

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

Vít Starý Novotný, Andrej Genčur  
witiko@mail.muni.cz

Version 3.7.0-0-g98dece19  
2024-08-30

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>	<b>3</b>	<b>Implementation</b>	<b>145</b>
1.1	Requirements . . . . .	2	3.1	Lua Implementation . . . . .	145
1.2	Feedback . . . . .	6	3.2	Plain T <sub>E</sub> X Implementation	356
1.3	Acknowledgements . . . . .	7	3.3	L <sup>A</sup> T <sub>E</sub> X Implementation . . . . .	378
<b>2</b>	<b>Interfaces</b>	<b>7</b>	3.4	ConT <sub>E</sub> Xt Implementation	410
2.1	Lua Interface . . . . .	7		<b>References</b>	<b>418</b>
2.2	Plain T <sub>E</sub> X Interface . . . . .	52		<b>Index</b>	<b>419</b>
2.3	L <sup>A</sup> T <sub>E</sub> X Interface . . . . .	133			
2.4	ConT <sub>E</sub> Xt Interface . . . . .	141			

## List of Figures

1	A block diagram of the Markdown package . . . . .	8
2	A sequence diagram of typesetting a document using the T <sub>E</sub> X interface . . . . .	48
3	A sequence diagram of typesetting a document using the Lua CLI . . . . .	49
4	Various formats of mathematical formulae . . . . .	140
5	The banner of the Markdown package . . . . .	141

## 1 Introduction

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

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

---

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

<sup>2</sup>See <https://commonmark.org/>.

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 "
4             .. "to plain TeX",
5   author    = "John MacFarlane, Hans Hagen, Vít Starý Novotný, "
6             .. "Andrej Genčur",
7   copyright = {"2009-2016 John MacFarlane, Hans Hagen",
8               "2016-2024 Vít Starý Novotný, Andrej Genčur"},
9   license   = "LPPL 1.3c"
10 }
11
12 if not modules then modules = { } end
13 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 Lua<sub>TeX</sub> engine (though not necessarily in the LuaMeta<sub>TeX</sub> 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 Lua<sub>TeX</sub>  $\geq$  0.72.0 (T<sub>E</sub>X Live  $\geq$  2013).

```
14 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 note tags to the lower case. Selene Unicode is included in all releases of Lua<sub>TeX</sub> (T<sub>E</sub>X Live  $\geq$  2008).

```
15 local unicode = require("unicode")
```

**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 Lua<sub>TeX</sub> (T<sub>E</sub>X Live  $\geq$  2008).

```
16 local md5 = require("md5");
```

---

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

**Kpathsea** A package that implements the loading of third-party Lua libraries and looking up files in the T<sub>E</sub>X directory structure.

```
17 (function()
```

If Kpathsea has not been loaded before or if LuaT<sub>E</sub>X has not yet been initialized, configure Kpathsea on top of loading it. Since ConT<sub>E</sub>Xt MkIV provides a `kpse` global that acts as a stub for Kpathsea and the lua-uni-case library expects that `kpse` is a reference to the full Kpathsea library, we load Kpathsea to the `kpse` global.

```
18   local should_initialize = package.loaded.kpse == nil
19                               or tex.initialize ~= nil
20   kpse = require("kpse")
21   if should_initialize then
22     kpse.set_program_name("luatex")
23   end
24 end)()
```

All the abovelisted modules are statically linked into the current version of the LuaT<sub>E</sub>X engine [1, Section 4.3]. Beside these, we also include the following third-party Lua libraries:

**lua-uni-algos** A package that implements Unicode case-folding in T<sub>E</sub>X Live  $\geq$  2020.

```
25 hard lua-uni-algos
26 local uni_algos = require("lua-uni-algos")
```

**api7/lua-tinyyaml** 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. We carry a copy of the library in file `markdown-tinyyaml.lua` distributed together with the Markdown package.

```
27 # hard lua-tinyyaml # TODO: Uncomment after TeX Live 2022 deprecation.
```

### 1.1.2 Plain T<sub>E</sub>X Requirements

The plain T<sub>E</sub>X part of the package requires that the plain T<sub>E</sub>X 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 T<sub>E</sub>X 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.

```
28 hard l3kernel
29 \unprotect
```

```

30 \ifx\ExplSyntaxOn\undefined
31   \input expl3-generic
32 \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.

```
33 hard lt3luabridge
```

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 `cacheDir` option 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 4.2.4].

The Lua File System module is statically linked into the LuaTeX engine [1, Section 4.3].

Unless you convert markdown documents to TeX manually using the Lua command-line interface (see Section 2.1.7), 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), 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 $\epsilon$  format is loaded,

```

34 \NeedsTeXFormat{LaTeX2e}
35 \RequirePackage{expl3}

```

a TeX engine that extends  $\epsilon$ -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 2.2.6 and 3.3.4) or L<sup>A</sup>TeX themes (see Section 2.3.3) and will not be loaded if the option `plain` has been enabled (see Section 2.2.2.3):

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

```
36 soft url
```

**graphicx** A package that provides the `\includegraphics` macro for the typesetting of images. Furthermore, it also provides a key-value interface that is used in the default renderer prototypes for image attribute contexts.

```
37 soft graphics
```

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

```
38 soft paralist
```

**ifthen** A package that provides a concise syntax for the inspection of macro values. It is used in the `witiko/dot` L<sup>A</sup>T<sub>E</sub>X theme (see Section 2.3.3).

```
39 soft latex
```

```
40 soft epstopdf-pkg # required by `latex`
```

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

```
41 soft fancyvrb
```

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

```
42 soft csvsimple
```

```
43 soft pgf # required by `csvsimple`, which loads `pgfkeys.sty`
```

```
44 soft tools # required by `csvsimple`, which loads `shellesc.sty`
```

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

```
45 soft gobble
```

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

```
46 soft amsmath
```

```
47 soft amssymb
```

**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 2.3.3.

```
48 soft catchfile
```

**grffile** A package that extends the name processing of the graphics package to support a larger range of file names in  $2006 \leq \text{\TeX Live} \leq 2019$ . Since  $\text{\TeX Live} \geq 2020$ , the functionality of the package has been integrated in the  $\text{\LaTeX} 2_{\epsilon}$  kernel. It is used in the [witiko/dot](#) and [witiko/graphicx/http](#)  $\text{\LaTeX}$  themes, see Section 2.3.3.

49 `soft grffile`

**etoolbox** A package that is used to polyfill the general hook management system in the default renderer prototypes for YAML metadata, see Section 3.3.4.8, and also in the default renderer prototype for identifier attributes.

50 `soft etoolbox`

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

51 `soft soul`

**ltxcmds** A package that is used to detect whether the minted and listings packages are loaded in the default renderer prototype for fenced code blocks.

52 `soft ltxcmds`

**verse** A package that is used in the default renderer prototypes for line blocks.

53 `soft verse`

#### 1.1.4 Con $\text{\TeX}$ t Prerequisites

The Con $\text{\TeX}$ t part of the package requires that either the Mark II or the Mark IV format is loaded, all the plain  $\text{\TeX}$  prerequisites (see Section 1.1.2), and the following Con $\text{\TeX}$ t modules:

**m-database** A module that provides the default token renderer prototype for iA Writer content blocks with the CSV filename extension (see Section 2.2.6).

## 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  $\text{\TeX}$ - $\text{\LaTeX}$  Stack Exchange.<sup>5</sup> community question answering web site under the `markdown` tag.

---

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

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

### 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.

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.

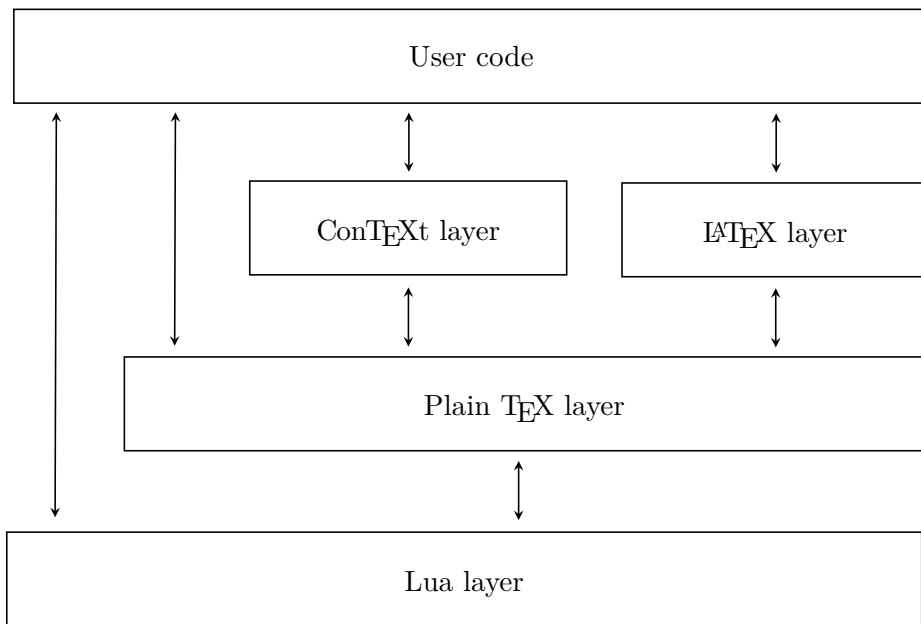
Figure 1 shows the high-level structure of the Markdown package: The translation from markdown to  $\text{T}_{\text{E}}\text{X}$  *token renderers* is exposed by the Lua layer. The plain  $\text{T}_{\text{E}}\text{X}$  layer exposes the conversion capabilities of Lua as  $\text{T}_{\text{E}}\text{X}$  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.

```
54 local M = {metadata = metadata}
```



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

</lua, lua-loader> <\*lua>

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

The Lua interface exposes the `new(options)` function. This function returns a conversion function from markdown to plain T<sub>E</sub>X according to the table `options` that contains options recognized by the Lua interface (see Section 2.1.3). The `options` parameter is optional; when unspecified, the behaviour will be the same as if `options` were an empty table.

The following example Lua code converts the markdown string `Hello *world*!` to a T<sub>E</sub>X output using the default options and prints the T<sub>E</sub>X output:

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

### 2.1.2 User-Defined Syntax Extensions

For the purpose of user-defined syntax extensions, the Lua interface also exposes the `reader` object, which performs the lexical and syntactic analysis of markdown text and which exposes the `reader->insert_pattern` and `reader->add_special_character` methods for extending the PEG grammar of markdown.



The read-only `walkable_syntax` hash table stores those rules of the PEG grammar of markdown that can be represented as an ordered choice of terminal symbols. These rules can be modified by user-defined syntax extensions.

```
55 local walkable_syntax = {
56   Block = {
57     "Blockquote",
58     "Verbatim",
59     "ThematicBreak",
60     "BulletList",
61     "OrderedList",
62     "DisplayHtml",
63     "Heading",
64   },
65   BlockOrParagraph = {
66     "Block",
67     "Paragraph",
68     "Plain",
69   },
70   Inline = {
71     "Str",
72     "Space",
73     "Endline",
74     "EndlineBreak",
75     "LinkAndEmph",
76     "Code",
77     "AutoLinkUrl",
78     "AutoLinkEmail",
79     "AutoLinkRelativeReference",
80     "InlineHtml",
81     "HtmlEntity",
82     "EscapedChar",
83     "Smart",
84     "Symbol",
85   },
86 }
```

The `reader->insert_pattern` method inserts a PEG pattern into the grammar of markdown. The method receives two mandatory arguments: a selector string in the form "*<left-hand side terminal symbol> <before, after, or instead of> <right-hand side terminal symbol>*" and a PEG pattern to insert, and an optional third argument with a name of the PEG pattern for debugging purposes (see the `debugExtensions` option). The name does not need to be unique and shall not be interpreted by the Markdown package; you can treat it as a comment.

