746         extend_writer = function(self)
6747             local options = self.options
6748

```

Define `writer->fancylist` as a function that will transform an input ordered list to the output format, where:

- `items` is an array of the list items,
- `tight` specifies, whether the list is tight or not,
- `startnum` is the number of the first list item,
- `numstyle` is the style of the list item labels from among the following:
  - `Decimal` – decimal arabic numbers,
  - `LowerRoman` – lower roman numbers,
  - `UpperRoman` – upper roman numbers,
  - `LowerAlpha` – lower ASCII alphabetic characters, and
  - `UpperAlpha` – upper ASCII alphabetic characters, and
- `numdelim` is the style of delimiters between list item labels and texts from among the following:
  - `Default` – default style,
  - `OneParen` – parentheses, and
  - `Period` – periods.

```

6749     function self.fancylist(items,tight,startnum,numstyle,numdelim)
6750         if not self.is_writing then return "" end
6751         local buffer = {}

```

```

6752     local num = startnum
6753     for _,item in ipairs(items) do
6754         buffer[#buffer + 1] = self.fancyitem(item,num)
6755         if num ~= nil then
6756             num = num + 1
6757         end
6758     end
6759     local contents = util.intersperse(buffer,"\n")
6760     if tight and options.tightLists then
6761         return {"\\markdownRenderFancy01BeginTight{",
6762             numstyle,"}{",numdelim,"}",contents,
6763             "\\markdownRenderFancy01EndTight "}
6764     else
6765         return {"\\markdownRenderFancy01Begin{",
6766             numstyle,"}{",numdelim,"}",contents,
6767             "\\markdownRenderFancy01End "}
6768     end
6769 end

```

Define `writer->fancyitem` as a function that will transform an input fancy ordered list item to the output format, where `s` is the text of the list item. If the optional parameter `num` is present, it is the number of the list item.

```

6770     function self.fancyitem(s,num)
6771         if num ~= nil then
6772             return {"\\markdownRenderFancy01ItemWithNumber{",num,"}",s,
6773                 "\\markdownRenderFancy01ItemEnd "}
6774         else
6775             return {"\\markdownRenderFancy01Item ",s,"\\markdownRenderFancy01ItemEnd "
6776         end
6777     end
6778 end, extend_reader = function(self)
6779     local parsers = self.parsers
6780     local options = self.options
6781     local syntax = self.syntax
6782     local writer = self.writer
6783
6784     local label = parsers.dig + parsers.letter
6785     local numdelim = parsers.period + parsers.rparent
6786     local enumerator = C(label^3 * numdelim) * #parsers.spacing
6787         + C(label^2 * numdelim) * #parsers.spacing
6788         * (parsers.tab + parsers.space^1)
6789         + C(label * numdelim) * #parsers.spacing
6790         * (parsers.tab + parsers.space^-2)
6791         + parsers.space * C(label^2 * numdelim)
6792         * #parsers.spacing
6793         + parsers.space * C(label * numdelim)
6794         * #parsers.spacing

```

```

6795             * (parsers.tab + parsers.space^-1)
6796             + parsers.space * parsers.space * C(label^1
6797             * numdelim) * #parsers.spacing
6798     local starter = parsers.bullet + enumerator
6799
6800     local NestedList = Cs((parsers.optionallyindentedline
6801             - starter)^1)
6802             / function(a) return "\001"..a end
6803
6804     local ListBlockLine = parsers.optionallyindentedline
6805             - parsers.blankline - (parsers.indent^-1
6806             * starter)
6807
6808     local ListBlock = parsers.line * ListBlockLine^0
6809
6810     local ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
6811             * ListBlock
6812
6813     local TightListItem = function(starter)
6814         return -parsers.HorizontalRule
6815             * (Cs(starter / "" * parsers.tickbox^-1 * ListBlock * NestedList^-
1)
6816             / self.parser_functions.parse_blocks_nested)
6817             * -(parsers.blanklines * parsers.indent)
6818     end
6819
6820     local LooseListItem = function(starter)
6821         return -parsers.HorizontalRule
6822             * Cs( starter / "" * parsers.tickbox^-1 * ListBlock * Cc("\n")
6823             * (NestedList + ListContinuationBlock^0)
6824             * (parsers.blanklines / "\n\n")
6825             ) / self.parser_functions.parse_blocks_nested
6826     end
6827
6828     local function roman2number(roman)
6829         local romans = { ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
6830         local numeral = 0
6831
6832         local i = 1
6833         local len = string.len(roman)
6834         while i < len do
6835             local z1, z2 = romans[ string.sub(roman, i, i) ], romans[ string.sub(roman, i,
6836             if z1 < z2 then
6837                 numeral = numeral + (z2 - z1)
6838                 i = i + 2
6839             else
6840                 numeral = numeral + z1

```

```

6841         i = i + 1
6842     end
6843 end
6844 if i <= len then numeral = numeral + romans[ string.sub(roman,i,i) ] end
6845 return numeral
6846 end
6847
6848 local function sniffstyle(itemprefix)
6849     local numstr, delimend = itemprefix:match("^([A-Za-z0-9]*)([.])*")
6850     local numdelim
6851     if delimend == ")" then
6852         numdelim = "OneParen"
6853     elseif delimend == "." then
6854         numdelim = "Period"
6855     else
6856         numdelim = "Default"
6857     end
6858     numstr = numstr or itemprefix
6859
6860     local num
6861     num = numstr:match("^([IVXL]+)")
6862     if num then
6863         return roman2number(num), "UpperRoman", numdelim
6864     end
6865     num = numstr:match("^([ivxl]+)")
6866     if num then
6867         return roman2number(string.upper(num)), "LowerRoman", numdelim
6868     end
6869     num = numstr:match("^([A-Z])")
6870     if num then
6871         return string.byte(num) - string.byte("A") + 1, "UpperAlpha", numdelim
6872     end
6873     num = numstr:match("^([a-z])")
6874     if num then
6875         return string.byte(num) - string.byte("a") + 1, "LowerAlpha", numdelim
6876     end
6877     return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
6878 end
6879
6880 local function fancylist(items,tight,start)
6881     local startnum, numstyle, numdelim = sniffstyle(start)
6882     return writer.fancylist(items,tight,
6883                             options.startNumber and startnum,
6884                             numstyle or "Decimal",
6885                             numdelim or "Default")
6886 end
6887

```

```

6888     local FancyList = Cg(enumerator, "listtype") *
6889         ( Ct(TightListItem(Cb("listtype")))
6890           * TightListItem(enumerator)^0
6891           * Cc(true) * parsers.skipblanklines * -enumerator
6892           + Ct(LooseListItem(Cb("listtype")))
6893             * LooseListItem(enumerator)^0
6894           * Cc(false) * parsers.skipblanklines
6895           ) * Cb("listtype") / fancylist
6896
6897     syntax.OrderedList = FancyList
6898
6899     end
6900 }
6901 end

```

### 3.1.7 Conversion from Markdown to Plain T<sub>E</sub>X

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`.

```

6902 function M.new(options)
    Make the options table inherit from the defaultOptions table.
6903     options = options or {}
6904     setmetatable(options, { __index = function (_, key)
6905         return defaultOptions[key] end })
6906 % \par
6907 % \begin{markdown}
6908 %
6909 % Apply syntax extensions based on `options`.
6910 %
6911 % \end{markdown}
6912 % \begin{macrocode}
6913     local extensions = {}
6914
6915     if options.citations then
6916         local citations_extension = M.extensions.citations(options.citationNbsps)
6917         table.insert(extensions, citations_extension)
6918     end
6919
6920     if options.contentBlocks then
6921         local content_blocks_extension = M.extensions.content_blocks(
6922             options.contentBlocksLanguageMap)
6923         table.insert(extensions, content_blocks_extension)
6924     end
6925
6926     if options.definitionLists then

```



```

6927     local definition_lists_extension = M.extensions.definition_lists(
6928         options.tightLists)
6929     table.insert(extensions, definition_lists_extension)
6930 end
6931
6932 if options.fencedCode then
6933     local fenced_code_extension = M.extensions.fenced_code(
6934         options.blankBeforeCodeFence)
6935     table.insert(extensions, fenced_code_extension)
6936 end
6937
6938 if options.footnotes or options.inlineFootnotes then
6939     local footnotes_extension = M.extensions.footnotes(
6940         options.footnotes, options.inlineFootnotes)
6941     table.insert(extensions, footnotes_extension)
6942 end
6943
6944 if options.headerAttributes then
6945     local header_attributes_extension = M.extensions.header_attributes()
6946     table.insert(extensions, header_attributes_extension)
6947 end
6948
6949 if options.jekyllData then
6950     local jekyll_data_extension = M.extensions.jekyll_data(
6951         options.expectJekyllData)
6952     table.insert(extensions, jekyll_data_extension)
6953 end
6954
6955 if options.pipeTables then
6956     local pipe_tables_extension = M.extensions.pipe_tables(
6957         options.tableCaptions)
6958     table.insert(extensions, pipe_tables_extension)
6959 end
6960
6961 if options.strikeThrough then
6962     local strike_through_extension = M.extensions.strike_through()
6963     table.insert(extensions, strike_through_extension)
6964 end
6965
6966 if options.subscripts then
6967     local subscript_extension = M.extensions.subscripts()
6968     table.insert(extensions, subscript_extension)
6969 end
6970
6971 if options.superscripts then
6972     local superscript_extension = M.extensions.superscripts()
6973     table.insert(extensions, superscript_extension)

```

```

6974 end
6975
6976 if options.fancyLists then
6977     local fancy_lists_extension = M.extensions.fancy_lists()
6978     table.insert(extensions, fancy_lists_extension)
6979 end
6980
6981 local writer = M.writer.new(options)
6982 local reader = M.reader.new(writer, options, extensions)
6983
6984 return reader.convert
6985 end
6986
6987 return M

```

### 3.1.8 Command-Line Implementation

The command-line implementation provides the actual conversion routine for the command-line interface described in Section 2.1.5.

```

6988
6989 local input
6990 if input_filename then
6991     local input_file = assert(io.open(input_filename, "r"),
6992     [[could not open file ]] .. input_filename .. [[ for reading]])
6993     input = assert(input_file:read("*a"))
6994     assert(input_file:close())
6995 else
6996     input = assert(io.read("*a"))
6997 end
6998

```

First, ensure that the `options.cacheDir` directory exists.

```

6999 local lfs = require("lfs")
7000 if options.cacheDir and not lfs.isdir(options.cacheDir) then
7001     assert(lfs.mkdir(options["cacheDir"]))
7002 end
7003
7004 local ran_ok, kpse = pcall(require, "kpse")
7005 if ran_ok then kpse.set_program_name("luatex") end
7006 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.

```

7007 if metadata.version ~= md.metadata.version then
7008     warn("markdown-cli.lua " .. metadata.version .. " used with " ..
7009     "markdown.lua " .. md.metadata.version .. ".")
7010 end

```

```
7011 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.

```
7012 local output = convert(input:gsub("\r\n?", "\n") .. "\n")
7013
7014 if output_filename then
7015   local output_file = assert(io.open(output_filename, "w"),
7016     [[could not open file ]] .. output_filename .. [[ for writing]])
7017   assert(output_file:write(output))
7018   assert(output_file:close())
7019 else
7020   assert(io.write(output))
7021 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

```
7022 \ifx\markdownInfo\undefined
7023   \def\markdownInfo#1{%
7024     \immediate\write-1{(1.\the\inputlineno) markdown.tex info: #1}}%
7025 \fi
7026 \ifx\markdownWarning\undefined
7027   \def\markdownWarning#1{%
7028     \immediate\write16{(1.\the\inputlineno) markdown.tex warning: #1}}%
7029 \fi
7030 \ifx\markdownError\undefined
7031   \def\markdownError#1#2{%
7032     \errhelp{#2.}%
7033     \errmessage{(1.\the\inputlineno) markdown.tex error: #1}}%
7034 \fi
```

### 3.2.2 Token Renderer Prototypes

The following definitions should be considered placeholder.

```
7035 \def\markdownRendererInterblockSeparatorPrototype{\par}%
7036 \def\markdownRendererLineBreakPrototype{\hfil\break}%
7037 \let\markdownRendererEllipsisPrototype\dots
7038 \def\markdownRendererNbspPrototype{~}%
7039 \def\markdownRendererLeftBracePrototype{\char`\{}%
7040 \def\markdownRendererRightBracePrototype{\char`\}}%
```

```

7041 \def\markdownRendererDollarSignPrototype{\char`$}%
7042 \def\markdownRendererPercentSignPrototype{\char`\}%
7043 \def\markdownRendererAmpersandPrototype{\&}%
7044 \def\markdownRendererUnderscorePrototype{\char`_}%
7045 \def\markdownRendererHashPrototype{\char`#}%
7046 \def\markdownRendererCircumflexPrototype{\char`^}%
7047 \def\markdownRendererBackslashPrototype{\char`\}%
7048 \def\markdownRendererTildePrototype{\char`~}%
7049 \def\markdownRendererPipePrototype{|}%
7050 \def\markdownRendererCodeSpanPrototype#1{\tt#1}%
7051 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
7052 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
7053   \markdownInput{#3}}%
7054 \def\markdownRendererContentBlockOnlineImagePrototype{%
7055   \markdownRendererImage}%
7056 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
7057   \markdownRendererInputFencedCode{#3}{#2}}%
7058 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
7059 \def\markdownRendererUlBeginPrototype{}%
7060 \def\markdownRendererUlBeginTightPrototype{}%
7061 \def\markdownRendererUlItemPrototype{}%
7062 \def\markdownRendererUlItemEndPrototype{}%
7063 \def\markdownRendererUlEndPrototype{}%
7064 \def\markdownRendererUlEndTightPrototype{}%
7065 \def\markdownRendererOlBeginPrototype{}%
7066 \def\markdownRendererOlBeginTightPrototype{}%
7067 \def\markdownRendererFancyOlBeginPrototype#1#2{\markdownRendererOlBegin}%
7068 \def\markdownRendererFancyOlBeginTightPrototype#1#2{\markdownRendererOlBeginTight}%
7069 \def\markdownRendererOlItemPrototype{}%
7070 \def\markdownRendererOlItemWithNumberPrototype#1{}%
7071 \def\markdownRendererOlItemEndPrototype{}%
7072 \def\markdownRendererFancyOlItemPrototype{\markdownRendererOlItem}%
7073 \def\markdownRendererFancyOlItemWithNumberPrototype{\markdownRendererOlItemWithNumber}%
7074 \def\markdownRendererFancyOlItemEndPrototype{}%
7075 \def\markdownRendererOlEndPrototype{}%
7076 \def\markdownRendererOlEndTightPrototype{}%
7077 \def\markdownRendererFancyOlEndPrototype{\markdownRendererOlEnd}%
7078 \def\markdownRendererFancyOlEndTightPrototype{\markdownRendererOlEndTight}%
7079 \def\markdownRendererDlBeginPrototype{}%
7080 \def\markdownRendererDlBeginTightPrototype{}%
7081 \def\markdownRendererDlItemPrototype#1{#1}%
7082 \def\markdownRendererDlItemEndPrototype{}%
7083 \def\markdownRendererDlDefinitionBeginPrototype{}%
7084 \def\markdownRendererDlDefinitionEndPrototype{\par}%
7085 \def\markdownRendererDlEndPrototype{}%
7086 \def\markdownRendererDlEndTightPrototype{}%
7087 \def\markdownRendererEmphasisPrototype#1{\it#1}%

```

```

7088 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}}%
7089 \def\markdownRendererBlockQuoteBeginPrototype{\par\begin group\it}%
7090 \def\markdownRendererBlockQuoteEndPrototype{\end group\par}%
7091 \def\markdownRendererInputVerbatimPrototype#1{%
7092   \par{\tt\input#1\relax{}}\par}%
7093 \def\markdownRendererInputFencedCodePrototype#1#2{%
7094   \markdownRendererInputVerbatimPrototype{#1}}%
7095 \def\markdownRendererHeadingOnePrototype#1{#1}%
7096 \def\markdownRendererHeadingTwoPrototype#1{#1}%
7097 \def\markdownRendererHeadingThreePrototype#1{#1}%
7098 \def\markdownRendererHeadingFourPrototype#1{#1}%
7099 \def\markdownRendererHeadingFivePrototype#1{#1}%
7100 \def\markdownRendererHeadingSixPrototype#1{#1}%
7101 \def\markdownRendererHorizontalRulePrototype{}%
7102 \def\markdownRendererFootnotePrototype#1{#1}%
7103 \def\markdownRendererCitePrototype#1{}%
7104 \def\markdownRendererTextCitePrototype#1{}%
7105 \def\markdownRendererTickedBoxPrototype{[X]}%
7106 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
7107 \def\markdownRendererUntickedBoxPrototype{[ ]}%
7108 \def\markdownRendererStrikeThroughPrototype#1{#1}%
7109 \def\markdownRendererSuperscriptPrototype#1{#1}%
7110 \def\markdownRendererSubscriptPrototype#1{#1}%

```

**3.2.2.1 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_t1` The currently traversed branch of the YAML document contains a sequence at depth  $p$ .

`\c_@@_jekyll_data_mapping_t1` The currently traversed branch of the YAML document contains a mapping at depth  $p$ .

`\c_@@_jekyll_data_scalar_t1` The currently traversed branch of the YAML document contains a scalar value at depth  $p$ .

```

7111 \ExplSyntaxOn
7112 \seq_new:N \g_@@_jekyll_data_datatypes_seq
7113 \tl_const:Nn \c_@@_jekyll_data_sequence_t1 { sequence }
7114 \tl_const:Nn \c_@@_jekyll_data_mapping_t1 { mapping }
7115 \tl_const:Nn \c_@@_jekyll_data_scalar_t1 { 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.

```

7116 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
7117 \cs_new:Nn \markdown_jekyll_data_push_address_segment:n
7118 {
7119     \seq_if_empty:NF
7120     \g_@@_jekyll_data_datatypes_seq
7121     {
7122         \seq_get_right:NN
7123         \g_@@_jekyll_data_datatypes_seq
7124         \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.

```

7125     \str_if_eq:NNTF
7126     \l_tmpa_tl
7127     \c_@@_jekyll_data_sequence_tl
7128     {
7129         \seq_put_right:Nn
7130         \g_@@_jekyll_data_wildcard_absolute_address_seq
7131         { * }
7132     }
7133     {
7134         \seq_put_right:Nn
7135         \g_@@_jekyll_data_wildcard_absolute_address_seq
7136         { #1 }
7137     }
7138 }
7139 }

```

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.

```

7140 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
7141 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
7142 \cs_new:Nn \markdown_jekyll_data_concatenate_address:NN
7143 {
7144   \seq_pop_left:NN #1 \l_tmpa_tl
7145   \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
7146   \seq_put_left:NV #1 \l_tmpa_tl
7147 }
7148 \cs_new:Nn \markdown_jekyll_data_update_address_tls:
7149 {
7150   \markdown_jekyll_data_concatenate_address:NN
7151   \g_@@_jekyll_data_wildcard_absolute_address_seq
7152   \g_@@_jekyll_data_wildcard_absolute_address_tl
7153   \seq_get_right:NN
7154   \g_@@_jekyll_data_wildcard_absolute_address_seq
7155   \g_@@_jekyll_data_wildcard_relative_address_tl
7156 }
```

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.

```

7157 \cs_new:Nn \markdown_jekyll_data_push:nN
7158 {
7159   \markdown_jekyll_data_push_address_segment:n
7160   { #1 }
7161   \seq_put_right:NV
7162   \g_@@_jekyll_data_datatypes_seq
7163   #2
7164   \markdown_jekyll_data_update_address_tls:
7165 }
7166 \cs_new:Nn \markdown_jekyll_data_pop:
7167 {
7168   \seq_pop_right:NN
7169   \g_@@_jekyll_data_wildcard_absolute_address_seq
7170   \l_tmpa_tl
7171   \seq_pop_right:NN
7172   \g_@@_jekyll_data_datatypes_seq
7173   \l_tmpa_tl
```

```

7174     \markdown_jekyll_data_update_address_tls:
7175 }

```

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.

```

7176 \cs_new:Nn \markdown_jekyll_data_set_keyval:nn
7177 {
7178     \keys_set_known:nn
7179     { markdown/jekyllData }
7180     { { #1 } = { #2 } }
7181 }
7182 \cs_generate_variant:Nn
7183 \markdown_jekyll_data_set_keyval:nn
7184 { Vn }
7185 \cs_new:Nn \markdown_jekyll_data_set_keyvals:nn
7186 {
7187     \markdown_jekyll_data_push:nN
7188     { #1 }
7189     \c_@@_jekyll_data_scalar_tl
7190     \markdown_jekyll_data_set_keyval:Vn
7191     \g_@@_jekyll_data_wildcard_absolute_address_tl
7192     { #2 }
7193     \markdown_jekyll_data_set_keyval:Vn
7194     \g_@@_jekyll_data_wildcard_relative_address_tl
7195     { #2 }
7196     \markdown_jekyll_data_pop:
7197 }

```

Finally, we will register our macros as token renderer prototypes to be able to react to the traversal of a YAML document.

```

7198 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
7199     \markdown_jekyll_data_push:nN
7200     { #1 }
7201     \c_@@_jekyll_data_sequence_tl
7202 }
7203 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
7204     \markdown_jekyll_data_push:nN
7205     { #1 }
7206     \c_@@_jekyll_data_mapping_tl
7207 }
7208 \def\markdownRendererJekyllDataSequenceEndPrototype{
7209     \markdown_jekyll_data_pop:
7210 }
7211 \def\markdownRendererJekyllDataMappingEndPrototype{
7212     \markdown_jekyll_data_pop:
7213 }
7214 \def\markdownRendererJekyllDataBooleanPrototype#1#2{

```



```

7215 \markdown_jekyll_data_set_keyvals:nn
7216   { #1 }
7217   { #2 }
7218 }
7219 \def\markdownRendererJekyllDataEmptyPrototype#1{}
7220 \def\markdownRendererJekyllDataNumberPrototype#1#2{
7221   \markdown_jekyll_data_set_keyvals:nn
7222     { #1 }
7223     { #2 }
7224 }
7225 \def\markdownRendererJekyllDataStringPrototype#1#2{
7226   \markdown_jekyll_data_set_keyvals:nn
7227     { #1 }
7228     { #2 }
7229 }
7230 \ExplSyntaxOff

```

### 3.2.3 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).

```

7231 \ExplSyntaxOn
7232 \tl_new:N \g_@@_formatted_lua_options_tl
7233 \cs_new:Nn \@@_format_lua_options:
7234 {
7235   \tl_gclear:N
7236   \g_@@_formatted_lua_options_tl
7237   \seq_map_function:NN
7238     \g_@@_lua_options_seq
7239     \@@_format_lua_option:n
7240 }
7241 \cs_new:Nn \@@_format_lua_option:n
7242 {
7243   \@@_typecheck_option:n
7244     { #1 }
7245   \@@_get_option_type:nN
7246     { #1 }
7247   \l_tmpa_tl
7248   \bool_if:nTF
7249     {
7250       \str_if_eq_p:VV
7251         \l_tmpa_tl
7252         \c_@@_option_type_boolean_tl ||
7253       \str_if_eq_p:VV
7254         \l_tmpa_tl

```

```

7255         \c_@@_option_type_number_tl ||
7256     \str_if_eq_p:VV
7257         \l_tmpa_tl
7258         \c_@@_option_type_counter_tl
7259     }
7260     {
7261         \@@_get_option_value:nN
7262         { #1 }
7263         \l_tmpa_tl
7264         \tl_gput_right:Nx
7265         \g_@@_formatted_lua_options_tl
7266         { #1~::~ \l_tmpa_tl ,~ }
7267     }
7268     {
7269         \@@_get_option_value:nN
7270         { #1 }
7271         \l_tmpa_tl
7272         \tl_gput_right:Nx
7273         \g_@@_formatted_lua_options_tl
7274         { #1~::~ " \l_tmpa_tl " ,~ }
7275     }
7276 }
7277 \let\markdownPrepareLuaOptions=\@@_format_lua_options:
7278 \def\markdownLuaOptions{{ \g_@@_formatted_lua_options_tl }}
7279 \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.

```

7280 \def\markdownPrepare{%

```

First, ensure that the `\markdownOptionCacheDir` directory exists.

```

7281     local lfs = require("lfs")
7282     local cacheDir = "\markdownOptionCacheDir"
7283     if not lfs.isdir(cacheDir) then
7284         assert(lfs.mkdir(cacheDir))
7285     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.

```

7286     local md = require("markdown")
7287     local convert = md.new(\markdownLuaOptions)
7288 }%

```

### 3.2.4 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.

```

7289 \ExplSyntaxOn
7290 \cs_new:Nn
7291   \@@_if_option:nTF
7292   {
7293     \@@_get_option_type:nN
7294     { #1 }
7295     \l_tmpa_tl
7296     \str_if_eq:NNF
7297     \l_tmpa_tl
7298     \c_@@_option_type_boolean_tl
7299     {
7300       \msg_error:nnxx
7301       { @@ }
7302       { expected-boolean-option }
7303       { #1 }
7304       { \l_tmpa_tl }
7305     }
7306     \@@_get_option_value:nN
7307     { #1 }
7308     \l_tmpa_tl
7309     \str_if_eq:NNTF
7310     \l_tmpa_tl
7311     \c_@@_option_value_true_tl
7312     { #2 }
7313     { #3 }
7314   }
7315 \msg_new:nnn
7316 { @@ }
7317 { expected-boolean-option }
7318 {
7319   Option~#1~has~type~#2,~
7320   but~a~boolean~was~expected.
7321 }
7322 \let\markdownIfOption=\@@_if_option:nTF
7323 \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.