For example. if we'd like to insert `pattern` into the grammar between the `Inline -> LinkAndEmph` and `Inline -> Code` rules, we would call

`reader->insert_pattern` with "Inline after LinkAndEmph" (or "Inline before Code") and `pattern` as the arguments.

The `reader->add_special_character` method adds a new character with special meaning to the grammar of markdown. The method receives the character as its only argument.

### 2.1.3 Options

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

```
87 local defaultOptions = {}
```

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

```
88 \ExplSyntaxOn
89 \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.

```
90 \prop_new:N \g_@@_lua_option_types_prop
91 \prop_new:N \g_@@_default_lua_options_prop
92 \seq_new:N \g_@@_option_layers_seq
93 \tl_const:Nn \c_@@_option_layer_lua_tl { lua }
94 \seq_gput_right:NV
95   \g_@@_option_layers_seq
96   \c_@@_option_layer_lua_tl
97 \cs_new:Nn
98   \@@_add_lua_option:nnn
99   {
100     \@@_add_option:Vnnn
101       \c_@@_option_layer_lua_tl
102       { #1 }
103       { #2 }
104       { #3 }
105   }
106 \cs_new:Nn
107   \@@_add_option:nnnn
108   {
109     \seq_gput_right:cn
110       { g_@@_ #1 _options_seq }
111       { #2 }
112     \prop_gput:cnn
113       { g_@@_ #1 _option_types_prop }
114       { #2 }
115       { #3 }
```

```

116     \prop_gput:cnn
117     { g_@@_default_ #1 _options_prop }
118     { #2 }
119     { #4 }
120     \@@_typecheck_option:n
121     { #2 }
122   }
123 \cs_generate_variant:Nn
124   \@@_add_option:nnnn
125   { Vnnn }
126 \tl_const:Nn \c_@@_option_value_true_tl { true }
127 \tl_const:Nn \c_@@_option_value_false_tl { false }
128 \cs_new:Nn \@@_typecheck_option:n
129   {
130     \@@_get_option_type:nN
131     { #1 }
132     \l_tmpa_tl
133     \str_case_e:Vn
134     \l_tmpa_tl
135     {
136       { \c_@@_option_type_boolean_tl }
137       {
138         \@@_get_option_value:nN
139         { #1 }
140         \l_tmpa_tl
141         \bool_if:nF
142         {
143           \str_if_eq_p:VV
144           \l_tmpa_tl
145           \c_@@_option_value_true_tl ||
146           \str_if_eq_p:VV
147           \l_tmpa_tl
148           \c_@@_option_value_false_tl
149         }
150         {
151           \msg_error:nnnV
152           { markdown }
153           { failed-typecheck-for-boolean-option }
154           { #1 }
155           \l_tmpa_tl
156         }
157       }
158     }
159   }
160 \msg_new:nnn
161   { markdown }
162   { failed-typecheck-for-boolean-option }

```

```

163 {
164   Option~#1~has~value~#2,~
165   but~a~boolean~(true~or~false)~was~expected.
166 }
167 \cs_generate_variant:Nn
168   \str_case_e:nn
169   { Vn }
170 \cs_generate_variant:Nn
171   \msg_error:nnnn
172   { nnnV }
173 \seq_new:N
174   \g_@@_option_types_seq
175 \tl_const:Nn
176   \c_@@_option_type_clist_tl
177   { clist }
178 \seq_gput_right:NV
179   \g_@@_option_types_seq
180   \c_@@_option_type_clist_tl
181 \tl_const:Nn
182   \c_@@_option_type_counter_tl
183   { counter }
184 \seq_gput_right:NV
185   \g_@@_option_types_seq
186   \c_@@_option_type_counter_tl
187 \tl_const:Nn
188   \c_@@_option_type_boolean_tl
189   { boolean }
190 \seq_gput_right:NV
191   \g_@@_option_types_seq
192   \c_@@_option_type_boolean_tl
193 \tl_const:Nn
194   \c_@@_option_type_number_tl
195   { number }
196 \seq_gput_right:NV
197   \g_@@_option_types_seq
198   \c_@@_option_type_number_tl
199 \tl_const:Nn
200   \c_@@_option_type_path_tl
201   { path }
202 \seq_gput_right:NV
203   \g_@@_option_types_seq
204   \c_@@_option_type_path_tl
205 \tl_const:Nn
206   \c_@@_option_type_slice_tl
207   { slice }
208 \seq_gput_right:NV
209   \g_@@_option_types_seq

```

```

210 \c_@@_option_type_slice_tl
211 \tl_const:Nn
212 \c_@@_option_type_string_tl
213 { string }
214 \seq_gput_right:NV
215 \g_@@_option_types_seq
216 \c_@@_option_type_string_tl
217 \cs_new:Nn
218 \@@_get_option_type:nN
219 {
220   \bool_set_false:N
221     \l_tmpa_bool
222   \seq_map_inline:Nn
223     \g_@@_option_layers_seq
224     {
225       \prop_get:cnNT
226         { g_@@_ ##1 _option_types_prop }
227         { #1 }
228         \l_tmpa_tl
229         {
230           \bool_set_true:N
231             \l_tmpa_bool
232           \seq_map_break:
233         }
234       }
235   \bool_if:nF
236     \l_tmpa_bool
237     {
238       \msg_error:nnn
239         { markdown }
240         { undefined-option }
241         { #1 }
242     }
243   \seq_if_in:NVF
244     \g_@@_option_types_seq
245     \l_tmpa_tl
246     {
247       \msg_error:nnnV
248         { markdown }
249         { unknown-option-type }
250         { #1 }
251       \l_tmpa_tl
252     }
253   \tl_set_eq:NN
254     #2
255     \l_tmpa_tl
256 }

```

```

257 \msg_new:nnn
258   { markdown }
259   { unknown-option-type }
260   {
261     Option~#1~has~unknown~type~#2.
262   }
263 \msg_new:nnn
264   { markdown }
265   { undefined-option }
266   {
267     Option~#1~is~undefined.
268   }
269 \cs_new:Nn
270   \@@_get_default_option_value:nN
271   {
272     \bool_set_false:N
273       \l_tmpa_bool
274     \seq_map_inline:Nn
275       \g_@@_option_layers_seq
276       {
277         \prop_get:cnNT
278           { g_@@_default_ ##1 _options_prop }
279           { #1 }
280           #2
281           {
282             \bool_set_true:N
283               \l_tmpa_bool
284             \seq_map_break:
285           }
286         }
287     \bool_if:nF
288       \l_tmpa_bool
289       {
290         \msg_error:nnn
291           { markdown }
292           { undefined-option }
293           { #1 }
294       }
295   }
296 \cs_new:Nn
297   \@@_get_option_value:nN
298   {
299     \@@_option_tl_to_csname:nN
300       { #1 }
301     \l_tmpa_tl
302     \cs_if_free:cTF
303       { \l_tmpa_tl }

```

```

304     {
305       \@@_get_default_option_value:nN
306       { #1 }
307       #2
308     }
309     {
310       \@@_get_option_type:nN
311       { #1 }
312       \l_tmpa_tl
313       \str_if_eq:NNTF
314       \c_@@_option_type_counter_tl
315       \l_tmpa_tl
316       {
317         \@@_option_tl_to_csname:nN
318         { #1 }
319         \l_tmpa_tl
320         \tl_set:Nx
321         #2
322         { \the \cs:w \l_tmpa_tl \cs_end: }
323       }
324       {
325         \@@_option_tl_to_csname:nN
326         { #1 }
327         \l_tmpa_tl
328         \tl_set:Nv
329         #2
330         { \l_tmpa_tl }
331       }
332     }
333   }
334   \cs_new:Nn \@@_option_tl_to_csname:nN
335   {
336     \tl_set:Nn
337     \l_tmpa_tl
338     { \str_uppercase:n { #1 } }
339     \tl_set:Nx
340     #2
341     {
342       markdownOption
343       \tl_head:f { \l_tmpa_tl }
344       \tl_tail:n { #1 }
345     }
346   }

```

To make it easier to support different coding styles in the interface, engines, we define the `\@@_with_various_cases:nn` function that allows us to generate different variants of a string using different cases.

```

347 \cs_new:Nn \@@_with_various_cases:nn
348 {
349   \seq_clear:N
350   \l_tmpa_seq
351   \seq_map_inline:Nn
352   \g_@@_cases_seq
353   {
354     \tl_set:Nn
355     \l_tmpa_tl
356     { #1 }
357     \use:c { ##1 }
358     \l_tmpa_tl
359     \seq_put_right:NV
360     \l_tmpa_seq
361     \l_tmpa_tl
362   }
363   \seq_map_inline:Nn
364   \l_tmpa_seq
365   { #2 }
366 }

```

To interrupt the `\@@_with_various_cases:nn` function prematurely, use the `\@@_with_various_cases_break:` function.

```

367 \cs_new:Nn \@@_with_various_cases_break:
368 {
369   \seq_map_break:
370 }

```

By default, camelCase and snake\_case are supported. Additional cases can be added by adding functions to the `\g_@@_cases_seq` sequence.

```

371 \seq_new:N \g_@@_cases_seq
372 \cs_new:Nn \@@_camel_case:N
373 {
374   \regex_replace_all:nnN
375   { _ ([a-z]) }
376   { \c { str_uppercase:n } \cB\{ \1 \cE\} }
377   #1
378   \tl_set:Nx
379   #1
380   { #1 }
381 }
382 \seq_gput_right:Nn \g_@@_cases_seq { @@_camel_case:N }
383 \cs_new:Nn \@@_snake_case:N
384 {
385   \regex_replace_all:nnN
386   { ([a-z])([A-Z]) }
387   { \1 _ \c { str_lowercase:n } \cB\{ \2 \cE\} }
388   #1

```



```

389     \tl_set:Nx
390         #1
391         { #1 }
392     }
393 \seq_gput_right:Nn \g_@@_cases_seq { @@_snake_case:N }

```

## 2.1.4 General Behavior

`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. Furthermore, it can also significantly improve the processing speed for documents that require multiple compilation runs, since each markdown document is only converted once. 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. However, it makes it impossible to post-process the converted documents and recover historical versions of the documents from the cache. Furthermore, it can significantly reduce the processing speed for documents that require multiple compilation runs, since each markdown document is converted multiple times needlessly.

This behavior will only be used when the `finalizeCache` option is disabled.

```

394 \@@_add_lua_option:nnn
395   { eagerCache }
396   { boolean }
397   { true }
398 defaultOptions.eagerCache = true

```

`singletonCache=true, false`

default: `true`

**true**      Conversion functions produced by the function `new(options)` will be cached in an LRU cache of size 1 keyed by `options`. This is more time-

and space-efficient than always producing a new conversion function but may expose bugs related to the idempotence of conversion functions.

This has been the default behavior since version 3.0.0 of the Markdown package.

**false** Every call to the function `new(options)` will produce a new conversion function that will not be cached. This is slower than caching conversion functions and may expose bugs related to memory leaks in the creation of conversion functions, see also #226 (comment)<sup>6</sup>.

This was the default behavior until version 3.0.0 of the Markdown package.