```

7324 \csname newread\endcsname\markdownInputFileStream
7325 \csname newwrite\endcsname\markdownOutputFileStream

```

The `\markdownReadAndConvertTab` macro contains the tab character literal.

```

7326 \begingroup
7327 \catcode`\^^I=12%

```

```

7328 \gdef\markdownReadAndConvertTab{^^I}%
7329 \endgroup

```

The `\markdownReadAndConvert` macro is largely a rewrite of the  $\text{\LaTeX 2}\epsilon$  `\filecontents` macro to plain  $\text{\TeX}$ .

```

7330 \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.

```

7331 \catcode\^^M=13%
7332 \catcode\^^I=13%
7333 \catcode|=0%
7334 \catcode\\=12%
7335 |catcode%=14%
7336 |catcode|=12@
7337 |gdef|markdownReadAndConvert#1#2{@
7338   |begingroup@

```

If we are not reading markdown documents from the frozen cache, open the `\markdownOptionInputTempFileName` file for writing.

```

7339   |markdownIfOption{frozenCache}{@}{@
7340     |immediate|openout|markdownOutputFileStream@
7341     |markdownOptionInputTempFileName|relax@
7342     |markdownInfo{Buffering markdown input into the temporary @
7343       input file "|markdownOptionInputTempFileName" and scanning @
7344       for the closing token sequence "#1"}@
7345   }@

```

Locally change the category of the special plain  $\text{\TeX}$  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.

```

7346   |def|do##1{|catcode`##1=12}|dospecials@
7347   |catcode`|=12@
7348   |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 (`^^M`) are produced.

```

7349   |def|markdownReadAndConvertStripPercentSign##1{@
7350     |markdownIfOption{stripPercentSigns}{@
7351       |if##1%@
7352       |expandafter|expandafter|expandafter@
7353       |markdownReadAndConvertProcessLine@

```

```

7354         |else@
7355         |expandafter|expandafter|expandafter@
7356         |markdownReadAndConvertProcessLine@
7357         |expandafter|expandafter|expandafter##1@
7358     |fi@
7359 }{@
7360     |expandafter@
7361     |markdownReadAndConvertProcessLine@
7362     |expandafter##1@
7363 }@
7364 }@

```

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 (`^^M`) are produced.

```

7365 |def|markdownReadAndConvertProcessLine##1#1##2#1##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.

```

7366     |ifx|relax##3|relax@
7367     |markdownIfOption{frozenCache}{-}{@
7368     |immediate|write|markdownOutputFileStream{##1}@
7369     }@
7370 |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.

```

7371 |def^^M{@
7372     |markdownInfo{The ending token sequence was found}@
7373     |markdownIfOption{frozenCache}{-}{@
7374     |immediate|closeout|markdownOutputFileStream@
7375     }@
7376     |endgroup@
7377     |markdownInput{@
7378     |markdownOptionOutputDir@
7379     /|markdownOptionInputTempFileName@
7380     }@
7381     #2}@
7382 |fi@

```

Repeat with the next line.

```

7383     ^^M}@

```

Make the tab character active at expansion time and make it expand to a literal tab character.

```
7384 |catcode`|^I=13@
7385 |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.

```
7386 |catcode`|^M=13@
7387 |def^^M##1^^M{@
7388 |def^^M###1^^M{@
7389 |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
7390 ^^M}@
7391 ^^M}@
```

Reset the character categories back to the former state.

```
7392 |endgroup
```

The following two sections of the implementation have been deprecated and will be removed in Markdown 3.0.0. The code that corresponds to `\markdownMode` value of **3** will be the only implementation.

```
7393 \ExplSyntaxOn
7394 \int_compare:nT
7395 { \markdownMode = 3 }
7396 {
7397   \markdownInfo{Using~mode~3:~The~lt3luabridge~package}
7398   \file_input:n { lt3luabridge.tex }
7399   \cs_new:Npn
7400     \markdownLuaExecute
7401     { \luabridgeExecute }
7402 }
7403 \ExplSyntaxOff
```

### 3.2.5 Lua Shell Escape Bridge

The following  $\text{\TeX}$  code is intended for  $\text{\TeX}$  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 3.2.6 are meant to be indistinguishable to the remaining code.

The package assumes that although the user is not using the Lua $\text{\TeX}$  engine, their  $\text{\TeX}$  distribution contains it, and uses shell access to produce and execute Lua scripts using the  $\text{\TeX}$ Lua interpreter [1, Section 3.1.1].

```
7404 \ifnum\markdownMode<2\relax
7405 \ifnum\markdownMode=0\relax
7406   \markdownWarning{Using mode 0: Shell escape via write18
```

```

7407             (deprecated, to be removed in Markdown 3.0.0))%
7408 \else
7409   \markdownWarning{Using mode 1: Shell escape via os.execute
7410                   (deprecated, to be removed in Markdown 3.0.0))%
7411 \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.

```

7412 \ifx\pdfshellescape\undefined
7413   \ifx\shellescape\undefined
7414     \ifnum\markdownMode=0\relax
7415       \def\markdownExecuteShellEscape{1}%
7416     \else
7417       \def\markdownExecuteShellEscape{%
7418         \directlua{tex.sprint(status.shell_escape or "1")}}%
7419     \fi
7420   \else
7421     \let\markdownExecuteShellEscape\shellescape
7422   \fi
7423 \else
7424   \let\markdownExecuteShellEscape\pdfshellescape
7425 \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.

```

7426 \ifnum\markdownMode=0\relax
7427   \def\markdownExecuteDirect#1{\immediate\write18{#1}}%
7428 \else
7429   \def\markdownExecuteDirect#1{%
7430     \directlua{os.execute("\luaescapestring{#1}")}}%
7431 \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.

```

7432 \def\markdownExecute#1{%
7433   \ifnum\markdownExecuteShellEscape=1\relax
7434     \markdownExecuteDirect{#1}%
7435   \else
7436     \markdownError{I can not access the shell}{Either run the TeX
7437       compiler with the --shell-escape or the --enable-write18 flag,
7438       or set shell_escape=t in the texmf.cnf file}%

```

7439 \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.

7440 \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.

7441 \catcode`\|=0%

7442 \catcode`\|=12%

7443 |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.

```
7444 |immediate|openout|markdownOutputFileStream=%
7445 |markdownOptionHelperScriptFileName
7446 |markdownInfo{Writing a helper Lua script to the file
7447 "|markdownOptionHelperScriptFileName"}%
7448 |immediate|write|markdownOutputFileStream{%
7449   local ran_ok, error = pcall(function()
7450     local ran_ok, kpse = pcall(require, "kpse")
7451     if ran_ok then kpse.set_program_name("luatex") end
7452     #1
7453   end)
```

If there was an error, use the file `\markdownOptionErrorTempFileName` to store the error message.

```
7454   if not ran_ok then
7455     local file = io.open("%
7456     |markdownOptionOutputDir
7457     /|markdownOptionErrorTempFileName", "w")
7458     if file then
7459       file:write(error .. "\n")
7460       file:close()
7461     end
7462     print('\|markdownError{An error was encountered while executing
7463     Lua code}{For further clues, examine the file
7464     "|markdownOptionOutputDir
7465     /|markdownOptionErrorTempFileName"}')
7466   end}%
7467 |immediate|closeout|markdownOutputFileStream
```

Execute the generated `\markdownOptionHelperScriptFileName` Lua script using the TeX Lua binary and store the output in the `\markdownOptionOutputTempFileName` file.

7468 |markdownInfo{Executing a helper Lua script from the file



```

7469      "|markdownOptionHelperScriptFileName" and storing the result in the
7470      file "|markdownOptionOutputTempFileName"}%
7471      |markdownExecute{texlua "|markdownOptionOutputDir
7472      /|markdownOptionHelperScriptFileName" > %
7473      "|markdownOptionOutputDir
7474      /|markdownOptionOutputTempFileName"}%
  \input the generated \markdownOptionOutputTempFileName file.
7475      |input|markdownOptionOutputTempFileName|relax}%
7476 |endgroup

```

### 3.2.6 Direct Lua Access

The following  $\text{\TeX}$  code is intended for  $\text{\TeX}$  engines that provide direct access to Lua (Lua $\text{\TeX}$ ). The macro `\markdownLuaExecute` defined here and in Section 3.2.5 are meant to be indistinguishable to the remaining code. This corresponds to the `\markdownMode` value of 2.

```

7477 \fi
7478 \ifnum\markdownMode=2\relax
7479   \markdownWarning{Using mode 2: Direct Lua access
7480                   (deprecated, to be removed in Markdown 3.0.0)}%

```

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 3.2.5,

```

7481 \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.

```

7482   \catcode`\|=0%
7483   \catcode`\|=12%
7484   |gdef|markdownLuaExecute#1{%
7485     |directlua{%
7486       local function print(input)
7487         local output = {}
7488         for line in input:gmatch("[^\r\n]+") do
7489           table.insert(output, line)
7490         end
7491         tex.print(output)
7492       end
7493       #1
7494     }%
7495   }%
7496 |endgroup
7497 \fi

```

### 3.2.7 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.

7498 `\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.

7499 `\catcode`\|=0%`

7500 `\catcode`\|=12%`

7501 `|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.

7502 `|begingroup`

7503 `|catcode`\|=12`

If we are reading from the frozen cache, input it, expand the corresponding `\markdownFrozenCache<number>` macro, and increment `frozenCacheCounter`.

7504 `|markdownIfOption{frozenCache}{%`

7505 `|ifnum|markdownOptionFrozenCacheCounter=0|relax`

7506 `|markdownInfo{Reading frozen cache from`

7507 `"|markdownOptionFrozenCacheFileName"}%`

7508 `|input|markdownOptionFrozenCacheFileName|relax`

7509 `|fi`

7510 `|markdownInfo{Including markdown document number`

7511 `"|the|markdownOptionFrozenCacheCounter" from frozen cache}%`

7512 `|csname markdownFrozenCache|the|markdownOptionFrozenCacheCounter|endcsname`

7513 `|global|advance|markdownOptionFrozenCacheCounter by 1|relax`

7514 `}{%`

7515 `|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.

7516 `|openin|markdownInputFileStream#1`

7517 `|closein|markdownInputFileStream`

7518 `|markdownPrepareLuaOptions`

7519 `|markdownLuaExecute{%`

7520 `|markdownPrepare`

7521 `local file = assert(io.open("#1", "r"),`

7522 `[[could not open file "#1" for reading]])`

7523 `local input = assert(file:read("*a"))`

7524 `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.

```
7525      print(convert(input:gsub("\r\n?", "\n") .. "\n"))}%
```

In case we were finalizing the frozen cache, increment `frozenCacheCounter`.

```
7526      |global|advance|markdownOptionFrozenCacheCounter by 1|relax
7527    }%
7528  |endgroup
7529 }%
7530 |endgroup
```

### 3.3 L<sup>A</sup>T<sub>E</sub>X Implementation

The L<sup>A</sup>T<sub>E</sub>X implemenation makes use of the fact that, apart from some subtle differences, L<sup>A</sup>T<sub>E</sub>X implements the majority of the plain T<sub>E</sub>X format [10, Section 9]. As a consequence, we can directly reuse the existing plain T<sub>E</sub>X implementation.

```
7531 \def\markdownVersionSpace{ }%
7532 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
7533 \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.

```
7534 \ExplSyntaxOn
7535 \@@_latex_define_renderers:
7536 \@@_latex_define_renderer_prototypes:
7537 \ExplSyntaxOff
```

#### 3.3.1 Logging Facilities

The L<sup>A</sup>T<sub>E</sub>X implementation redefines the plain T<sub>E</sub>X logging macros (see Section 3.2.1) to use the L<sup>A</sup>T<sub>E</sub>X `\PackageInfo`, `\PackageWarning`, and `\PackageError` macros.

#### 3.3.2 Typesetting Markdown

The `\markdownInputPlainTeX` macro is used to store the original plain T<sub>E</sub>X implementation of the `\markdownInput` macro. The `\markdownInput` is then redefined to accept an optional argument with options recognized by the L<sup>A</sup>T<sub>E</sub>X interface (see Section 2.3.2).

```
7538 \let\markdownInputPlainTeX\markdownInput
7539 \renewcommand\markdownInput[2][]{%
7540   \begingroup
7541     \markdownSetup{#1}%
7542     \markdownInputPlainTeX{#2}%
7543   \endgroup}%
```

The `markdown`, and `markdown*` L<sup>A</sup>T<sub>E</sub>X environments are implemented using the `\markdownReadAndConvert` macro.

```

7544 \renewenvironment{markdown}{%
7545   \markdownReadAndConvert@markdown{}}{%
7546   \markdownEnd}%
7547 \renewenvironment{markdown*}[1]{%
7548   \markdownSetup{#1}%
7549   \markdownReadAndConvert@markdown*}{%
7550   \markdownEnd}%
7551 \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*.

```

7552 \catcode`\|=0\catcode`\<=1\catcode`\>=2%
7553 \catcode`\|=12\catcode`\{=12\catcode`\}=12%
7554 |gdef|markdownReadAndConvert@markdown#1<%
7555   |markdownReadAndConvert<\end{markdown#1}>%
7556                               <|end<markdown#1>>>%
7557 |endgroup

```

**3.3.2.1 L<sup>A</sup>T<sub>E</sub>X Themes** This section implements the theme-loading mechanism and the example themes provided with the Markdown package.

```

7558 \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.

```

7559 \newcommand\markdownLaTeXThemeName{}
7560 \seq_new:N \g_@@_latex_themes_seq
7561 \seq_put_right:NV
7562   \g_@@_latex_themes_seq
7563   \markdownLaTeXThemeName
7564 \newcommand\markdownLaTeXThemeLoad[2]{
7565   \def\@tempa{%
7566     \def\markdownLaTeXThemeName{#2}
7567     \seq_put_right:NV
7568       \g_@@_latex_themes_seq
7569       \markdownLaTeXThemeName
7570     \RequirePackage{#1}
7571     \seq_pop_right:NN
7572       \g_@@_latex_themes_seq
7573     \l_tmpa_tl
7574     \seq_get_right:NN
7575       \g_@@_latex_themes_seq
7576     \l_tmpa_tl

```

```

7577     \exp_args:NNV
7578     \def
7579     \markdownLaTeXThemeName
7580     \l_tmpa_tl}
7581 \ifmarkdownLaTeXLoaded
7582     \@tempa
7583 \else
7584     \exp_args:No
7585     \AtEndOfPackage
7586     { \@tempa }
7587 \fi}
7588 \ExplSyntaxOff

```

The `witiko/dot` theme enables the `fencedCode` Lua option:

```

7589 \markdownSetup{fencedCode}%

```

We load the `ifthen` and `grffile` packages, see also Section 1.1.3:

```

7590 \RequirePackage{ifthen,grffile}

```

We store the previous definition of the fenced code token renderer prototype:

```

7591 \let\markdown@witiko@dot@oldRendererInputFencedCodePrototype
7592 \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:

```

7593 \renewcommand\markdownRendererInputFencedCode[2]{%
7594     \def\next##1 ##2\relax{%
7595         \ifthenelse{\equal{##1}{dot}}{%
7596             \markdownIfOption{frozenCache}{}{%
7597                 \immediate\write18{%
7598                     if ! test -e #1.pdf.source || ! diff #1 #1.pdf.source;
7599                     then
7600                         dot -Tpdf -o #1.pdf #1;
7601                         cp #1 #1.pdf.source;
7602                     fi}}%

```

We include the typeset image using the image token renderer:

```

7603     \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:

```

7604     }{%
7605         \markdown@witiko@dot@oldRendererInputFencedCodePrototype{#1}{#2}%
7606     }%
7607 }%
7608 \next#2 \relax}%

```

The `witiko/graphicx/http` theme stores the previous definition of the image token renderer prototype:

```
7609 \let\markdown@witiko@graphicx@http@oldRendererImagePrototype
7610 \markdownRendererImagePrototype
```

We load the catchfile and grffile packages, see also Section 1.1.3:

```
7611 \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.

```
7612 \newcount\markdown@witiko@graphicx@http@counter
7613 \markdown@witiko@graphicx@http@counter=0
7614 \newcommand\markdown@witiko@graphicx@http@filename{%
7615   \markdownOptionCacheDir/witiko_graphicx_http%
7616   .\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.

```
7617 \newcommand\markdown@witiko@graphicx@http@download[2]{%
7618   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:

```
7619 \begingroup
7620 \catcode`\%=12
7621 \catcode`\^A=14
```

We redefine the image token renderer prototype, so that it tries to download an online image.

```
7622 \global\def\markdownRendererImagePrototype#1#2#3#4{^^A
7623   \begingroup
7624   \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:

```
7625   \markdownIfOption{frozenCache}{}{^^A
7626   \immediate\write18{^^A
7627     mkdir -p "\markdownOptionCacheDir";
7628     if printf '%s' "#3" | grep -q -E '^https?:';
7629     then
```

The image will be downloaded to the pathname `\markdownOptionCacheDir/⟨the MD5 digest of the image URL⟩.⟨the suffix of the image URL⟩`:

```
7630     OUTPUT_PREFIX="\markdownOptionCacheDir";
```

```

7631     OUTPUT_BODY="$(printf '%s' '#3' | md5sum | cut -d ' ' -f1)";
7632     OUTPUT_SUFFIX="$(printf '%s' '#3' | sed 's/.*[.]/')";
7633     OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";

```

The image will be downloaded only if it has not already been downloaded:

```

7634     if ! [ -e "$OUTPUT" ];
7635     then
7636         \markdown@witiko@graphicx@http@download{'#3'}{"$OUTPUT"};
7637         printf '%s' "$OUTPUT" > "\filename";
7638     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:

```

7639     else
7640         printf '%s' '#3' > "\filename";
7641     fi}}^^A

```

We load the pathname of the downloaded image and we typeset the image using the previous definition of the image renderer prototype:

```

7642     \CatchFileDef{\filename}{\filename}{\newlinechar=-1}^^A
7643     \markdown@witiko@graphicx@http@oldRendererImagePrototype^^A
7644     {#1}{#2}{\filename}{#4}^^A
7645     \endgroup
7646     \global\advance\markdown@witiko@graphicx@http@counter by 1\relax}^^A
7647 \endgroup

```

The [witiko/tilde](#) theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```

7648 \renewcommand\markdownRendererTildePrototype{~}%

```

### 3.3.3 Options

The supplied package options are processed using the [\markdownSetup](#) macro.

```

7649 \DeclareOption*{%
7650     \expandafter\markdownSetup\expandafter{\CurrentOption}}%
7651 \ProcessOptions\relax

```

After processing the options, activate the [jekyllDataRenderes](#), [renderers](#), [rendererPrototypes](#), and [code](#) keys.

```

7652 \ExplSyntaxOn
7653 \keys_define:nn
7654 { markdown/latex-options }
7655 {
7656     renderers .code:n = {
7657         \keys_set:nn
7658         { markdown/latex-options/renderers }
7659         { #1 }
7660     },

```

```

7661     rendererPrototypes .code:n = {
7662         \keys_set:nn
7663         { markdown/latex-options/renderer-prototypes }
7664         { #1 }
7665     },

```

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.

```

7666     code .code:n = { #1 },

```

The `jeekyllDataRenderers` key can be used as a syntactic sugar for setting the `markdown/jeekyllData` key-values (see Section 2.2.4.1) without using the `expl3` language.

```

7667     jeekyllDataRenderers .code:n = {
7668         \keys_set:nn
7669         { markdown/latex-options/jeekyll-data-renderers }
7670         { #1 }
7671     },
7672 }
7673 \keys_define:nn
7674 { markdown/latex-options/jeekyll-data-renderers }
7675 {
7676     unknown .code:n = {
7677         \tl_set_eq:NN
7678         \l_tmpa_tl
7679         \l_keys_key_str
7680         \tl_put_right:Nn
7681         \l_tmpa_tl
7682         {
7683             .code:n = { #1 }
7684         }
7685         \keys_define:nV
7686         { markdown/jeekyllData }
7687         \l_tmpa_tl
7688     }
7689 }
7690 \cs_generate_variant:Nn
7691 \keys_define:nn
7692 { nV }
7693 \ExplSyntaxOff

```

### 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.

```

7694 \markdownIfOption{plain}{\iffalse}{\iftrue}

```



If the `tightLists` Lua option is disabled or the current document class is beamer, do not load the paralist package.

```
7695 \markdownIfOption{tightLists}{
7696   \@ifclassloaded{beamer}{}{\RequirePackage{paralist}}}%
7697 }{}
```

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.

```
7698 \ExplSyntaxOn
7699 \@ifpackageloaded{paralist}{
7700   \tl_new:N
7701     \l_@@_latex_fancy_list_item_label_number_style_tl
7702   \tl_new:N
7703     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
7704   \cs_new:Nn
7705     \@@_latex_fancy_list_item_label_number:nn
7706     {
7707       \str_case:nn
7708         { #1 }
7709         {
7710           { Decimal } { #2 }
7711           { LowerRoman } { \int_to_roman:n { #2 } }
7712           { UpperRoman } { \int_to_Roman:n { #2 } }
7713           { LowerAlpha } { \int_to_alph:n { #2 } }
7714           { UpperAlpha } { \int_to_alph:n { #2 } }
7715         }
7716     }
7717   \cs_new:Nn
7718     \@@_latex_fancy_list_item_label_delimiter:n
7719     {
7720       \str_case:nn
7721         { #1 }
7722         {
7723           { Default } { . }
7724           { OneParen } { ) }
7725           { Period } { - }
7726         }
7727     }
7728   \cs_new:Nn
7729     \@@_latex_fancy_list_item_label:nnn
7730     {
7731       \@@_latex_fancy_list_item_label_delimiter:n
7732         { #2 }
7733       \@@_latex_fancy_list_item_label_number:nn
7734         { #1 }
7735         { #3 }
```

```

7736     }
7737 \cs_new:Nn
7738   \@@_latex_paralist_style:nn
7739   {
7740     \str_case:nn
7741       { #1 }
7742       {
7743         { Decimal } { 1 }
7744         { LowerRoman } { i }
7745         { UpperRoman } { I }
7746         { LowerAlpha } { a }
7747         { UpperAlpha } { A }
7748       }
7749     \@@_latex_fancy_list_item_label_delimiter:n
7750       { #2 }
7751   }
7752 \markdownSetup{rendererPrototypes={
7753   ulBeginTight = {\begin{compactitem}},
7754   ulEndTight = {\end{compactitem}},
7755   fancyOlBegin = {
7756     \group_begin:
7757     \tl_set:Nn
7758       \l_@@_latex_fancy_list_item_label_number_style_tl
7759       { #1 }
7760     \tl_set:Nn
7761       \l_@@_latex_fancy_list_item_label_delimiter_style_tl
7762       { #2 }
7763     \begin{enumerate}[ \@@_latex_paralist_style:nn { #1 } { #2 } ]
7764   },
7765   fancyOlEnd = {
7766     \end{enumerate}
7767     \group_end:
7768   },
7769   olBeginTight = {\begin{compactenum}},
7770   olEndTight = {\end{compactenum}},
7771   fancyOlBeginTight = {
7772     \group_begin:
7773     \tl_set:Nn
7774       \l_@@_latex_fancy_list_item_label_number_style_tl
7775       { #1 }
7776     \tl_set:Nn
7777       \l_@@_latex_fancy_list_item_label_delimiter_style_tl
7778       { #2 }
7779     \begin{compactenum}[ \@@_latex_paralist_style:nn { #1 } { #2 } ]
7780   },
7781   fancyOlEndTight = {
7782     \end{compactenum}

```

```

7783     \group_end:
7784 },
7785 fancyO1ItemWithNumber = {
7786     \markdownRendererO1ItemWithNumber
7787     {
7788         \@@_latex_fancy_list_item_label:VVn
7789         \l_@@_latex_fancy_list_item_label_number_style_tl
7790         \l_@@_latex_fancy_list_item_label_delimiter_style_tl
7791         { #1 }
7792     }
7793 },
7794 dlBeginTight = {\begin{compactdesc}},
7795 dlEndTight = {\end{compactdesc}}}}
7796 \cs_generate_variant:Nn
7797     \@@_latex_fancy_list_item_label:nnn
7798     { VVn }
7799 }{
7800     \markdownSetup{rendererPrototypes={
7801         ulBeginTight = {\markdownRendererUlBegin},
7802         ulEndTight = {\markdownRendererUlEnd},
7803         fancyO1Begin = {\markdownRendererO1Begin},
7804         fancyO1End = {\markdownRendererO1End},
7805         olBeginTight = {\markdownRendererO1Begin},
7806         olEndTight = {\markdownRendererO1End},
7807         fancyO1BeginTight = {\markdownRendererO1Begin},
7808         fancyO1EndTight = {\markdownRendererO1End},
7809         dlBeginTight = {\markdownRendererDlBegin},
7810         dlEndTight = {\markdownRendererDlEnd}}}
7811 }
7812 \ExplSyntaxOff
7813 \RequirePackage{amsmath,ifthen}

    Unless the unicode-math package has been loaded, load the amssymb package
    with symbols to be used for tickboxes.