```
399 \@@_add_lua_option:nnn
400   { singletonCache }
401   { boolean }
402   { true }

403 defaultOptions.singletonCache = true

404 local singletonCache = {
405   convert = nil,
406   options = nil,
407 }
```

`unicodeNormalization=true, false`

default: true

**true** Markdown documents will be normalized using one of the four Unicode normalization forms<sup>7</sup> before conversion. The Unicode normalization norm used is determined by option `unicodeNormalizationForm`.

**false** Markdown documents will not be Unicode-normalized before conversion.

```
408 \@@_add_lua_option:nnn
409   { unicodeNormalization }
410   { boolean }
411   { true }

412 defaultOptions.unicodeNormalization = true
```

---

<sup>6</sup>See <https://github.com/witiko/markdown/pull/226#issuecomment-1599641634>.

<sup>7</sup>See <https://unicode.org/faq/normalization.html>.

`unicodeNormalizationForm=nfc, nfd, nfkc, nfkd`  
default: `nfc`

- `nfc` When option `unicodeNormalization` has been enabled, markdown documents will be normalized using Unicode Normalization Form C (NFC) before conversion.
- `nfd` When option `unicodeNormalization` has been enabled, markdown documents will be normalized using Unicode Normalization Form D (NFD) before conversion.
- `nfkc` When option `unicodeNormalization` has been enabled, markdown documents will be normalized using Unicode Normalization Form KC (NFKC) before conversion.
- `nfkd` When option `unicodeNormalization` has been enabled, markdown documents will be normalized using Unicode Normalization Form KD (NFKD) before conversion.

```
413 \@@_add_lua_option:nnn
414   { unicodeNormalizationForm }
415   { string }
416   { nfc }
417 defaultOptions.unicodeNormalizationForm = "nfc"
```

### 2.1.5 File and Directory Names

`cacheDir=<path>` 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.

```
418 \@@_add_lua_option:nnn
419   { cacheDir }
420   { path }
421   { \markdownOptionOutputDir / _markdown_\jobname }
422 defaultOptions.cacheDir = "."
```

`contentBlocksLanguageMap`= $\langle filename \rangle$   
default: `markdown-languages.json`

The filename of the JSON file that maps filename extensions to programming language names in the iA Writer content blocks when the `contentBlocks` option is enabled. See Section 2.2.5.9 for more information.

```
423 \@@_add_lua_option:nnn
424   { contentBlocksLanguageMap }
425   { path }
426   { markdown-languages.json }
427 defaultOptions.contentBlocksLanguageMap = "markdown-languages.json"
```

`debugExtensionsFileName`= $\langle filename \rangle$  default: `debug-extensions.json`

The filename of the JSON file that will be produced when the `debugExtensions` option is enabled. This file will contain the extensible subset of the PEG grammar of markdown (see the `walkable_syntax` hash table) after built-in syntax extensions (see Section 3.1.7) and user-defined syntax extensions (see Section 2.1.2) have been applied.

```
428 \@@_add_lua_option:nnn
429   { debugExtensionsFileName }
430   { path }
431   { \markdownOptionOutputDir / \jobname .debug-extensions.json }
432 defaultOptions.debugExtensionsFileName = "debug-extensions.json"
```

`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  $\TeX$  document that contains markdown documents without invoking Lua using the `frozenCache` plain  $\TeX$  option. 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.

```
433 \@@_add_lua_option:nnn
434   { frozenCacheFileName }
435   { path }
436   { \markdownOptionCacheDir / frozenCache.tex }
437 defaultOptions.frozenCacheFileName = "frozenCache.tex"
```

## 2.1.6 Parser Options

`autoIdentifiers=true, false` default: false

`true` Enable the Pandoc auto identifiers syntax extension<sup>8</sup>:

The following heading received the identifier ``sesame-street``:

```
# 123 Sesame Street
```

`false` Disable the Pandoc auto identifiers syntax extension.

See also the option `gfmAutoIdentifiers`.

```
438 \@@_add_lua_option:nnn
439   { autoIdentifiers }
440   { boolean }
441   { false }
442 defaultOptions.autoIdentifiers = false
```

`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.

```
443 \@@_add_lua_option:nnn
444   { blankBeforeBlockquote }
445   { boolean }
446   { false }
447 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.

```
448 \@@_add_lua_option:nnn
449   { blankBeforeCodeFence }
450   { boolean }
451   { false }
452 defaultOptions.blankBeforeCodeFence = false
```

---

<sup>8</sup>See [https://pandoc.org/MANUAL.html#extension-auto\\_identifiers](https://pandoc.org/MANUAL.html#extension-auto_identifiers).

`blankBeforeDivFence=true, false` default: false

- `true`      Require a blank line before the closing fence of a fenced div.
- `false`     Do not require a blank line before the closing fence of a fenced div.

```
453 \@@_add_lua_option:nnn
454   { blankBeforeDivFence }
455   { boolean }
456   { false }

457 defaultOptions.blankBeforeDivFence = 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.

```
458 \@@_add_lua_option:nnn
459   { blankBeforeHeading }
460   { boolean }
461   { false }

462 defaultOptions.blankBeforeHeading = false
```

`blankBeforeList=true, false` default: false

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

```
463 \@@_add_lua_option:nnn
464   { blankBeforeList }
465   { boolean }
466   { false }

467 defaultOptions.blankBeforeList = false
```

`bracketedSpans=true, false` default: `false`

`true` Enable the Pandoc bracketed span syntax extension<sup>9</sup>:

`[This is *some text*]{.class key=val}`

`false` Disable the Pandoc bracketed span syntax extension.

```
468 \@@_add_lua_option:nnn
469   { bracketedSpans }
470   { boolean }
471   { false }

472 defaultOptions.bracketedSpans = false
```

`breakableBlockquotes=true, false` default: `true`

`true` A blank line separates block quotes.

`false` Blank lines in the middle of a block quote are ignored.

```
473 \@@_add_lua_option:nnn
474   { breakableBlockquotes }
475   { boolean }
476   { true }

477 defaultOptions.breakableBlockquotes = true
```

`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.

```
478 \@@_add_lua_option:nnn
479   { citationNbsps }
480   { boolean }
481   { true }

482 defaultOptions.citationNbsps = true
```

---

<sup>9</sup>See [https://pandoc.org/MANUAL.html#extension-bracketed\\_spans](https://pandoc.org/MANUAL.html#extension-bracketed_spans).

`citations=true, false`

default: false

`true` Enable the Pandoc citation syntax extension<sup>10</sup>:

```
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.

```
483 \@@_add_lua_option:nnn
484 { citations }
485 { boolean }
486 { false }
487 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.``
```

```
488 \@@_add_lua_option:nnn
489 { codeSpans }
490 { boolean }
491 { true }
492 defaultOptions.codeSpans = true
```

<sup>10</sup>See <https://pandoc.org/MANUAL.html#extension-citations>.



`contentBlocks=true, false`

default: `false`

`true`

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

```
``` md
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.

```
493 \@@_add_lua_option:nnn
494   { contentBlocks }
495   { boolean }
496   { false }

497 defaultOptions.contentBlocks = false
```

`contentLevel=block, inline`

default: `block`

`block`      Treat content as a sequence of blocks.

```
- this is a list
- it contains two items
```

`inline`     Treat all content as inline content.

```
- this is a text
- not a list
```

```
498 \@@_add_lua_option:nnn
499   { contentLevel }
500   { string }
501   { block }

502 defaultOptions.contentLevel = "block"
```

`debugExtensions=true, false`

default: `false`

- `true` Produce a JSON file that will contain the extensible subset of the PEG grammar of markdown (see the `walkable_syntax` hash table) after built-in syntax extensions (see Section 3.1.7) and user-defined syntax extensions (see Section 2.1.2) have been applied. This helps you to see how the different extensions interact. The name of the produced JSON file is controlled by the `debugExtensionsFileName` option.
- `false` Do not produce a JSON file with the PEG grammar of markdown.

```
503 \@@_add_lua_option:nnn
504   { debugExtensions }
505   { boolean }
506   { false }

507 defaultOptions.debugExtensions = false
```

`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.

```
508 \@@_add_lua_option:nnn
509   { definitionLists }
510   { boolean }
511   { false }

512 defaultOptions.definitionLists = false
```

`ensureJekyllData=true, false`

default: `false`

- false** When the `jekyllData` and `expectJekyllData` options are enabled, then a markdown document may begin directly with YAML metadata and may contain nothing but YAML metadata. Otherwise, the markdown document is processed as markdown text.
- true** When the `jekyllData` and `expectJekyllData` options are enabled, then a markdown document must begin directly with YAML metadata and must contain nothing but YAML metadata. Otherwise, an error is produced.

```
513 \@@_add_lua_option:nnn
514   { ensureJekyllData }
515   { boolean }
516   { false }
517 defaultOptions.ensureJekyllData = false
```

`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 `jeekyllData` 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}
```

```
518 \@@_add_lua_option:nnn
519 { expectJekyllData }
520 { boolean }
521 { false }

522 defaultOptions.expectJekyllData = false
```

`extensions=<filenames>`

The filenames of user-defined syntax extensions that will be applied to the markdown reader. If the `kpathsea` library is available, files will be searched for not only in the current working directory but also in the  $\TeX$  directory structure.

A user-defined syntax extension is a Lua file in the following format:

```
local strike_through = {
  api_version = 2,
  grammar_version = 4,
  finalize_grammar = function(reader)
    local nonspacechar = lpeg.P(1) - lpeg.S("\t ")
    local doubleslashes = lpeg.P("//")
```

```

local function between(p, starter, ender)
    ender = lpeg.B(nonspacechar) * ender
    return (starter * #nonspacechar
            * lpeg.Ct(p * (p - ender)^0) * ender)
end

local read_strike_through = between(
    lpeg.V("Inline"), doubleslashes, doubleslashes
) / function(s) return {"\\st{" , s, "}"} end

reader.insert_pattern("Inline after LinkAndEmph", read_strike_through,
                    "StrikeThrough")
reader.add_special_character("/")
end
}

return strike_through

```

The `api_version` and `grammar_version` fields specify the version of the user-defined syntax extension API and the markdown grammar for which the extension was written. See the current API and grammar versions below:

```

523 metadata.user_extension_api_version = 2
524 metadata.grammar_version = 4

```

Any changes to the syntax extension API or grammar will cause the corresponding current version to be incremented. After Markdown 3.0.0, any changes to the API and the grammar will be either backwards-compatible or constitute a breaking change that will cause the major version of the Markdown package to increment (to 4.0.0).

The `finalize_grammar` field is a function that finalizes the grammar of markdown using the interface of a Lua `reader` object, such as the `reader->insert_pattern` and `reader->add_special_character` methods, see Section 2.1.2.

```

525 \cs_generate_variant:Nn
526 \@@_add_lua_option:nnn
527 { nnV }
528 \@@_add_lua_option:nnV
529 { extensions }
530 { clist }
531 \c_empty_clist
532 defaultOptions.extensions = {}

```

`fancyLists=true, false`

default: false

**true** Enable the Pandoc fancy list syntax extension<sup>11</sup>:

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

**false** Disable the Pandoc fancy list syntax extension.

```
533 \@@_add_lua_option:nnn
534 { fancyLists }
535 { boolean }
536 { false }
537 defaultOptions.fancyLists = false
```

`fencedCode=true, false`

default: true

**true** Enable the commonmark fenced code block extension:

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

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

**false** Disable the commonmark fenced code block extension.