7814 \ifpackageloaded{unicode-math}{
7815     \markdownSetup{rendererPrototypes={
7816         untickedBox = {\$ \mdlgwhtsquare$},
7817     }}
7818 }{
7819     \RequirePackage{amssymb}
7820     \markdownSetup{rendererPrototypes={
7821         untickedBox = {\$ \square$},
7822     }}
7823 }
7824 \RequirePackage{csvsimple}
7825 \RequirePackage{fancyvrb}
7826 \RequirePackage{graphicx}

```

```

7827 \markdownSetup{rendererPrototypes={
7828   lineBreak = {\},
7829   leftBrace = {\textbraceleft},
7830   rightBrace = {\textbraceright},
7831   dollarSign = {\textdollar},
7832   underscore = {\textunderscore},
7833   circumflex = {\textasciicircum},
7834   backslash = {\textbackslash},
7835   tilde = {\textasciitilde},
7836   pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by  $\text{\TeX}$  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.

```

7837   codeSpan = {%
7838     \ifmmode
7839       \text{#1}%
7840     \else
7841       \texttt{#1}%
7842     \fi
7843   },
7844   contentBlock = {%
7845     \ifthenelse{\equal{#1}{csv}}{%
7846       \begin{table}%
7847         \begin{center}%
7848           \csvautotabular{#3}%
7849         \end{center}
7850       \ifx\empty#4\empty\else
7851         \caption{#4}%
7852       \fi
7853     \end{table}%
7854   }{%
7855     \ifthenelse{\equal{#1}{tex}}{%
7856       \catcode`\%=14\relax
7857       \input #3\relax
7858       \catcode`\%=12\relax
7859     }{%
7860       \markdownInput{#3}%
7861     }%
7862   }%

```

---

<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.

```

7863 },
7864 image = {%
7865     \begin{figure}%
7866     \begin{center}%
7867     \includegraphics{#3}%
7868     \end{center}%
7869     \ifx\empty#4\empty\else
7870     \caption{#4}%
7871     \fi
7872     \end{figure}},
7873 ulBegin = {\begin{itemize}},
7874 ulEnd = {\end{itemize}},
7875 olBegin = {\begin{enumerate}},
7876 olItem = {\item{}},
7877 olItemWithNumber = {\item[#1.]},
7878 olEnd = {\end{enumerate}},
7879 dlBegin = {\begin{description}},
7880 dlItem = {\item[#1]},
7881 dlEnd = {\end{description}},
7882 emphasis = {\emph{#1}},
7883 tickedTextBox = {\$\boxtimes$},
7884 halfTickedTextBox = {\$\boxdot$},

```

If identifier attributes appear at the beginning of a section, we make the next heading produce the `\label` macro.

```

7885 headerAttributeContextBegin = {
7886     \markdownSetup{
7887         rendererPrototypes = {
7888             attributeIdentifier = {%
7889                 \begingroup
7890                 \def\next####1{%
7891                     \def####1#####1{%
7892                         \endgroup
7893                         #####1{#####1}%
7894                         \label{##1}%
7895                     }%
7896                 }%
7897                 \next\markdownRendererHeadingOne
7898                 \next\markdownRendererHeadingTwo
7899                 \next\markdownRendererHeadingThree
7900                 \next\markdownRendererHeadingFour
7901                 \next\markdownRendererHeadingFive
7902                 \next\markdownRendererHeadingSix
7903             },
7904         },
7905     }%
7906 },

```

```

7907 superscript = {\textsuperscript{#1}},
7908 subscript = {\textsubscript{#1}},
7909 blockQuoteBegin = {\begin{quotation}},
7910 blockQuoteEnd = {\end{quotation}},
7911 inputVerbatim = {\VerbatimInput{#1}},
7912 inputFencedCode = {%
7913   \ifx\relax#2\relax
7914     \VerbatimInput{#1}%
7915   \else
7916     \@ifundefined{minted@code}{%
7917       \@ifundefined{lst@version}{%
7918         \markdownRendererInputFencedCode{#1}{}%

```

When the listings package is loaded, use it for syntax highlighting.

```

7919   }{%
7920     \lstinputlisting[language=#2]{#1}%
7921   }%

```

When the minted package is loaded, use it for syntax highlighting. The minted package is preferred over listings.

```

7922   }{%
7923     \inputminted{#2}{#1}%
7924   }%
7925   \fi},
7926 horizontalRule = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
7927 footnote = {\footnote{#1}}}

```

Support the nesting of strong emphasis.

```

7928 \ExplSyntaxOn
7929 \def\markdownLATEXStrongEmphasis#1{%
7930   \str_if_in:NnTF
7931     \f@series
7932     { b }
7933     { \textnormal{#1} }
7934     { \textbf{#1} }
7935 }
7936 \ExplSyntaxOff
7937 \markdownSetup{rendererPrototypes={strongEmphasis={%
7938   \protect\markdownLATEXStrongEmphasis{#1}}}}

```

Support L<sup>A</sup>T<sub>E</sub>X document classes that do not provide chapters.

```

7939 \@ifundefined{chapter}{%
7940   \markdownSetup{rendererPrototypes = {
7941     headingOne = {\section{#1}},
7942     headingTwo = {\subsection{#1}},
7943     headingThree = {\subsubsection{#1}},
7944     headingFour = {\paragraph{#1}\leavevmode},
7945     headingFive = {\subparagraph{#1}\leavevmode}}}
7946 }{%

```

```

7947 \markdownSetup{rendererPrototypes = {
7948     headingOne = {\chapter{#1}},
7949     headingTwo = {\section{#1}},
7950     headingThree = {\subsection{#1}},
7951     headingFour = {\subsubsection{#1}},
7952     headingFive = {\paragraph{#1}\leavevmode},
7953     headingSix = {\subparagraph{#1}\leavevmode}}}
7954 }%

```

**3.3.4.1 Tickboxes** If the `taskLists` option is enabled, we will hide bullets in unordered list items with tickboxes.

```

7955 \markdownSetup{
7956     rendererPrototypes = {
7957         ulItem = {%
7958             \futurelet\markdownLaTeXCheckbox\markdownLaTeXUListItem
7959         },
7960     },
7961 }
7962 \def\markdownLaTeXUListItem{%
7963     \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
7964         \item[\markdownLaTeXCheckbox]%
7965         \expandafter\@gobble
7966     \else
7967         \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
7968             \item[\markdownLaTeXCheckbox]%
7969             \expandafter\expandafter\expandafter\@gobble
7970         \else
7971             \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
7972                 \item[\markdownLaTeXCheckbox]%
7973                 \expandafter\expandafter\expandafter\expandafter
7974                 \expandafter\expandafter\expandafter\@gobble
7975             \else
7976                 \item{}%
7977             \fi
7978         \fi
7979     \fi
7980 }

```

**3.3.4.2 HTML elements** If the `html` option is enabled and we are using  $\text{\TeX}4\text{ht}$ <sup>9</sup>, we will pass HTML elements to the output HTML document unchanged.

```

7981 \@ifundefined{HCode}{}{
7982     \markdownSetup{
7983         rendererPrototypes = {
7984             inlineHtmlTag = {%

```

---

<sup>9</sup>See <https://tug.org/tex4ht/>.

```

7985     \ifvmode
7986     \IgnorePar
7987     \EndP
7988     \fi
7989     \HCode{#1}%
7990   },
7991   inputBlockHtmlElement = {%
7992     \ifvmode
7993     \IgnorePar
7994     \fi
7995     \EndP
7996     \special{t4ht*<#1}%
7997     \par
7998     \ShowPar
7999   },
8000 },
8001 }
8002 }

```

**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.

```

8003 \newcount\markdownLaTeXCitationsCounter
8004
8005 % Basic implementation
8006 \RequirePackage{gobble}
8007 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
8008   \advance\markdownLaTeXCitationsCounter by 1\relax
8009   \ifx\relax#4\relax
8010     \ifx\relax#5\relax
8011       \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8012         \cite{#1#2#6}% Without prenotes and postnotes, just accumulate cites
8013         \expandafter\expandafter\expandafter
8014         \expandafter\expandafter\expandafter\expandafter
8015         \@gobblethree
8016       \fi
8017     \else% Before a postnote (#5), dump the accumulator
8018       \ifx\relax#1\relax\else
8019         \cite{#1}%
8020       \fi
8021       \cite[#5]{#6}%
8022     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8023     \else
8024       \expandafter\expandafter\expandafter
8025       \expandafter\expandafter\expandafter\expandafter

```



```

8026         \expandafter\expandafter\expandafter
8027         \expandafter\expandafter\expandafter\expandafter
8028         \markdownLaTeXBasicCitations
8029     \fi
8030     \expandafter\expandafter\expandafter
8031     \expandafter\expandafter\expandafter\expandafter{%
8032     \expandafter\expandafter\expandafter
8033     \expandafter\expandafter\expandafter\expandafter}%
8034     \expandafter\expandafter\expandafter
8035     \expandafter\expandafter\expandafter\expandafter{%
8036     \expandafter\expandafter\expandafter
8037     \expandafter\expandafter\expandafter\expandafter}%
8038     \expandafter\expandafter\expandafter
8039     \@gobblethree
8040 \fi
8041 \else% Before a prenote (#4), dump the accumulator
8042     \ifx\relax#1\relax\else
8043         \cite{#1}%
8044     \fi
8045     \ifnum\markdownLaTeXCitationsCounter>1\relax
8046         \space % Insert a space before the prenote in later citations
8047     \fi
8048     #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
8049     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8050     \else
8051         \expandafter\expandafter\expandafter
8052         \expandafter\expandafter\expandafter\expandafter
8053         \markdownLaTeXBasicCitations
8054     \fi
8055     \expandafter\expandafter\expandafter{%
8056     \expandafter\expandafter\expandafter}%
8057     \expandafter\expandafter\expandafter{%
8058     \expandafter\expandafter\expandafter}%
8059     \expandafter
8060     \@gobblethree
8061 \fi\markdownLaTeXBasicCitations{#1#2#6},}
8062 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
8063
8064 % Natbib implementation
8065 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
8066     \advance\markdownLaTeXCitationsCounter by 1\relax
8067     \ifx\relax#3\relax
8068         \ifx\relax#4\relax
8069             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8070                 \citep{#1,#5}% Without prenotes and postnotes, just accumulate cites
8071             \expandafter\expandafter\expandafter
8072             \expandafter\expandafter\expandafter\expandafter

```

```

8073     \@gobbletwo
8074     \fi
8075 \else% Before a postnote (#4), dump the accumulator
8076     \ifx\relax#1\relax\else
8077         \citep{#1}%
8078     \fi
8079     \citep[] [#4]{#5}%
8080     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8081     \else
8082         \expandafter\expandafter\expandafter
8083         \expandafter\expandafter\expandafter\expandafter
8084         \expandafter\expandafter\expandafter
8085         \expandafter\expandafter\expandafter\expandafter
8086         \markdownLaTeXNatbibCitations
8087     \fi
8088     \expandafter\expandafter\expandafter
8089     \expandafter\expandafter\expandafter\expandafter{%
8090     \expandafter\expandafter\expandafter
8091     \expandafter\expandafter\expandafter\expandafter}%
8092     \expandafter\expandafter\expandafter
8093     \@gobbletwo
8094     \fi
8095 \else% Before a prenote (#3), dump the accumulator
8096     \ifx\relax#1\relax\relax\else
8097         \citep{#1}%
8098     \fi
8099     \citep[#3] [#4]{#5}%
8100     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8101     \else
8102         \expandafter\expandafter\expandafter
8103         \expandafter\expandafter\expandafter\expandafter
8104         \markdownLaTeXNatbibCitations
8105     \fi
8106     \expandafter\expandafter\expandafter{%
8107     \expandafter\expandafter\expandafter}%
8108     \expandafter
8109     \@gobbletwo
8110     \fi\markdownLaTeXNatbibCitations{#1,#5}}
8111 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
8112     \advance\markdownLaTeXCitationsCounter by 1\relax
8113     \ifx\relax#3\relax
8114         \ifx\relax#4\relax
8115             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8116                 \citet{#1,#5}% Without prenotes and postnotes, just accumulate cites
8117             \expandafter\expandafter\expandafter
8118             \expandafter\expandafter\expandafter\expandafter
8119             \@gobbletwo