```
538 \@@_add_lua_option:nnn
539 { fencedCode }
540 { boolean }
541 { true }
542 defaultOptions.fencedCode = true
```

<sup>11</sup>See <https://pandoc.org/MANUAL.html#org-fancy-lists>.

`fencedCodeAttributes=true, false`

default: false

**true** Enable the Pandoc fenced code attribute syntax extension<sup>12</sup>:

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

**false** Disable the Pandoc fenced code attribute syntax extension.

```
543 \@@_add_lua_option:nnn
544 { fencedCodeAttributes }
545 { boolean }
546 { false }

547 defaultOptions.fencedCodeAttributes = false
```

`fencedDivs=true, false`

default: false

**true** Enable the Pandoc fenced div syntax extension<sup>13</sup>:

```
::::: {#special .sidebar}
Here is a paragraph.

And another.
:::::
```

**false** Disable the Pandoc fenced div syntax extension.

```
548 \@@_add_lua_option:nnn
549 { fencedDivs }
550 { boolean }
551 { false }

552 defaultOptions.fencedDivs = false
```

<sup>12</sup>See [https://pandoc.org/MANUAL.html#extension-fenced\\_code\\_attributes](https://pandoc.org/MANUAL.html#extension-fenced_code_attributes).

<sup>13</sup>See [https://pandoc.org/MANUAL.html#extension-fenced\\_divs](https://pandoc.org/MANUAL.html#extension-fenced_divs).

`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  $\text{T}_{\text{E}}\text{X}$  document that contains markdown documents without invoking Lua using the `frozenCache` plain  $\text{T}_{\text{E}}\text{X}$  option. As a result, the plain  $\text{T}_{\text{E}}\text{X}$  document becomes more portable, but further changes in the order and the content of markdown documents will not be reflected.

```
553 \@@_add_lua_option:nnn
554   { finalizeCache }
555   { boolean }
556   { false }

557 defaultOptions.finalizeCache = 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  $\text{T}_{\text{E}}\text{X}$  macro `\markdownFrozenCache<number>` that will typeset markdown document number `<number>`.

```
558 \@@_add_lua_option:nnn
559   { frozenCacheCounter }
560   { counter }
561   { 0 }

562 defaultOptions.frozenCacheCounter = 0
```

`gfmAutoIdentifiers=true, false`

default: `false`

`true` Enable the Pandoc GitHub-flavored auto identifiers syntax extension<sup>14</sup>:

```
The following heading received the identifier `123-sesame-street`:

# 123 Sesame Street
```

`false` Disable the Pandoc GitHub-flavored auto identifiers syntax extension.

---

<sup>14</sup>See [https://pandoc.org/MANUAL.html#extension-gfm\\_auto\\_identifiers](https://pandoc.org/MANUAL.html#extension-gfm_auto_identifiers).



See also the option [autoIdentifiers](#).

```
563 \@@_add_lua_option:nnn
564   { gfmAutoIdentifiers }
565   { boolean }
566   { false }

567 defaultOptions.gfmAutoIdentifiers = 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.

```
568 \@@_add_lua_option:nnn
569   { hashEnumerators }
570   { boolean }
571   { false }

572 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}
=====
```

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

```
573 \@@_add_lua_option:nnn
574   { headerAttributes }
575   { boolean }
576   { false }

577 defaultOptions.headerAttributes = false
```

`html=true, false` default: true

- `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.

```
578 \@@_add_lua_option:nnn
579   { html }
580   { boolean }
581   { true }

582 defaultOptions.html = true
```

`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.

```
583 \@@_add_lua_option:nnn
584   { hybrid }
585   { boolean }
586   { false }

587 defaultOptions.hybrid = false
```

`inlineCodeAttributes=true, false` default: false

- `true` Enable the Pandoc inline code span attribute extension<sup>15</sup>:

``<$>`{.haskell}`

---

<sup>15</sup>See [https://pandoc.org/MANUAL.html#extension-inline\\_code\\_attributes](https://pandoc.org/MANUAL.html#extension-inline_code_attributes).

`false` Enable the Pandoc inline code span attribute extension.

```
588 \@@_add_lua_option:nnn
589 { inlineCodeAttributes }
590 { boolean }
591 { false }

592 defaultOptions.inlineCodeAttributes = false
```

`inlineNotes=true, false` default: false

`true` Enable the Pandoc inline note syntax extension<sup>16</sup>:

```
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 note syntax extension.

```
593 \@@_add_lua_option:nnn
594 { inlineNotes }
595 { boolean }
596 { false }

597 defaultOptions.inlineNotes = false
```

`jeekyllData=true, false` default: false

`true` Enable the Pandoc YAML metadata block syntax extension<sup>17</sup> 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.
---
```

<sup>16</sup>See [https://pandoc.org/MANUAL.html#extension-inline\\_notes](https://pandoc.org/MANUAL.html#extension-inline_notes).

<sup>17</sup>See [https://pandoc.org/MANUAL.html#extension-yaml\\_metadata\\_block](https://pandoc.org/MANUAL.html#extension-yaml_metadata_block).

**false** Disable the Pandoc YAML metadata block syntax extension for entering metadata in YAML.

```
598 \@@_add_lua_option:nnn
599 { jekyllData }
600 { boolean }
601 { false }
602 defaultOptions.jekyllData = false
```

**linkAttributes=true, false** default: false

**true** Enable the Pandoc link and image attribute syntax extension<sup>18</sup>:

An inline `![image](foo.jpg){#id .class width=30 height=20px}` and a reference `![image][ref]` with attributes.

`[ref]: foo.jpg "optional title" {#id .class key=val key2=val2}`

**false** Enable the Pandoc link and image attribute syntax extension.

```
603 \@@_add_lua_option:nnn
604 { linkAttributes }
605 { boolean }
606 { false }
607 defaultOptions.linkAttributes = false
```

**lineBlocks=true, false** default: false

**true** Enable the Pandoc line block syntax extension<sup>19</sup>:

```
| this is a line block that
| spans multiple
| even
| discontinuous
| lines
```

**false** Disable the Pandoc line block syntax extension.

```
608 \@@_add_lua_option:nnn
609 { lineBlocks }
610 { boolean }
611 { false }
612 defaultOptions.lineBlocks = false
```

---

<sup>18</sup>See [https://pandoc.org/MANUAL.html#extension-link\\_attributes](https://pandoc.org/MANUAL.html#extension-link_attributes).

<sup>19</sup>See [https://pandoc.org/MANUAL.html#extension-line\\_blocks](https://pandoc.org/MANUAL.html#extension-line_blocks).

`mark=true, false` default: false

`true` Enable the Pandoc mark syntax extension<sup>20</sup>:

```
This ==is highlighted text.==
```

`false` Disable the Pandoc mark syntax extension.

```
613 \@@_add_lua_option:nnn
614   { mark }
615   { boolean }
616   { false }
617 defaultOptions.mark = false
```

`notes=true, false` default: false

`true` Enable the Pandoc note syntax extension<sup>21</sup>:

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

```
[^1]: Here is the note.
```

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

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

```
        { some.code }
```

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

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

`false` Disable the Pandoc note syntax extension.

```
618 \@@_add_lua_option:nnn
619   { notes }
620   { boolean }
621   { false }
622 defaultOptions.notes = false
```

<sup>20</sup>See <https://pandoc.org/MANUAL.html#extension-mark>.

<sup>21</sup>See <https://pandoc.org/MANUAL.html#extension-footnotes>.

`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.

```
623 \@@_add_lua_option:nnn
624 { pipeTables }
625 { boolean }
626 { false }

627 defaultOptions.pipeTables = false
```

`preserveTabs=true, false`

default: true

**true** Preserve tabs in code block and fenced code blocks.

**false** Convert any tabs in the input to spaces.

```
628 \@@_add_lua_option:nnn
629 { preserveTabs }
630 { boolean }
631 { true }

632 defaultOptions.preserveTabs = true
```

`rawAttribute=true, false`

default: false

**true** Enable the Pandoc raw attribute syntax extension<sup>22</sup>:

```
`$H_2 O$`{=tex} is a liquid.
```

To enable raw blocks, the `fencedCode` option must also be enabled:

```
Here is a mathematical formula:
```{=tex}
\[distance[i] =
  \begin{dcases}
    a & b \\
  \end{dcases}
\]
```

<sup>22</sup>See [https://pandoc.org/MANUAL.html#extension-raw\\_attribute](https://pandoc.org/MANUAL.html#extension-raw_attribute).

```
        c & d
    \end{dcases}
\]
---
```

The `rawAttribute` option is a good alternative to the `hybrid` option. Unlike the `hybrid` option, which affects the entire document, the `rawAttribute` option allows you to isolate the parts of your documents that use TeX:

`false`      Disable the Pandoc raw attribute syntax extension.

```
633 \@@_add_lua_option:nnn
634   { rawAttribute }
635   { boolean }
636   { false }
637 defaultOptions.rawAttribute = false
```

`relativeReferences=true, false`

default: `false`

`true`      Enable relative references<sup>23</sup> in autolinks:

```
I conclude in Section <#conclusion>.

Conclusion {#conclusion}
=====
In this paper, we have discovered that most
grandmas would rather eat dinner with their
grandchildren than get eaten. Begone, wolf!
```

`false`      Disable relative references in autolinks.

```
638 \@@_add_lua_option:nnn
639   { relativeReferences }
640   { boolean }
641   { false }
642 defaultOptions.relativeReferences = false
```

---

<sup>23</sup>See <https://datatracker.ietf.org/doc/html/rfc3986#section-4.2>.

`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.

```
643 \@@_add_lua_option:nnn
644   { shiftHeadings }
645   { number }
646   { 0 }

647 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 (`$`) selects the end of a document.
- `^<identifier>` selects the beginning of a section (see the `headerAttributes` option) or a fenced div (see the `fencedDivs` option) with the HTML attribute `#<identifier>`.
- `$<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.

```
648 \@@_add_lua_option:nnn
649   { slice }
650   { slice }
651   { ^-$ }

652 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.

```
653 \@@_add_lua_option:nnn
654 { smartEllipses }
655 { boolean }
656 { false }

657 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.

```
658 \@@_add_lua_option:nnn
659 { startNumber }
660 { boolean }
661 { true }

662 defaultOptions.startNumber = true
```

`strikeThrough=true, false` default: false

`true` Enable the Pandoc strike-through syntax extension<sup>24</sup>:

This ~~is deleted text.~~

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

```
663 \@@_add_lua_option:nnn
664 { strikeThrough }
665 { boolean }
666 { false }

667 defaultOptions.strikeThrough = false
```

---

<sup>24</sup>See <https://pandoc.org/MANUAL.html#extension-strikeout>.

`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.

```
668 \@@_add_lua_option:nnn
669   { stripIndent }
670   { boolean }
671   { false }
672 defaultOptions.stripIndent = false
```

`subscripts=true, false`

default: `false`

**true** Enable the Pandoc subscript syntax extension<sup>25</sup>:

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

**false** Disable the Pandoc subscript syntax extension.

```
673 \@@_add_lua_option:nnn
674   { subscripts }
675   { boolean }
676   { false }
677 defaultOptions.subscripts = false
```

---

<sup>25</sup>See <https://pandoc.org/MANUAL.html#extension-superscript-subscript>.

`superscripts=true, false`

default: `false`

`true` Enable the Pandoc superscript syntax extension<sup>26</sup>:

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

`false` Disable the Pandoc superscript syntax extension.