```

```

8120     \fi
8121 \else% After a prenote or a postnote, dump the accumulator
8122     \ifx\relax#1\relax\else
8123         \citet{#1}%
8124     \fi
8125     , \citet[#3][#4]{#5}%
8126     \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
8127     ,
8128     \else
8129         \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
8130         ,
8131     \fi
8132     \fi
8133     \expandafter\expandafter\expandafter
8134     \expandafter\expandafter\expandafter\expandafter
8135     \markdownLaTeXNatbibTextCitations
8136     \expandafter\expandafter\expandafter
8137     \expandafter\expandafter\expandafter\expandafter{%
8138     \expandafter\expandafter\expandafter
8139     \expandafter\expandafter\expandafter\expandafter}%
8140     \expandafter\expandafter\expandafter
8141     \@gobbletwo
8142 \fi
8143 \else% After a prenote or a postnote, dump the accumulator
8144     \ifx\relax#1\relax\relax\else
8145         \citet{#1}%
8146     \fi
8147     , \citet[#3][#4]{#5}%
8148     \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
8149     ,
8150     \else
8151         \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
8152         ,
8153     \fi
8154     \fi
8155     \expandafter\expandafter\expandafter
8156     \markdownLaTeXNatbibTextCitations
8157     \expandafter\expandafter\expandafter{%
8158     \expandafter\expandafter\expandafter}%
8159     \expandafter
8160     \@gobbletwo
8161 \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
8162
8163 % BibLaTeX implementation
8164 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
8165     \advance\markdownLaTeXCitationsCounter by 1\relax
8166     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax

```

```

8167 \autocites#1[#3][#4]{#5}%
8168 \expandafter\@gobbletwo
8169 \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}
8170 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
8171 \advance\markdownLaTeXCitationsCounter by 1\relax
8172 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8173 \textcites#1[#3][#4]{#5}%
8174 \expandafter\@gobbletwo
8175 \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}
8176
8177 \markdownSetup{rendererPrototypes = {
8178 cite = {%
8179 \markdownLaTeXCitationsCounter=1%
8180 \def\markdownLaTeXCitationsTotal{#1}%
8181 \@ifundefined{autocites}{%
8182 \ifundefined{citep}{%
8183 \expandafter\expandafter\expandafter
8184 \markdownLaTeXBasicCitations
8185 \expandafter\expandafter\expandafter{%
8186 \expandafter\expandafter\expandafter}%
8187 \expandafter\expandafter\expandafter{%
8188 \expandafter\expandafter\expandafter}%
8189 }{%
8190 \expandafter\expandafter\expandafter
8191 \markdownLaTeXNatbibCitations
8192 \expandafter\expandafter\expandafter{%
8193 \expandafter\expandafter\expandafter}%
8194 }%
8195 }{%
8196 \expandafter\expandafter\expandafter
8197 \markdownLaTeXBibLaTeXCitations
8198 \expandafter{\expandafter}%
8199 }},
8200 textCite = {%
8201 \markdownLaTeXCitationsCounter=1%
8202 \def\markdownLaTeXCitationsTotal{#1}%
8203 \@ifundefined{autocites}{%
8204 \ifundefined{citep}{%
8205 \expandafter\expandafter\expandafter
8206 \markdownLaTeXBasicTextCitations
8207 \expandafter\expandafter\expandafter{%
8208 \expandafter\expandafter\expandafter}%
8209 \expandafter\expandafter\expandafter{%
8210 \expandafter\expandafter\expandafter}%
8211 }{%
8212 \expandafter\expandafter\expandafter
8213 \markdownLaTeXNatbibTextCitations

```

```

8214         \expandafter\expandafter\expandafter{%
8215         \expandafter\expandafter\expandafter}%
8216     }%
8217 }{%
8218     \expandafter\expandafter\expandafter
8219     \markdownLaTeXBibLaTeXTextCitations
8220     \expandafter{\expandafter}%
8221 }}}

```

**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.

```

8222 \RequirePackage{url}
8223 \RequirePackage{expl3}
8224 \ExplSyntaxOn
8225 \def\markdownRendererLinkPrototype{
8226     \begingroup
8227     \catcode`\#=12
8228     \def\next##1##2##3##4{
8229         \endgroup

```

If the label and the fully-escaped URI are equivalent and the title is empty, assume that the link is an autolink. Otherwise, assume that the link is either direct or indirect.

```

8230     \tl_set:Nn \l_tmpa_tl { ##1 }
8231     \tl_set:Nn \l_tmpb_tl { ##2 }
8232     \bool_set:Nn
8233         \l_tmpa_bool
8234     {
8235         \tl_if_eq_p:NN
8236             \l_tmpa_tl
8237             \l_tmpb_tl
8238     }
8239     \tl_set:Nn \l_tmpa_tl { ##4 }
8240     \bool_set:Nn
8241         \l_tmpb_bool
8242     {
8243         \tl_if_empty_p:N
8244             \l_tmpa_tl
8245     }
8246     \bool_if:nTF
8247     {
8248         \l_tmpa_bool && \l_tmpb_bool
8249     }
8250     {
8251         \markdownLaTeXRendererAutolink { ##2 } { ##3 }

```

```

8252     }{
8253     \markdownLaTeXRendererDirectOrIndirectLink { ##1 } { ##2 } { ##3 } { ##4 }
8254     }
8255   }
8256   \next
8257 }
8258 \def\markdownLaTeXRendererAutolink#1#2{%
  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.

8259   \tl_set:Nn
8260     \l_tmpa_tl
8261     { #2 }
8262   \tl_trim_spaces:N
8263     \l_tmpa_tl
8264   \tl_set:Nx
8265     \l_tmpb_tl
8266     {
8267       \tl_range:Nnn
8268         \l_tmpa_tl
8269         { 1 }
8270         { 1 }
8271     }
8272   \str_if_eq:NNTF
8273     \l_tmpb_tl
8274     \c_hash_str
8275     {
8276       \tl_set:Nx
8277         \l_tmpb_tl
8278         {
8279           \tl_range:Nnn
8280             \l_tmpa_tl
8281             { 2 }
8282             { -1 }
8283         }
8284       \exp_args:NV
8285         \ref
8286         \l_tmpb_tl
8287     }{
8288       \url { #2 }
8289     }
8290 }
8291 \ExplSyntaxOff
8292 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
8293   #1\footnote{\ifx\empty#4\empty\else#4: \fi\url{#3}}}
```

**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.

```

8294 \newcount\markdownLaTeXRowCount
8295 \newcount\markdownLaTeXRowTotal
8296 \newcount\markdownLaTeXColumnCounter
8297 \newcount\markdownLaTeXColumnTotal
8298 \newtoks\markdownLaTeXTable
8299 \newtoks\markdownLaTeXTableAlignment
8300 \newtoks\markdownLaTeXTableEnd
8301 \AtBeginDocument{%
8302   \ifpackageloaded{booktabs}{%
8303     \def\markdownLaTeXTopRule{\toprule}%
8304     \def\markdownLaTeXMidRule{\midrule}%
8305     \def\markdownLaTeXBottomRule{\bottomrule}%
8306   }{%
8307     \def\markdownLaTeXTopRule{\hline}%
8308     \def\markdownLaTeXMidRule{\hline}%
8309     \def\markdownLaTeXBottomRule{\hline}%
8310   }%
8311 }
8312 \markdownSetup{rendererPrototypes={
8313   table = {%
8314     \markdownLaTeXTable={}%
8315     \markdownLaTeXTableAlignment={}%
8316     \markdownLaTeXTableEnd={%
8317       \markdownLaTeXBottomRule
8318     \end{tabular}}}%
8319   \ifx\empty#1\empty\else
8320     \addto@hook\markdownLaTeXTable{%
8321       \begin{table}
8322       \centering}%
8323     \addto@hook\markdownLaTeXTableEnd{%
8324       \caption{#1}
8325     \end{table}}}%
8326   \fi
8327   \addto@hook\markdownLaTeXTable{\begin{tabular}}}%
8328   \markdownLaTeXRowCount=0%
8329   \markdownLaTeXRowTotal=#2%
8330   \markdownLaTeXColumnTotal=#3%
8331   \markdownLaTeXRenderTableRow
8332 }
8333 }}
8334 \def\markdownLaTeXRenderTableRow#1{%
8335   \markdownLaTeXColumnCounter=0%
8336   \ifnum\markdownLaTeXRowCount=0\relax
8337     \markdownLaTeXReadAlignments#1%
8338     \markdownLaTeXTable=\expandafter\expandafter\expandafter{%

```

```

8339     \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
8340     \the\markdownLaTeXTableAlignment}}}%
8341     \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
8342 \else
8343     \markdownLaTeXRenderTableCell#1%
8344 \fi
8345 \ifnum\markdownLaTeXRowCount=1\relax
8346     \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
8347 \fi
8348 \advance\markdownLaTeXRowCount by 1\relax
8349 \ifnum\markdownLaTeXRowCount>\markdownLaTeXRowTotal\relax
8350     \the\markdownLaTeXTable
8351     \the\markdownLaTeXTableEnd
8352     \expandafter\@gobble
8353 \fi\markdownLaTeXRenderTableRow}
8354 \def\markdownLaTeXReadAlignments#1{%
8355     \advance\markdownLaTeXColumnCounter by 1\relax
8356     \if#1d%
8357         \addto@hook\markdownLaTeXTableAlignment{1}%
8358     \else
8359         \addto@hook\markdownLaTeXTableAlignment{#1}%
8360     \fi
8361     \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
8362         \expandafter\@gobble
8363     \fi\markdownLaTeXReadAlignments}
8364 \def\markdownLaTeXRenderTableCell#1{%
8365     \advance\markdownLaTeXColumnCounter by 1\relax
8366     \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
8367         \addto@hook\markdownLaTeXTable{#1&}%
8368     \else
8369         \addto@hook\markdownLaTeXTable{#1\\}%
8370     \expandafter\@gobble
8371     \fi\markdownLaTeXRenderTableCell}
8372 \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.

```

8373 \ExplSyntaxOn
8374 \keys_define:nn
8375 { markdown/jekyllData }
8376 {
8377     author .code:n = { \author{#1} },
8378     date .code:n = { \date{#1} },
8379     title .code:n = { \title{#1} },
8380 }

```



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.

```

8381 % TODO: Remove the command definition in TeX Live 2021.
8382 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
8383 \markdownSetup{
8384   rendererPrototypes = {
8385     jekyllDataEnd = {
8386 %       TODO: Remove the else branch in TeX Live 2021.
8387       \IfFormatAtLeastTF
8388         { 2020-10-01 }
8389         { \AddToHook{begindocument/end}{\maketitle} }
8390         {
8391           \ifx\@onlypreamble\@notprerr
8392             % We are in the document
8393             \maketitle
8394           \else
8395             % We are in the preamble
8396             \RequirePackage{etoolbox}
8397             \AfterEndPreamble{\maketitle}
8398           \fi
8399         }
8400     },
8401   },
8402 }
8403 \ExplSyntaxOff

```

**3.3.4.7 Strike-Through** If the `strikeThrough` option is enabled, we will load the `soulutf8` package and use it to implement strike-throughs.

```

8404 \markdownIfOption{strikeThrough}{%
8405   \RequirePackage{soulutf8}%
8406   \markdownSetup{
8407     rendererPrototypes = {
8408       strikeThrough = {%
8409         \st{#1}%
8410       },
8411     }
8412   }
8413 }{}

```

### 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.

```
8414 \newcommand\markdownMakeOther{%
8415   \count0=128\relax
8416   \loop
8417     \catcode\count0=11\relax
8418     \advance\count0 by 1\relax
8419   \ifnum\count0<256\repeat}%
```

### 3.4 ConT<sub>E</sub>Xt Implementation

The ConT<sub>E</sub>Xt implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV ConT<sub>E</sub>Xt formats *seem* to implement (the documentation is scarce) the majority of the plain T<sub>E</sub>X format required by the plain T<sub>E</sub>X implementation. As a consequence, we can directly reuse the existing plain T<sub>E</sub>X implementation after supplying the missing plain T<sub>E</sub>X macros.

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` L<sup>A</sup>T<sub>E</sub>X package.

```
8420 \def\markdownMakeOther{%
8421   \count0=128\relax
8422   \loop
8423     \catcode\count0=11\relax
8424     \advance\count0 by 1\relax
8425   \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 ConT<sub>E</sub>Xt.

```
8426   \catcode`|=12}%
```

#### 3.4.1 Typesetting Markdown

The `\inputmarkdown` is defined to accept an optional argument with options recognized by the ConT<sub>E</sub>Xt interface (see Section 2.4.2).

```
8427 \long\def\inputmarkdown{%
8428   \dosingleempty
8429   \doinputmarkdown}%
8430 \long\def\doinputmarkdown[#1]#2{%
8431   \begingroup
8432     \iffirstargument
8433       \setupmarkdown{#1}%
8434     \fi
8435     \markdownInput{#2}%
8436   \endgroup}%
```

The `\startmarkdown` and `\stopmarkdown` macros are implemented using the `\markdownReadAndConvert` macro.