```
678 \@@_add_lua_option:nnn
679 { superscripts }
680 { boolean }
681 { false }

682 defaultOptions.superscripts = false
```

`tableAttributes=true, false`

default: `false`

`true`

: Enable the assignment of HTML attributes to table captions (see the `tableCaptions` option).

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

: Demonstration of pipe table syntax. {#example-table}
```
```

`false` Disable the assignment of HTML attributes to table captions.

```
683 \@@_add_lua_option:nnn
684 { tableAttributes }
685 { boolean }
686 { false }

687 defaultOptions.tableAttributes = false
```

<sup>26</sup>See <https://pandoc.org/MANUAL.html#extension-superscript-subscript>.

`tableCaptions=true, false`

default: `false`

`true`

: Enable the Pandoc table caption syntax extension<sup>27</sup> for pipe tables (see the `pipeTables` option).

```
``` md
| 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 caption syntax extension.

```
688 \@@_add_lua_option:nnn
689 { tableCaptions }
690 { boolean }
691 { false }

692 defaultOptions.tableCaptions = false
```

`taskLists=true, false`

default: `false`

`true` Enable the Pandoc task list syntax extension<sup>28</sup>:

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

`false` Disable the Pandoc task list syntax extension.

```
693 \@@_add_lua_option:nnn
694 { taskLists }
695 { boolean }
696 { false }

697 defaultOptions.taskLists = false
```

<sup>27</sup>See [https://pandoc.org/MANUAL.html#extension-table\\_captions](https://pandoc.org/MANUAL.html#extension-table_captions).

<sup>28</sup>See [https://pandoc.org/MANUAL.html#extension-task\\_lists](https://pandoc.org/MANUAL.html#extension-task_lists).

`texComments=true, false`

default: false

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

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

Always enabled when `hybrid` is enabled.

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

```
698 \@@_add_lua_option:nnn
699   { texComments }
700   { boolean }
701   { false }
702 defaultOptions.texComments = false
```

`texMathDollars=true, false`

default: false

**true** Enable the Pandoc dollar math syntax extension<sup>29</sup>:

```
inline math: $E=mc^2$
display math: $$E=mc^2$$
```

**false** Disable the Pandoc dollar math syntax extension.

```
703 \@@_add_lua_option:nnn
704   { texMathDollars }
705   { boolean }
706   { false }
707 defaultOptions.texMathDollars = false
```

---

<sup>29</sup>See [https://pandoc.org/MANUAL.html#extension-tex\\_math\\_dollars](https://pandoc.org/MANUAL.html#extension-tex_math_dollars).

`texMathDoubleBackslash=true, false` default: false

**true** Enable the Pandoc double backslash math syntax extension<sup>30</sup>:

```
inline math: \\(E=mc^2\\)
display math: \\[E=mc^2\\]
```

**false** Disable the Pandoc double backslash math syntax extension.

```
708 \\@@_add_lua_option:nnn
709   { texMathDoubleBackslash }
710   { boolean }
711   { false }

712 defaultOptions.texMathDoubleBackslash = false
```

`texMathSingleBackslash=true, false` default: false

**true** Enable the Pandoc single backslash math syntax extension<sup>31</sup>:

```
inline math: \ (E=mc^2\ )
display math: \ [E=mc^2\ ]
```

**false** Disable the Pandoc single backslash math syntax extension.

```
713 \\@@_add_lua_option:nnn
714   { texMathSingleBackslash }
715   { boolean }
716   { false }

717 defaultOptions.texMathSingleBackslash = 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:

---

<sup>30</sup>See [https://pandoc.org/MANUAL.html#extension-tex\\_math\\_double\\_backslash](https://pandoc.org/MANUAL.html#extension-tex_math_double_backslash).

<sup>31</sup>See [https://pandoc.org/MANUAL.html#extension-tex\\_math\\_single\\_backslash](https://pandoc.org/MANUAL.html#extension-tex_math_single_backslash).

```

- 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.

```

718 \@@_add_lua_option:nmn
719 { tightLists }
720 { boolean }
721 { true }

722 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.

```

723 \@@_add_lua_option:nmn
724 { underscores }
725 { boolean }
726 { true }
727 \ExplSyntaxOff

728 defaultOptions.underscores = true

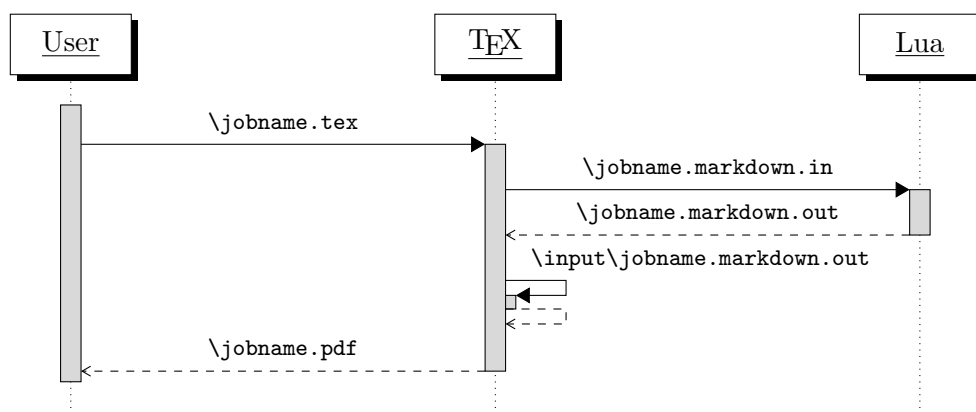
```

### 2.1.7 Command-Line Interface

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

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

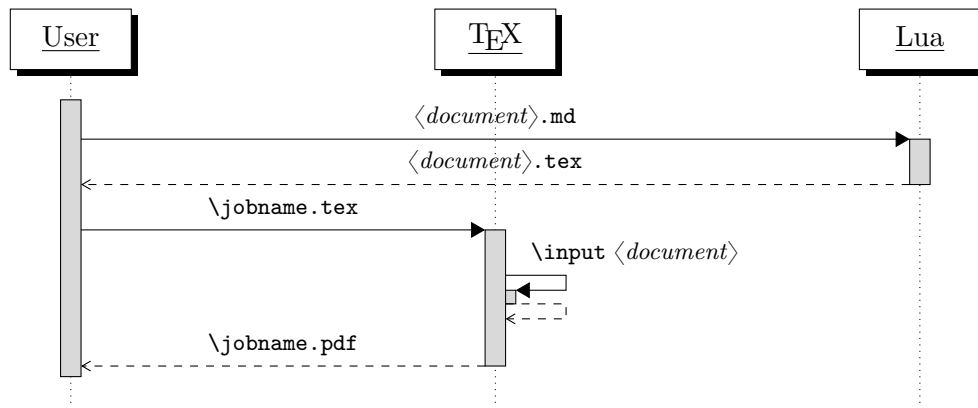
A solution to the above problems is to decouple the conversion from the typesetting. For this reason, a command-line Lua interface for converting a markdown document to T<sub>E</sub>X is also provided, see Figure 3.



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

```
729
730 local HELP_STRING = [[
731 Usage: texlua ]] .. arg[0] .. [[ [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
732 where OPTIONS are documented in the Lua interface section of the
733 technical Markdown package documentation.
734
735 When OUTPUT_FILE is unspecified, the result of the conversion will be
736 written to the standard output. When INPUT_FILE is also unspecified, the
737 result of the conversion will be read from the standard input.
738
739 Report bugs to: witiko@mail.muni.cz
740 Markdown package home page: <https://github.com/witiko/markdown>]]
741
```





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

```

742 local VERSION_STRING = [[
743 markdown-cli.lua (Markdown) ]] .. metadata.version .. [[
744
745 Copyright (C) ]] .. table.concat(metadata.copyright,
746                                     "\nCopyright (C) ") .. [[
747
748 License: ]] .. metadata.license
749
750 local function warn(s)
751   io.stderr:write("Warning: " .. s .. "\n")
752 end
753
754 local function error(s)
755   io.stderr:write("Error: " .. s .. "\n")
756   os.exit(1)
757 end
  
```

To make it easier to copy-and-paste options from Pandoc [4] such as `fancy_lists`, `header_attributes`, and `pipe_tables`, we accept snake\_case in addition to camel-Case variants of options. As a bonus, studies [5] also show that snake\_case is faster to read than camelCase.

```

758 local function camel_case(option_name)
759   local cased_option_name = option_name:gsub("_(%l)", function(match)
760     return match:sub(2, 2):upper()
761   end)
762   return cased_option_name
763 end
764
765 local function snake_case(option_name)
766   local cased_option_name = option_name:gsub("%l%u", function(match)
  
```

```

767     return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()
768   end)
769   return cased_option_name
770 end
771
772 local cases = {camel_case, snake_case}
773 local various_case_options = {}
774 for option_name, _ in pairs(defaultOptions) do
775   for _, case in ipairs(cases) do
776     various_case_options[case(option_name)] = option_name
777   end
778 end
779
780 local process_options = true
781 local options = {}
782 local input_filename
783 local output_filename
784 for i = 1, #arg do
785   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.

```

786     if arg[i] == "--" then
787       process_options = false
788       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 2.1.3.

```

789     elseif arg[i]:match("=") then
790       local key, value = arg[i]:match("(.)=(.*)")
791       if defaultOptions[key] == nil and
792         various_case_options[key] ~= nil then
793         key = various_case_options[key]
794       end

```

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

```

795     local default_type = type(defaultOptions[key])
796     if default_type == "boolean" then
797       options[key] = (value == "true")
798     elseif default_type == "number" then
799       options[key] = tonumber(value)
800     elseif default_type == "table" then
801       options[key] = {}
802       for item in value:gmatch("[^,]+") do

```

```

803         table.insert(options[key], item)
804     end
805     else
806         if default_type ~= "string" then
807             if default_type == "nil" then
808                 warn('Option "' .. key .. '" not recognized.')
809             else
810                 warn('Option "' .. key .. '" type not recognized, ' ..
811                     'please file a report to the package maintainer.')
812             end
813             warn('Parsing the ' .. 'value "' .. value ..'" of option "' ..
814                 key .. '" as a string.')
815         end
816         options[key] = value
817     end
818     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.

```

819     elseif arg[i] == "--help" or arg[i] == "-h" then
820         print(HELP_STRING)
821         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.

```

822     elseif arg[i] == "--version" or arg[i] == "-v" then
823         print(VERSION_STRING)
824         os.exit()
825     end
826 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  $\text{T}_{\text{E}}\text{X}$  document.

```

827 if input_filename == nil then
828     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  $\text{T}_{\text{E}}\text{X}$  document that will result from the conversion.

```

829 elseif output_filename == nil then
830     output_filename = arg[i]
831 else
832     error('Unexpected argument: "' .. arg[i] .. "'.')
833 end
834 ::continue::

```

```
835 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 2.1.3) used during the conversion from markdown to plain T<sub>E</sub>X and for changing the way markdown the tokens are rendered.

```
836 \def\markdownLastModified{((LASTMODIFIED))}%
```

```
837 \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 `\inputting` the file.

### 2.2.1 Typesetting Markdown

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

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

```
838 \let\markdownBegin\relax
```

```
839 \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 [6, 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
```

You can use the `\markinline` macro to input inline markdown content.

```
840 \let\markinline\relax
```

The following example plain T<sub>E</sub>X code showcases the usage of the `\markinline` macro:

```
\input markdown
\markinline{ _Hello_ **world** }
\bye
```

The above code has the same effect as the below code:

```
\input markdown
\markdownSetup{contentLevel=inline}
```

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

The `\markinline` macro is subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

You can use the `\markdownInput` macro to include markdown documents, similarly to how you might use the `\input` T<sub>E</sub>X primitive to include T<sub>E</sub>X documents. The `\markdownInput` macro accepts a single parameter with 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.

```
841 \let\markdownInput\relax
```

This macro is not subject to the 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
```

The `\markdownEscape` macro accepts a single parameter with the filename of a T<sub>E</sub>X document and executes the T<sub>E</sub>X document in the middle of a markdown document fragment. Unlike the `\input` built-in of T<sub>E</sub>X, `\markdownEscape` guarantees that the standard catcode regime of your T<sub>E</sub>X format will be used.

```
842 \let\markdownEscape\relax
```

## 2.2.2 Options

The plain T<sub>E</sub>X options are represented by T<sub>E</sub>X commands. Some of them map directly to the options recognized by the Lua interface (see Section 2.1.3), while some of them are specific to the plain T<sub>E</sub>X interface.

To determine whether plain T<sub>E</sub>X is the top layer or if there are other layers above plain T<sub>E</sub>X, we take a look on whether the `\c_@@_top_layer_tl` token list has already been defined. If not, we will assume that plain T<sub>E</sub>X is the top layer.

```
843 \ExplSyntaxOn
844 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
845 \cs_generate_variant:Nn
846   \tl_const:Nn
847   { NV }
848 \tl_if_exist:NF
```

```

849 \c_@@_top_layer_tl
850 {
851   \tl_const:NV
852     \c_@@_top_layer_tl
853     \c_@@_option_layer_plain_tex_tl
854 }

```

To enable the enumeration of plain T<sub>E</sub>X options, we will maintain the `\g_@@_plain_tex_options_seq` sequence.

```
855 \seq_new:N \g_@@_plain_tex_options_seq
```

To enable the reflection of default plain T<sub>E</sub>X options and their types, we will maintain the `\g_@@_default_plain_tex_options_prop` and `\g_@@_plain_tex_option_types_prop` property lists, respectively.

```

856 \prop_new:N \g_@@_plain_tex_option_types_prop
857 \prop_new:N \g_@@_default_plain_tex_options_prop
858 \seq_gput_right:NV
859   \g_@@_option_layers_seq
860   \c_@@_option_layer_plain_tex_tl
861 \cs_new:Nn
862   \@@_add_plain_tex_option:nmn
863   {
864     \@@_add_option:Vmn
865     \c_@@_option_layer_plain_tex_tl
866     { #1 }
867     { #2 }
868     { #3 }
869   }

```

The plain T<sub>E</sub>X options may be also be specified via the `\markdownSetup` macro. Here, the plain T<sub>E</sub>X options are represented by a comma-delimited list of `<key>=<value>` pairs. For boolean options, the `=<value>` part is optional, and `<key>` will be interpreted as `<key>=true` if the `=<value>` part has been omitted. The `\markdownSetup` macro receives the options to set up as its only argument.

```

870 \cs_new:Nn
871   \@@_setup:n
872   {
873     \keys_set:nn
874       { markdown/options }
875       { #1 }
876   }
877 \cs_gset_eq:NN
878   \markdownSetup
879   \@@_setup:n

```

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.

```

880 \prg_new_conditional:Nnn
881   \@@_if_option:n
882   { TF, T, F }
883   {
884     \@@_get_option_type:nN
885     { #1 }
886     \l_tmpa_tl
887     \str_if_eq:NNF
888     \l_tmpa_tl
889     \c_@@_option_type_boolean_tl
890     {
891       \msg_error:nxxx
892       { markdown }
893       { expected-boolean-option }
894       { #1 }
895       { \l_tmpa_tl }
896     }
897     \@@_get_option_value:nN
898     { #1 }
899     \l_tmpa_tl
900     \str_if_eq:NNTF
901     \l_tmpa_tl
902     \c_@@_option_value_true_tl
903     { \prg_return_true: }
904     { \prg_return_false: }
905   }
906 \msg_new:nnn
907   { markdown }
908   { expected-boolean-option }
909   {
910     Option~#1~has~type~#2,~
911     but~a~boolean~was~expected.
912   }
913 \let\markdownIfOption=\@@_if_option:nTF

```

### 2.2.2.1 Finalizing and Freezing the Cache

The `\markdownOptionFinalizeCache` option corresponds to the Lua interface `finalizeCache` option, which creates an output file `frozenCacheFileName` (frozen cache) that contains a mapping between an enumeration of the markdown documents in the plain T<sub>E</sub>X document and their auxiliary files cached in the `cacheDir` directory.

The `\markdownOptionFrozenCache` option uses the mapping previously created by the `finalizeCache` option, and uses it to typeset the plain T<sub>E</sub>X document without invoking Lua. 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. It defaults to `false`.



```

914 \@@_add_plain_tex_option:nnn
915   { frozenCache }
916   { boolean }
917   { 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 `finalizeCache` option.
4. Typeset the plain  $\TeX$  document to populate and finalize the cache.
5. Enable the `frozenCache` option.
6. Publish the source code of the plain  $\TeX$  document and the `cacheDir` directory.

**2.2.2.2 File and Directory Names** The `\markdownOptionInputTempFileName` macro sets the filename of the temporary input file that is created during the buffering of markdown text from a  $\TeX$  source. It defaults to `\jobname.markdown.in`.

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

```

918 \@@_add_plain_tex_option:nnn
919   { inputTempFileName }
920   { path }
921   { \jobname.markdown.in }

```

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 `.` or, since  $\TeX$  Live 2024, to the value of the `-output-directory` option of your  $\TeX$  engine.

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 `inputTempFileName` macro.

The `\markdownOptionOutputDir` macro has been deprecated and will be removed in the next major version of the Markdown package.

```

922 \@@_add_plain_tex_option:nnn
923   { outputDir }
924   { path }
925   { . }

```

### 2.2.2.3 No default token renderer prototypes

The Markdown package provides default definitions for token renderer prototypes using the `witiko/markdown/defaults` theme (see Section `sec:#themes`). Although these default definitions provide a useful starting point for authors, they use extra

resources, especially with higher-level T<sub>E</sub>X formats such as L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt. Furthermore, the default definitions may change at any time, which may pose a problem for maintainers of Markdown themes and templates who may require a stable output.

The `\markdownOptionPlain` macro specifies whether higher-level T<sub>E</sub>X formats should only use the plain T<sub>E</sub>X default definitions or whether they should also use the format-specific default definitions. Whereas plain T<sub>E</sub>X default definitions only provide definitions for simple elements such as emphasis, strong emphasis, and paragraph separators, format-specific default definitions add support for more complex elements such as lists, tables, and citations. On the flip side, plain T<sub>E</sub>X default definitions load no extra resources and are rather stable, whereas format-specific default definitions load extra resources and are subject to a more rapid change.

Here is how you would enable the macro in a L<sup>A</sup>T<sub>E</sub>X document:

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

Here is how you would enable the macro in a ConT<sub>E</sub>Xt document:

```
\def\markdownOptionPlain{true}  
\usemodule[t][markdown]
```

The macro must be set before or during the loading of the package. Setting the macro after loading the package has no effect.

```
926 \@_add_plain_tex_option:nnn  
927   { plain }  
928   { boolean }  
929   { false }
```

The `\markdownOptionNoDefaults` macro specifies whether we should prevent the loading of default definitions or not. This is useful in contexts, where we want to have total control over how all elements are rendered.

Here is how you would enable the macro in a L<sup>A</sup>T<sub>E</sub>X document:

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

Here is how you would enable the macro in a ConT<sub>E</sub>Xt document:

```
\def\markdownOptionNoDefaults{true}  
\usemodule[t][markdown]
```

The macro must be set before or during the loading of the package. Setting the macro after loading the package has no effect.

```
930 \@@_add_plain_tex_option:nnn
931   { noDefaults }
932   { boolean }
933   { false }
```

#### 2.2.2.4 Miscellaneous Options

The `\markdownOptionStripPercentSigns` macro controls whether a percent sign (%) at the beginning of a line will be discarded when buffering Markdown input (see sections 3.2.5 and 3.2.6) 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 [7] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```
934 \seq_gput_right:Nn
935   \g_@@_plain_tex_options_seq
936   { stripPercentSigns }
937 \prop_gput:Nnn
938   \g_@@_plain_tex_option_types_prop
939   { stripPercentSigns }
940   { boolean }
941 \prop_gput:Nnx
942   \g_@@_default_plain_tex_options_prop
943   { stripPercentSigns }
944   { false }
```

#### 2.2.2.5 Generating Plain T<sub>E</sub>X Option Macros and Key-Values

We define the command `\@@_define_option_commands_and_keyvals:` that defines plain T<sub>E</sub>X macros and the key-value interface of the `\markdownSetup` macro for the above plain T<sub>E</sub>X options.

The command also defines macros and key-values that map directly to the options recognized by the Lua interface, such as `\markdownOptionHybrid` for the `hybrid` Lua option (see Section 2.1.3), which are not processed by the plain T<sub>E</sub>X implementation, only passed along to Lua.

Furthermore, the command also defines options and key-values for subsequently loaded layers that correspond to higher-level T<sub>E</sub>X formats such as L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt.

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 `inputTempFileName` macro.

```
945 \cs_new:Nn
946   \@@_define_option_commands_and_keyvals:
947   {
948     \seq_map_inline:Nn
```

```

949     \g_@@_option_layers_seq
950     {
951         \seq_map_inline:cn
952         { g_@@_ ##1 _options_seq }
953         {
954             \@@_define_option_command:n
955             { ####1 }

```

To make it easier to copy-and-paste options from Pandoc [4] such as `fancy_lists`, `header_attributes`, and `pipe_tables`, we accept snake\_case in addition to camel-Case variants of options. As a bonus, studies [5] also show that snake\_case is faster to read than camelCase.

```

956         \@@_with_various_cases:n
957         { ####1 }
958         {
959             \@@_define_option_keyval:nnn
960             { ##1 }
961             { ####1 }
962             { #####1 }
963         }
964     }
965 }
966 }
967 \cs_new:Nn
968 \@@_define_option_command:n
969 {

```

Use the `lt3luabridge` library to determine the default value of the `\markdownOptionOutputDir` macro by using the environmental variable `TEXMF_OUTPUT_DIRECTORY` that is available since TeX Live 2024.

```

970     \str_if_eq:nnTF
971     { #1 }
972     { outputDir }
973     { \@@_define_option_command_output_dir: }
974     {

```

Do not override options defined before loading the package.