In Knuth’s  $\text{\TeX}$ , 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) $\text{\TeX}$ , but Con $\text{\TeX}$ t MkIV funnels all input through its own input handler. This makes it possible to suppress the removal of trailing spaces in Con $\text{\TeX}$ t MkIV and therefore to insert hard line breaks into markdown text.

```

8437 \ifx\startluacode\undefined % MkII
8438   \begingroup
8439     \catcode`\|=0%
8440     \catcode`\|=12%
8441     |gdef|startmarkdown{%
8442       |markdownReadAndConvert{\stopmarkdown}%
8443       {|stopmarkdown}}%
8444     |gdef|stopmarkdown{%
8445       |markdownEnd}%
8446   |endgroup
8447 \else % MkIV
8448   \startluacode
8449     document.markdown_buffering = false
8450     local function preserve_trailing_spaces(line)
8451       if document.markdown_buffering then
8452         line = line:gsub("[ \t][ \t]$", "\t\t")
8453       end
8454       return line
8455     end
8456     resolvers.installinputlinehandler(preserve_trailing_spaces)
8457   \stopluacode
8458   \begingroup
8459     \catcode`\|=0%
8460     \catcode`\|=12%
8461     |gdef|startmarkdown{%
8462       |ctxlua{document.markdown_buffering = true}%
8463       |markdownReadAndConvert{\stopmarkdown}%
8464       {|stopmarkdown}}%
8465     |gdef|stopmarkdown{%
8466       |ctxlua{document.markdown_buffering = false}%
8467       |markdownEnd}%
8468   |endgroup
8469 \fi

```

### 3.4.2 Token Renderer Prototypes

The following configuration should be considered placeholder.

```

8470 \def\markdownRendererLineBreakPrototype{\blank}%

```

```

8471 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
8472 \def\markdownRendererRightBracePrototype{\textbraceright}%
8473 \def\markdownRendererDollarSignPrototype{\textdollar}%
8474 \def\markdownRendererPercentSignPrototype{\percent}%
8475 \def\markdownRendererUnderscorePrototype{\textunderscore}%
8476 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
8477 \def\markdownRendererBackslashPrototype{\textbackslash}%
8478 \def\markdownRendererTildePrototype{\textasciitilde}%
8479 \def\markdownRendererPipePrototype{\char`|}%
8480 \def\markdownRendererLinkPrototype#1#2#3#4{%
8481   \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
8482   \fi\tt<\hyphenatedurl{#3}>}}%
8483 \usemodule[database]
8484 \defineseparatedlist
8485   [MarkdownConTeXtCSV]
8486   [separator={,},
8487   before=\bTABLE,after=\eTABLE,
8488   first=\bTR,last=\eTR,
8489   left=\bTD,right=\eTD]
8490 \def\markdownConTeXtCSV{csv}
8491 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
8492   \def\markdownConTeXtCSV@arg{#1}%
8493   \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
8494     \placetable[] [tab:#1]{#4}{%
8495       \processseparatedfile[MarkdownConTeXtCSV][#3]}%
8496   \else
8497     \markdownInput{#3}%
8498   \fi}%
8499 \def\markdownRendererImagePrototype#1#2#3#4{%
8500   \placefigure[] []{#4}{\externalfigure[#3]}}%
8501 \def\markdownRendererULBeginPrototype{\startitemize}%
8502 \def\markdownRendererULBeginTightPrototype{\startitemize[packed]}%
8503 \def\markdownRendererULItemPrototype{\item}%
8504 \def\markdownRendererULEndPrototype{\stopitemize}%
8505 \def\markdownRendererULEndTightPrototype{\stopitemize}%
8506 \def\markdownRendererOLBeginPrototype{\startitemize[n]}%
8507 \def\markdownRendererOLBeginTightPrototype{\startitemize[packed,n]}%
8508 \def\markdownRendererOLItemPrototype{\item}%
8509 \def\markdownRendererOLItemWithNumberPrototype#1{\sym{#1.}}%
8510 \def\markdownRendererOLEndPrototype{\stopitemize}%
8511 \def\markdownRendererOLEndTightPrototype{\stopitemize}%
8512 \definedescription
8513   [MarkdownConTeXtDlItemPrototype]
8514   [location=hanging,
8515   margin=standard,
8516   headstyle=bold]%
8517 \definestartstop

```

```

8518 [MarkdownConTeXtDlPrototype]
8519 [before=\blank,
8520   after=\blank]%
8521 \definestartstop
8522 [MarkdownConTeXtDlTightPrototype]
8523 [before=\blank\startpacked,
8524   after=\stoppacked\blank]%
8525 \def\markdownRendererDlBeginPrototype{%
8526   \startMarkdownConTeXtDlPrototype}%
8527 \def\markdownRendererDlBeginTightPrototype{%
8528   \startMarkdownConTeXtDlTightPrototype}%
8529 \def\markdownRendererDlItemPrototype#1{%
8530   \startMarkdownConTeXtDlItemPrototype{#1}}%
8531 \def\markdownRendererDlItemEndPrototype{%
8532   \stopMarkdownConTeXtDlItemPrototype}%
8533 \def\markdownRendererDlEndPrototype{%
8534   \stopMarkdownConTeXtDlPrototype}%
8535 \def\markdownRendererDlEndTightPrototype{%
8536   \stopMarkdownConTeXtDlTightPrototype}%
8537 \def\markdownRendererEmphasisPrototype#1{{\em#1}}%
8538 \def\markdownRendererStrongEmphasisPrototype#1{{\bf#1}}%
8539 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
8540 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
8541 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%
8542 \def\markdownRendererInputFencedCodePrototype#1#2{%
8543   \ifx\relax#2\relax
8544     \typefile{#1}%
8545   \else

```

The code fence infostring is used as a name from the ConT<sub>E</sub>Xt `\definetying` macro. This allows the user to set up code highlighting mapping as follows:

```

\definetying [latex]
\setuptyping [latex] [option=TEX]

\starttext
  \startmarkdown
~~~ latex
\documentclass{article}
\begin{document}
  Hello world!
\end{document}
~~~
  \stopmarkdown
\stoptext

```

```

8546 \typefile[#2] []{#1}%
8547 \fi}%
8548 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
8549 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
8550 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
8551 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
8552 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
8553 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
8554 \def\markdownRendererHorizontalRulePrototype{%
8555 \blackrule[height=1pt, width=\hsize]}%
8556 \def\markdownRendererFootnotePrototype#1{\footnote{#1}}%
8557 \def\markdownRendererTickedBoxPrototype{\boxtimes$}%
8558 \def\markdownRendererHalfTickedBoxPrototype{\boxdot$}%
8559 \def\markdownRendererUntickedBoxPrototype{\square$}%
8560 \def\markdownRendererStrikeThroughPrototype#1{\overstrikes{#1}}
8561 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
8562 \def\markdownRendererSubscriptPrototype#1{\low{#1}}

```

### 3.4.2.1 Tables

There is a basic implementation of tables.

```

8563 \newcount\markdownConTeXtRowCounter
8564 \newcount\markdownConTeXtRowTotal
8565 \newcount\markdownConTeXtColumnCounter
8566 \newcount\markdownConTeXtColumnTotal
8567 \newtoks\markdownConTeXtTable
8568 \newtoks\markdownConTeXtTableFloat
8569 \def\markdownRendererTablePrototype#1#2#3{%
8570 \markdownConTeXtTable={}%
8571 \ifx\empty#1\empty
8572 \markdownConTeXtTableFloat={%
8573 \the\markdownConTeXtTable}%
8574 \else
8575 \markdownConTeXtTableFloat={%
8576 \placetable{#1}{\the\markdownConTeXtTable}}%
8577 \fi
8578 \begingroup
8579 \setupTABLE[r][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
8580 \setupTABLE[c][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
8581 \setupTABLE[r][1][topframe=on, bottomframe=on]
8582 \setupTABLE[r][#1][bottomframe=on]
8583 \markdownConTeXtRowCounter=0%
8584 \markdownConTeXtRowTotal=#2%
8585 \markdownConTeXtColumnTotal=#3%
8586 \markdownConTeXtRenderTableRow}
8587 \def\markdownConTeXtRenderTableRow#1{%
8588 \markdownConTeXtColumnCounter=0%
8589 \ifnum\markdownConTeXtRowCounter=0\relax

```

```

8590 \markdownConTeXtReadAlignments#1%
8591 \markdownConTeXtTable={\bTABLE}%
8592 \else
8593 \markdownConTeXtTable=\expandafter{%
8594 \the\markdownConTeXtTable\bTR}%
8595 \markdownConTeXtRenderTableCell#1%
8596 \markdownConTeXtTable=\expandafter{%
8597 \the\markdownConTeXtTable\eTR}%
8598 \fi
8599 \advance\markdownConTeXtRowCounter by 1\relax
8600 \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
8601 \markdownConTeXtTable=\expandafter{%
8602 \the\markdownConTeXtTable\eTABLE}%
8603 \the\markdownConTeXtTableFloat
8604 \endgroup
8605 \expandafter\gobbleoneargument
8606 \fi\markdownConTeXtRenderTableRow}
8607 \def\markdownConTeXtReadAlignments#1{%
8608 \advance\markdownConTeXtColumnCounter by 1\relax
8609 \if#1d%
8610 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
8611 \fi\if#1l%
8612 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
8613 \fi\if#1c%
8614 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
8615 \fi\if#1r%
8616 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
8617 \fi
8618 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
8619 \expandafter\gobbleoneargument
8620 \fi\markdownConTeXtReadAlignments}
8621 \def\markdownConTeXtRenderTableCell#1{%
8622 \advance\markdownConTeXtColumnCounter by 1\relax
8623 \markdownConTeXtTable=\expandafter{%
8624 \the\markdownConTeXtTable\bTD#1\eTD}%
8625 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
8626 \expandafter\gobbleoneargument
8627 \fi\markdownConTeXtRenderTableCell}
8628 \stopmodule\protect

```

## 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 T<sub>E</sub>Xbook*. 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 Miletic. *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>	248
<code>\autocites</code>	240
<code>blankBeforeBlockquote</code>	13
<code>blankBeforeCodeFence</code>	14
<code>blankBeforeHeading</code>	14
<code>breakableBlockquotes</code>	14
<code>cacheDir</code>	12, 17, 37, 38, 181



citationNbsps	15
citations	15, 68
\cite	240
\citep	240
\citet	240
codeSpans	16
compactdesc	4
compactenum	4
compactitem	4
contentBlocks	16
contentBlocksLanguageMap	17, 186
convert	218
\csvautotabular	4
\date	248
defaultOptions	7, 33, 208
\definetyping	253
definitionLists	17, 57
\directlua	4, 38, 225
eagerCache	17, 18
\enableregime	250
\endmarkdown	77
entities.char_entity	142
entities.dec_entity	142
entities.hex_entity	142
escape_citation	182, 183
escaped_citation_chars	182, 182
expandtabs	165
expectJekyllData	18
extensions	94, 182
extensions.citations	182
extensions.content_blocks	186
extensions.definition_lists	189
extensions.fancy_lists	204
extensions.fenced_code	191
extensions.footnotes	193
extensions.header_attributes	195
extensions.jekyll_data	196
extensions.pipe_table	199
extensions.strike_through	202
extensions.subscripts	203

<code>extensions.superscripts</code>	203
<code>fancyLists</code>	19, 52–57
<code>fencedCode</code>	20, 61, 229
<code>\filecontents</code>	220
<code>finalizeCache</code>	13, 17, 18, 20, 21, 37, 181
<code>footnotes</code>	21, 67
<code>frozenCacheCounter</code>	21, 181, 226, 227
<code>frozenCacheFileName</code>	13, 20, 181
<code>\g_luabridge_error_output_filename_str</code>	39
<code>\g_luabridge_helper_script_filename_str</code>	38
<code>hardLineBreaks</code>	22
<code>hashEnumerators</code>	22
<code>headerAttributes</code>	22, 26, 70–72
<code>html</code>	23, 69, 70, 239
<code>hybrid</code>	23, 30, 31, 39, 45, 84, 146, 167, 226
<code>\includegraphics</code>	4
<code>inlineFootnotes</code>	24
<code>\input</code>	35, 91, 221, 225
<code>\inputmarkdown</code>	91, 92, 250
<code>isdir</code>	3
<code>iterlines</code>	165
<code>jekyllData</code>	3, 18, 24, 62–65
<code>\jobname</code>	38, 39
<code>\label</code>	237
<code>languages_json</code>	186, 186
<code>\maketitle</code>	249
<code>\markdown</code>	77
<code>markdown</code>	77, 77, 228
<code>markdown*</code>	77, 77, 78, 228
<code>\markdown_jekyll_data_concatenate_address:NN</code>	215
<code>\markdown_jekyll_data_pop:</code>	215
<code>\markdown_jekyll_data_push:nN</code>	215
<code>\markdown_jekyll_data_push_address_segment:n</code>	213
<code>\markdown_jekyll_data_set_keyval:Nn</code>	216
<code>\markdown_jekyll_data_set_keyvals:nn</code>	216
<code>\markdown_jekyll_data_update_address_tls:</code>	215
<code>\markdownBegin</code>	35, 35, 36, 75, 77, 91