```

975     \@@_option_tl_to_csname:nN
976     { #1 }
977     \l_tmpa_tl
978     \cs_if_exist:cF
979     { \l_tmpa_tl }
980     {
981         \@@_get_default_option_value:nN
982         { #1 }
983         \l_tmpa_tl
984         \@@_set_option_value:nV
985         { #1 }

```

```

986         \l_tmpa_tl
987     }
988 }
989 }
990 \ExplSyntaxOff
991 \input lt3luabridge.tex
992 \ExplSyntaxOn
993 \cs_new:Nn
994   \@@_define_option_command_output_dir:
995   {
996     \cs_if_free:NT
997       \markdownOptionOutputDir
998     {
999       \bool_if:nTF
1000         {
1001           \cs_if_exist_p:N
1002             \luabridge_tl_set:Nn &&
1003           (
1004             \int_compare_p:nNn
1005               { \g_luabridge_method_int }
1006               =
1007               { \c_luabridge_method_directlua_int } ||
1008             \sys_if_shell_unrestricted_p:
1009           )
1010         }
1011         {
1012           \luabridge_tl_set:Nn
1013             \l_tmpa_tl
1014             { print(os.getenv("TEXMF_OUTPUT_DIRECTORY") or ".") }
1015           \tl_gset:NV
1016             \markdownOptionOutputDir
1017             \l_tmpa_tl
1018         }
1019         {
1020           \tl_gset:Nn
1021             \markdownOptionOutputDir
1022             { . }
1023         }
1024     }
1025 }
1026 \cs_new:Nn
1027   \@@_set_option_value:nn
1028   {
1029     \@@_define_option:n
1030       { #1 }
1031     \@@_get_option_type:nN
1032       { #1 }

```

```

1033     \l_tmpa_tl
1034 \str_if_eq:NNTF
1035     \c_@@_option_type_counter_tl
1036     \l_tmpa_tl
1037     {
1038         \@@_option_tl_to_csname:nN
1039         { #1 }
1040         \l_tmpa_tl
1041         \int_gset:cn
1042         { \l_tmpa_tl }
1043         { #2 }
1044     }
1045     {
1046         \@@_option_tl_to_csname:nN
1047         { #1 }
1048         \l_tmpa_tl
1049         \cs_set:cpn
1050         { \l_tmpa_tl }
1051         { #2 }
1052     }
1053 }
1054 \cs_generate_variant:Nn
1055     \@@_set_option_value:nn
1056     { nV }
1057 \cs_new:Nn
1058     \@@_define_option:n
1059     {
1060         \@@_option_tl_to_csname:nN
1061         { #1 }
1062         \l_tmpa_tl
1063         \cs_if_free:cT
1064         { \l_tmpa_tl }
1065         {
1066             \@@_get_option_type:nN
1067             { #1 }
1068             \l_tmpb_tl
1069             \str_if_eq:NNT
1070             \c_@@_option_type_counter_tl
1071             \l_tmpb_tl
1072             {
1073                 \@@_option_tl_to_csname:nN
1074                 { #1 }
1075                 \l_tmpa_tl
1076                 \int_new:c
1077                 { \l_tmpa_tl }
1078             }
1079         }

```

```

1080 }
1081 \cs_new:Nn
1082 \@@_define_option_keyval:nnn
1083 {
1084   \prop_get:cnN
1085     { g_@@_ #1 _option_types_prop }
1086     { #2 }
1087     \l_tmpa_tl
1088   \str_if_eq:VVTF
1089     \l_tmpa_tl
1090     \c_@@_option_type_boolean_tl
1091     {
1092       \keys_define:nn
1093         { markdown/options }
1094       {

```

For boolean options, we also accept **yes** as an alias for **true** and **no** as an alias for **false**.

```

1095         #3 .code:n = {
1096           \tl_set:Nx
1097             \l_tmpa_tl
1098             {
1099               \str_case:nnF
1100                 { ##1 }
1101                 {
1102                   { yes } { true }
1103                   { no } { false }
1104                 }
1105                 { ##1 }
1106             }
1107           \@@_set_option_value:nV
1108             { #2 }
1109             \l_tmpa_tl
1110         },
1111         #3 .default:n = { true },
1112       }
1113     }
1114   {
1115     \keys_define:nn
1116       { markdown/options }
1117     {
1118       #3 .code:n = {
1119         \@@_set_option_value:nn
1120           { #2 }
1121           { ##1 }
1122       },
1123     }

```

```
1124     }
```

For options of type `clist`, we assume that  $\langle key \rangle$  is a regular English noun in plural (such as `extensions`) and we also define the  $\langle singular\ key \rangle = \langle value \rangle$  interface, where  $\langle singular\ key \rangle$  is  $\langle key \rangle$  after stripping the trailing `-s` (such as `extension`). Rather than setting the option to  $\langle value \rangle$ , this interface appends  $\langle value \rangle$  to the current value as the rightmost item in the list.

```
1125     \str_if_eq:VVT
1126         \l_tmpa_tl
1127         \c_@@_option_type_clist_tl
1128     {
1129         \tl_set:Nn
1130             \l_tmpa_tl
1131             { #3 }
1132         \tl_reverse:N
1133             \l_tmpa_tl
1134         \str_if_eq:enF
1135             {
1136                 \tl_head:V
1137                 \l_tmpa_tl
1138             }
1139             { s }
1140             {
1141                 \msg_error:nnn
1142                     { markdown }
1143                     { malformed-name-for-clist-option }
1144                     { #3 }
1145             }
1146         \tl_set:Nx
1147             \l_tmpa_tl
1148             {
1149                 \tl_tail:V
1150                 \l_tmpa_tl
1151             }
1152         \tl_reverse:N
1153             \l_tmpa_tl
1154         \tl_put_right:Nn
1155             \l_tmpa_tl
1156             {
1157                 .code:n = {
1158                     \@@_get_option_value:nN
1159                         { #2 }
1160                     \l_tmpa_tl
1161                     \clist_set:NV
1162                         \l_tmpa_clist
1163                         { \l_tmpa_tl, { ##1 } }
1164                     \@@_set_option_value:nV
```



```

1165             { #2 }
1166             \l_tmpa_clist
1167         }
1168     }
1169     \keys_define:nV
1170     { markdown/options }
1171     \l_tmpa_tl
1172 }
1173 }
1174 \cs_generate_variant:Nn
1175   \clist_set:Nn
1176   { NV }
1177 \cs_generate_variant:Nn
1178   \keys_define:nn
1179   { nV }
1180 \cs_generate_variant:Nn
1181   \@@_set_option_value:nn
1182   { nV }
1183 \prg_generate_conditional_variant:Nnn
1184   \str_if_eq:nn
1185   { en }
1186   { F }
1187 \msg_new:nnn
1188   { markdown }
1189   { malformed-name-for-clist-option }
1190   {
1191     Clist-option-name~#1~does~not~end~with~-s.
1192   }

```

If plain T<sub>E</sub>X is the top layer, we use the `\@@_define_option_commands_and_keyvals:` macro to define plain T<sub>E</sub>X option macros and key-values immediately. Otherwise, we postpone the definition until the upper layers have been loaded.

```

1193 \str_if_eq:VVT
1194   \c_@@_top_layer_tl
1195   \c_@@_option_layer_plain_tex_tl
1196   {
1197     \@@_define_option_commands_and_keyvals:
1198   }
1199 \ExplSyntaxOff

```

### 2.2.3 Themes

User-defined themes for the Markdown package provide a domain-specific interpretation of Markdown tokens. Themes allow the authors to achieve a specific look and other high-level goals without low-level programming.

The key-values `theme=<theme name>` and `import=<theme name>` load a T<sub>E</sub>X document (further referred to as *a theme*) named `markdowntheme<munged theme`

*name*⟩.tex, where the *munged theme name* is the *theme name* after the substitution of all forward slashes (/) for an underscore (\_). The theme name is *qualified* and contains no underscores. A theme name 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 [8, Section 15.1].

Theme names must be qualified to minimize naming conflicts between different themes with a similar purpose. The preferred format of a theme name is  $\langle theme\ author\rangle/\langle theme\ purpose\rangle/\langle private\ naming\ scheme\rangle$ , 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 to allow structure inside theme names without dictating where the themes should be located inside the T<sub>E</sub>X directory structure. For example, loading a theme named `witiko/beamer/MU` would load a T<sub>E</sub>X document package named `markdownthemewitiko_beamer_MU.tex`.

```

1200 \ExplSyntaxOn
1201 \keys_define:nn
1202   { markdown/options }
1203   {
1204     theme .code:n = {
1205       \@@_set_theme:n
1206         { #1 }
1207     },
1208     import .code:n = {
1209       \tl_set:Nn
1210         \l_tmpa_tl
1211         { #1 }

```

To ensure that keys containing forward slashes get passed correctly, we replace all forward slashes in the input with backslash tokens with category code letter and then undo the replacement. This means that if any unbraced backslash tokens with category code letter exist in the input, they will be replaced with forward slashes. However, this should be extremely rare.

```

1212     \tl_replace_all:NnV
1213       \l_tmpa_tl
1214       { / }
1215     \c_backslash_str
1216     \keys_set:nV
1217       { markdown/options/import }
1218     \l_tmpa_tl
1219   },
1220 }

```

To keep track of the current theme when themes are nested, we will maintain the `\g_@@_themes_seq` stack of theme names. For convenience, the name of the current theme is also available in the `\g_@@_current_theme_tl` macro.

```

1221 \seq_new:N
1222   \g_@@_themes_seq
1223 \tl_new:N
1224   \g_@@_current_theme_tl
1225 \tl_gset:Nn
1226   \g_@@_current_theme_tl
1227   { }
1228 \seq_gput_right:NV
1229   \g_@@_themes_seq
1230   \g_@@_current_theme_tl
1231 \cs_new:Nn
1232   \@@_set_theme:n
1233   {

```

First, we validate the theme name.

```

1234   \str_if_in:nnF
1235     { #1 }
1236     { / }
1237     {
1238       \msg_error:nnn
1239         { markdown }
1240         { unqualified-theme-name }
1241         { #1 }
1242     }
1243   \str_if_in:nnT
1244     { #1 }
1245     { _ }
1246     {
1247       \msg_error:nnn
1248         { markdown }
1249         { underscores-in-theme-name }
1250         { #1 }
1251     }

```

Next, we munge the theme name.

```

1252   \str_set:Nn
1253     \l_tmpa_str
1254     { #1 }
1255   \str_replace_all:Nnn
1256     \l_tmpa_str
1257     { / }
1258     { _ }

```

Finally, we load the theme.