<code>\markdownEnd</code>	35, 35, 36, 75, 77, 91
<code>\markdownError</code>	75, 75
<code>\markdownExecute</code>	223
<code>\markdownExecuteDirect</code>	223, 223
<code>\markdownExecuteShellEscape</code>	223, 223
<code>\markdownIfOption</code>	218
<code>\markdownIfSnippetExists</code>	79
<code>\markdownInfo</code>	75
<code>\markdownInput</code>	35, 36, 77, 78, 92, 226, 227
<code>\markdownInputFileStream</code>	219
<code>\markdownInputPlainTeX</code>	227
<code>\markdownLuaExecute</code>	222, 224, 225, 226
<code>\markdownLuaOptions</code>	217, 218
<code>\markdownLuaRegisterIBCallback</code>	76
<code>\markdownLuaUnregisterIBCallback</code>	76
<code>\markdownMakeOther</code>	75, 250
<code>\markdownMode</code>	4, 38, 39, 76, 76, 222, 225
<code>\markdownOptionCacheDir</code>	3, 88, 218, 230
<code>\markdownOptionErrorTempFileName</code>	39, 39, 224
<code>\markdownOptionFinalizeCache</code>	37, 37, 87
<code>\markdownOptionFrozenCache</code>	13, 20, 37, 38, 83, 84, 87, 229, 230
<code>\markdownOptionFrozenCacheFileName</code>	37
<code>\markdownOptionHelperScriptFileName</code>	38, 38–40, 224
<code>\markdownOptionHybrid</code>	39, 88
<code>\markdownOptionInputTempFileName</code>	38, 220, 221
<code>\markdownOptionOutputDir</code>	39
<code>\markdownOptionOutputTempFileName</code>	38, 39, 224, 225
<code>\markdownOptionSmartEllipses</code>	88
<code>\markdownOptionStripPercentSigns</code>	41, 220
<code>\markdownOutputFileStream</code>	219
<code>\markdownPrepare</code>	218
<code>\markdownPrepareLuaOptions</code>	217
<code>\markdownReadAndConvert</code>	75, 220, 228, 251
<code>\markdownReadAndConvertProcessLine</code>	221, 222
<code>\markdownReadAndConvertStripPercentSigns</code>	220
<code>\markdownReadAndConvertTab</code>	219
<code>\markdownRendererAttributeClassName</code>	71
<code>\markdownRendererAttributeIdentifier</code>	70
<code>\markdownRendererAttributeKeyValue</code>	71
<code>\markdownRendererBlockHtmlCommentBegin</code>	69
<code>\markdownRendererBlockHtmlCommentEnd</code>	69
<code>\markdownRendererBlockQuoteBegin</code>	60

\markdownRendererBlockQuoteEnd	61
\markdownRendererCite	68, 68
\markdownRendererCodeSpan	48
\markdownRendererCodeSpanPrototype	90
\markdownRendererContentBlock	49, 49
\markdownRendererContentBlockCode	49
\markdownRendererContentBlockOnlineImage	49
\markdownRendererDlBegin	57
\markdownRendererDlBeginTight	57
\markdownRendererDlDefinitionBegin	58
\markdownRendererDlDefinitionEnd	59
\markdownRendererDlEnd	59
\markdownRendererDlEndTight	59
\markdownRendererDlItem	58
\markdownRendererDlItemEnd	58
\markdownRendererDocumentBegin	43
\markdownRendererDocumentEnd	43
\markdownRendererEllipsis	27, 44
\markdownRendererEmphasis	60, 89
\markdownRendererFancyOlBegin	53, 53
\markdownRendererFancyOlBeginTight	53
\markdownRendererFancyOlEnd	56
\markdownRendererFancyOlEndTight	57
\markdownRendererFancyOlItem	55
\markdownRendererFancyOlItemEnd	55
\markdownRendererFancyOlItemWithNumber	55
\markdownRendererFootnote	67
\markdownRendererHalfTickedBox	42
\markdownRendererHeaderAttributeContextBegin	72
\markdownRendererHeaderAttributeContextEnd	72
\markdownRendererHeadingFive	66
\markdownRendererHeadingFour	66
\markdownRendererHeadingOne	65
\markdownRendererHeadingSix	67
\markdownRendererHeadingThree	66
\markdownRendererHeadingTwo	65
\markdownRendererHorizontalRule	67
\markdownRendererImage	48
\markdownRendererImagePrototype	90
\markdownRendererInlineHtmlComment	69
\markdownRendererInlineHtmlTag	70
\markdownRendererInputBlockHtmlElement	70

\markdownRendererInputFencedCode	61
\markdownRendererInputVerbatim	61
\markdownRendererInterblockSeparator	44
\markdownRendererJekyllDataBegin	62
\markdownRendererJekyllDataBoolean	64
\markdownRendererJekyllDataEmpty	65
\markdownRendererJekyllDataEnd	62
\markdownRendererJekyllDataMappingBegin	62
\markdownRendererJekyllDataMappingEnd	63
\markdownRendererJekyllDataNumber	64
\markdownRendererJekyllDataSequenceBegin	63
\markdownRendererJekyllDataSequenceEnd	63
\markdownRendererJekyllDataString	64
\markdownRendererLineBreak	44
\markdownRendererLink	48, 89
\markdownRendererNbsp	45
\markdownRendererOlBegin	52
\markdownRendererOlBeginTight	52
\markdownRendererOlEnd	56
\markdownRendererOlEndTight	56
\markdownRendererOlItem	27, 54
\markdownRendererOlItemEnd	54
\markdownRendererOlItemWithNumber	27, 54
\markdownRendererStrikeThrough	72
\markdownRendererStrongEmphasis	60
\markdownRendererSubscript	73
\markdownRendererSuperscript	73
\markdownRendererTable	68
\markdownRendererTextCite	68
\markdownRendererTickedBox	42
\markdownRendererUlBegin	50
\markdownRendererUlBeginTight	50
\markdownRendererUlEnd	51
\markdownRendererUlEndTight	52
\markdownRendererUlItem	51
\markdownRendererUlItemEnd	51
\markdownRendererUntickedBox	42
\markdownSetup	78, 78, 227, 231
\markdownSetupSnippet	79, 79
\markdownWarning	75
new	6, 208

<code>os.execute</code>	76, 223
<code>\PackageError</code>	77, 227
<code>\PackageInfo</code>	77, 227
<code>\PackageWarning</code>	77, 227
<code>parsers</code>	153, 165
<code>parsers.commented_line</code>	155
<code>\pdfshellescape</code>	223
<code>pipeTables</code>	6, 25, 29, 68
<code>preserveTabs</code>	25, 28, 165
<code>print</code>	224, 225
<code>reader</code>	94, 94, 153, 164, 182
<code>reader-&gt;convert</code>	180, 208
<code>reader-&gt;create_parser</code>	166
<code>reader-&gt;normalize_tag</code>	165
<code>reader-&gt;options</code>	165
<code>reader-&gt;parser_functions</code>	166
<code>reader-&gt;parser_functions.name</code>	166
<code>reader-&gt;parsers</code>	165, 165
<code>reader-&gt;syntax</code>	176
<code>reader-&gt;writer</code>	165
<code>reader.new</code>	164, 164, 165
<code>relativeReferences</code>	25
<code>\setupmarkdown</code>	92, 92
<code>\shellescape</code>	223
<code>shiftHeadings</code>	6, 26
<code>slice</code>	6, 22, 26, 143
<code>smartEllipses</code>	27, 44
<code>\startmarkdown</code>	91, 91, 251
<code>startNumber</code>	27, 54, 55
<code>status.shell_escape</code>	223
<code>\stopmarkdown</code>	91, 91, 251
<code>strikeThrough</code>	27, 72, 249
<code>stripIndent</code>	28, 166
<code>subscripts</code>	28, 73
<code>superscripts</code>	28, 73
<code>tableCaptions</code>	6, 29
<code>taskLists</code>	29, 42, 239
<code>tex.print</code>	224, 225
<code>texComments</code>	30, 167

<code>\textcites</code>	240
<code>tightLists</code>	30, 50, 52, 53, 56, 57, 59, 233
<code>\title</code>	248
<code>underscores</code>	31
<code>\url</code>	4
<code>\usepackage</code>	77, 81
<code>\usetheme</code>	80
<code>util.cache</code>	95
<code>util.err</code>	95
<code>util.escaper</code>	98
<code>util.expand_tabs_in_line</code>	95
<code>util.flatten</code>	96
<code>util.intersperse</code>	97
<code>util.map</code>	97
<code>util.pathname</code>	98
<code>util.rope_last</code>	97
<code>util.rope_to_string</code>	97
<code>util.table_copy</code>	95
<code>util.walk</code>	96, 97
<code>\VerbatimInput</code>	4
<code>writer</code>	94, 94, 143, 182
<code>writer-&gt;active_attributes</code>	150
<code>writer-&gt;active_headings</code>	150
<code>writer-&gt;block_html_comment</code>	148
<code>writer-&gt;block_html_element</code>	148
<code>writer-&gt;blockquote</code>	149
<code>writer-&gt;bulletitem</code>	147
<code>writer-&gt;bulletlist</code>	146
<code>writer-&gt;citation</code>	183
<code>writer-&gt;citations</code>	183
<code>writer-&gt;code</code>	146
<code>writer-&gt;codeFence</code>	191
<code>writer-&gt;contentblock</code>	187
<code>writer-&gt;defer_call</code>	153, 153
<code>writer-&gt;definitionlist</code>	189
<code>writer-&gt;document</code>	149
<code>writer-&gt;ellipsis</code>	144
<code>writer-&gt;emphasis</code>	148
<code>writer-&gt;escape</code>	145, 146
<code>writer-&gt;escape_minimal</code>	145, 146, 183

<code>writer-&gt;escape_uri</code>	145, 146
<code>writer-&gt;escaped_chars</code>	145, 145
<code>writer-&gt;escaped_minimal_strings</code>	145, 145
<code>writer-&gt;escaped_uri_chars</code>	145, 145
<code>writer-&gt;fancyitem</code>	205
<code>writer-&gt;fancylist</code>	204
<code>writer-&gt;get_state</code>	152
<code>writer-&gt;heading</code>	150
<code>writer-&gt;hrule</code>	145
<code>writer-&gt;hybrid</code>	183
<code>writer-&gt;image</code>	146
<code>writer-&gt;inline_html_comment</code>	148
<code>writer-&gt;inline_html_tag</code>	148
<code>writer-&gt;interblocksep</code>	144
<code>writer-&gt;is_writing</code>	143, 143
<code>writer-&gt;jekyllData</code>	196
<code>writer-&gt;linebreak</code>	144
<code>writer-&gt;link</code>	146
<code>writer-&gt;nbsp</code>	144
<code>writer-&gt;note</code>	193
<code>writer-&gt;options</code>	143
<code>writer-&gt;ordereditem</code>	147
<code>writer-&gt;orderedlist</code>	147
<code>writer-&gt;pack</code>	144, 181
<code>writer-&gt;paragraph</code>	144
<code>writer-&gt;plain</code>	144
<code>writer-&gt;set_state</code>	152
<code>writer-&gt;slice_begin</code>	143
<code>writer-&gt;slice_end</code>	143
<code>writer-&gt;space</code>	144
<code>writer-&gt;strike_through</code>	202
<code>writer-&gt;string</code>	145
<code>writer-&gt;strong</code>	149
<code>writer-&gt;subscript</code>	203
<code>writer-&gt;suffix</code>	144
<code>writer-&gt;superscript</code>	203
<code>writer-&gt;table</code>	200
<code>writer-&gt;checkbox</code>	148
<code>writer-&gt;uri</code>	145
<code>writer-&gt;verbatim</code>	149
<code>writer.new</code>	143, 143
<code>\writestatus</code>	91