```

1259   \tl_gset:Nn
1260     \g_@@_current_theme_tl
1261     { #1 / }
1262   \seq_gput_right:NV

```

```

1263     \g_@@_themes_seq
1264     \g_@@_current_theme_tl
1265     \@@_load_theme:nV
1266     { #1 }
1267     \l_tmpa_str
1268     \seq_gpop_right:NN
1269     \g_@@_themes_seq
1270     \l_tmpa_tl
1271     \seq_get_right:NN
1272     \g_@@_themes_seq
1273     \l_tmpa_tl
1274     \tl_gset:NV
1275     \g_@@_current_theme_tl
1276     \l_tmpa_tl
1277   }
1278 \msg_new:nnnn
1279   { markdown }
1280   { unqualified-theme-name }
1281   { Won't load theme with unqualified name #1 }
1282   { Theme names must contain at least one forward slash }
1283 \msg_new:nnnn
1284   { markdown }
1285   { underscores-in-theme-name }
1286   { Won't load theme with an underscore in its name #1 }
1287   { Theme names must not contain underscores in their names }
1288 \cs_generate_variant:Nn
1289   \tl_replace_all:Nnn
1290   { NnV }
1291 \ExplSyntaxOff

```

Built-in plain T<sub>E</sub>X themes provided with the Markdown package include:

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

```


\input markdown
\markdownSetup{import=witiko/tilde}
\markdownBegin
Bartel~Leendert van~der~Waerden
\markdownEnd
\bye

```

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

**witiko/markdown/defaults** A plain T<sub>E</sub>X theme with the default definitions of token renderer prototypes for plain T<sub>E</sub>X. This theme is loaded automatically together with the package and explicitly loading it has no effect.

Please, see Section 3.2.2 for implementation details of the built-in plain T<sub>E</sub>X themes.

## 2.2.4 Snippets

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

```
1292 \ExplSyntaxOn
1293 \prop_new:N
1294   \g_@@_snippets_prop
1295 \cs_new:Nn
1296   \@@_setup_snippet:nn
1297   {
1298     \tl_if_empty:nT
1299       { #1 }
1300     {
1301       \msg_error:nnn
1302         { markdown }
1303         { empty-snippet-name }
1304         { #1 }
1305     }
1306     \tl_set:NV
1307       \l_tmpa_tl
1308       \g_@@_current_theme_tl
1309     \tl_put_right:Nn
1310       \l_tmpa_tl
1311       { #1 }
1312     \@@_if_snippet_exists:nT
1313       { #1 }
1314     {
1315       \msg_warning:nnV
1316         { markdown }
1317         { redefined-snippet }
1318         \l_tmpa_tl
1319     }
1320     \keys_precompile:nnN
1321       { markdown/options }
1322       { #2 }
1323     \l_tmpb_tl
1324     \prop_gput:NVV
1325       \g_@@_snippets_prop
```

```

1326     \l_tmpa_tl
1327     \l_tmpb_tl
1328   }
1329 \cs_gset_eq:NN
1330   \markdownSetupSnippet
1331   \@_setup_snippet:nn
1332 \msg_new:nnnn
1333   { markdown }
1334   { empty-snippet-name }
1335   { Empty-snippet-name~#1 }
1336   { Pick-a-non-empty-name-for-your-snippet }
1337 \msg_new:nnn
1338   { markdown }
1339   { redefined-snippet }
1340   { Redefined~snippet~#1 }

```

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

```

1341 \prg_new_conditional:Nnn
1342   \@_if_snippet_exists:n
1343   { TF, T, F }
1344   {
1345     \tl_set:NV
1346       \l_tmpa_tl
1347       \g_@_current_theme_tl
1348     \tl_put_right:Nn
1349       \l_tmpa_tl
1350       { #1 }
1351     \prop_get:NVNTF
1352       \g_@_snippets_prop
1353       \l_tmpa_tl
1354       \l_tmpb_tl
1355       { \prg_return_true: }
1356       { \prg_return_false: }
1357   }
1358 \cs_gset_eq:NN
1359   \markdownIfSnippetExists
1360   \@_if_snippet_exists:nTF

```

The option with key `snippet` invokes a snippet named  $\langle value \rangle$ .

```

1361 \keys_define:nn
1362   { markdown/options }
1363   {
1364     snippet .code:n = {
1365       \tl_set:NV
1366         \l_tmpa_tl
1367         \g_@_current_theme_tl
1368       \tl_put_right:Nn

```

```

1369     \l_tmpa_tl
1370     { #1 }
1371     \@@_if_snippet_exists:nTF
1372     { #1 }
1373     {
1374         \prop_get:NVN
1375         \g_@@_snippets_prop
1376         \l_tmpa_tl
1377         \l_tmpb_tl
1378         \tl_use:N
1379         \l_tmpb_tl
1380     }
1381     {
1382         \msg_error:nnV
1383         { markdown }
1384         { undefined-snippet }
1385         \l_tmpa_tl
1386     }
1387 }
1388 }
1389 \msg_new:nnn
1390 { markdown }
1391 { undefined-snippet }
1392 { Can't invoke undefined snippet ~#1 }
1393 \ExplSyntaxOff

```

Here is how we can use snippets to store options and invoke them later in  $\text{\LaTeX}$ :

```

\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}
```

If the `romanNumerals` snippet were defined in the `jdooe/lists` theme, we could import the `jdooe/lists` theme and use the qualified name `jdooe/lists/romanNumerals` to invoke the snippet:

```
\markdownSetup{import=jdooe/lists}
\begin{markdown}[snippet=jdooe/lists/romanNumerals]
```

The following ordered list will be preceded by roman numerals:

```
3. tres
4. quattuor
```

```
\end{markdown}
```

Alternatively, we can use the extended variant of the `import`  $\LaTeX$  option that allows us to import the `romanNumerals` snippet to the current namespace for easier access:

```
\markdownSetup{
  import = {
    jdooe/lists = romanNumerals,
  },
}
\begin{markdown}[snippet=romanNumerals]
```

The following ordered list will be preceded by roman numerals:

```
3. tres
4. quattuor
```

```
\end{markdown}
```

Furthermore, we can also specify the name of the snippet in the current namespace, which can be different from the name of the snippet in the `jdooe/lists` theme. For example, we can make the snippet `jdooe/lists/romanNumerals` available under the name `roman`.



```

\markdownSetup{
  import = {
    jdoe/lists = romanNumerals as roman,
  },
}
\begin{markdown}[snippet=roman]

```

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```

\end{markdown}

```

Several themes and/or snippets can be loaded at once using the extended variant of the `import` L<sup>A</sup>T<sub>E</sub>X option:

```

\markdownSetup{
  import = {
    jdoe/longpackagename/lists = {
      arabic as arabic1,
      roman,
      alphabetic,
    },
    jdoe/anotherlongpackagename/lists = {
      arabic as arabic2,
    },
    jdoe/yetanotherlongpackagename,
  },
}

```

```

1394 \ExplSyntaxOn
1395 \tl_new:N
1396   \l_@@_import_current_theme_tl
1397 \keys_define:nn
1398   { markdown/options/import }
1399   {

```

If a theme name is given without a list of snippets to import, we assume that an empty list was given.

```

1400   unknown .default:n = {},
1401   unknown .code:n = {

```

To ensure that keys containing forward slashes get passed correctly, we replace all forward slashes in the input with backslash tokens with category code letter and then undo the replacement. This means that if any unbraced backslash tokens with category code letter exist in the input, they will be replaced with forward slashes. However, this should be extremely rare.

```

1402     \tl_set_eq:NN
1403     \l_@@_import_current_theme_tl
1404     \l_keys_key_str
1405     \tl_replace_all:NVn
1406     \l_@@_import_current_theme_tl
1407     \c_backslash_str
1408     { / }

```

Here, we import the snippets.

```

1409     \clist_map_inline:nn
1410     { #1 }
1411     {
1412         \regex_extract_once:nnNTF
1413         { ^(.*)\s+as\s+(.*)$ }
1414         { ##1 }
1415         \l_tmpa_seq
1416         {
1417             \seq_pop:NN
1418             \l_tmpa_seq
1419             \l_tmpa_tl
1420             \seq_pop:NN
1421             \l_tmpa_seq
1422             \l_tmpa_tl
1423             \seq_pop:NN
1424             \l_tmpa_seq
1425             \l_tmpb_tl
1426         }
1427         {
1428             \tl_set:Nn
1429             \l_tmpa_tl
1430             { ##1 }
1431             \tl_set:Nn
1432             \l_tmpb_tl
1433             { ##1 }
1434         }
1435         \tl_put_left:Nn
1436         \l_tmpa_tl
1437         { / }
1438         \tl_put_left:NV
1439         \l_tmpa_tl
1440         \l_@@_import_current_theme_tl
1441         \@@_setup_snippet:Vx

```

```

1442         \l_tmpb_tl
1443         { snippet = { \l_tmpa_tl } }
1444     }

```

Here, we load the theme.

```

1445     \@@_set_theme:V
1446     \l_@@_import_current_theme_tl
1447 },
1448 }
1449 \cs_generate_variant:Nn
1450 \tl_replace_all:Nnn
1451 { NVn }
1452 \cs_generate_variant:Nn
1453 \@@_set_theme:n
1454 { V }
1455 \cs_generate_variant:Nn
1456 \@@_setup_snippet:nn
1457 { Vx }

```

## 2.2.5 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 2.2.6).

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

```

1458 \ExplSyntaxOn
1459 \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.

```

1460 \prop_new:N \g_@@_renderer_arities_prop
1461 \ExplSyntaxOff

```

### 2.2.5.1 Attribute Renderers

The following macros are only produced, when at least one of the following options for markdown attributes on different elements is enabled:

- `autoIdentifiers`
- `fencedCodeAttributes`
- `gfmAutoIdentifiers`
- `headerAttributes`
- `inlineCodeAttributes`
- `linkAttributes`

`\markdownRendererAttributeIdentifier` represents the  $\langle identifier \rangle$  of a markdown element (`id="⟨identifier⟩"` in HTML and `#⟨identifier⟩` in markdown attributes). 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 attributes). 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.

```

1462 \def\markdownRendererAttributeIdentifier{%
1463   \markdownRendererAttributeIdentifierPrototype}%
1464 \ExplSyntaxOn
1465 \seq_gput_right:Nn
1466   \g_@@_renderers_seq
1467   { attributeIdentifier }
1468 \prop_gput:Nnn
1469   \g_@@_renderer_arities_prop
1470   { attributeIdentifier }
1471   { 1 }
1472 \ExplSyntaxOff
1473 \def\markdownRendererAttributeClassName{%
1474   \markdownRendererAttributeClassNamePrototype}%
1475 \ExplSyntaxOn
1476 \seq_gput_right:Nn
1477   \g_@@_renderers_seq
1478   { attributeClassName }
1479 \prop_gput:Nnn
1480   \g_@@_renderer_arities_prop
1481   { attributeClassName }
1482   { 1 }
1483 \ExplSyntaxOff
1484 \def\markdownRendererAttributeKeyValue{%
1485   \markdownRendererAttributeKeyValuePrototype}%
1486 \ExplSyntaxOn
1487 \seq_gput_right:Nn
1488   \g_@@_renderers_seq
1489   { attributeKeyValue }
1490 \prop_gput:Nnn
1491   \g_@@_renderer_arities_prop
1492   { attributeKeyValue }
1493   { 2 }
1494 \ExplSyntaxOff

```

### 2.2.5.2 Block Quote Renderers

The `\markdownRendererBlockQuoteBegin` macro represents the beginning of a block quote. The macro receives no arguments.

```
1495 \def\markdownRendererBlockQuoteBegin{%
1496   \markdownRendererBlockQuoteBeginPrototype}%
1497 \ExplSyntaxOn
1498 \seq_gput_right:Nn
1499   \g_@@_renderers_seq
1500   { blockQuoteBegin }
1501 \prop_gput:Nnn
1502   \g_@@_renderer_arities_prop
1503   { blockQuoteBegin }
1504   { 0 }
1505 \ExplSyntaxOff
```

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

```
1506 \def\markdownRendererBlockQuoteEnd{%
1507   \markdownRendererBlockQuoteEndPrototype}%
1508 \ExplSyntaxOn
1509 \seq_gput_right:Nn
1510   \g_@@_renderers_seq
1511   { blockQuoteEnd }
1512 \prop_gput:Nnn
1513   \g_@@_renderer_arities_prop
1514   { blockQuoteEnd }
1515   { 0 }
1516 \ExplSyntaxOff
```

### 2.2.5.3 Bracketed Spans Attribute Context Renderers

The following macros are only produced, when the `bracketedSpans` option is enabled.

The `\markdownRendererBracketedSpanAttributeContextBegin` and `\markdownRendererBracketedSpanAttributeContextEnd` macros represent the beginning and the end of a context in which the attributes of an inline bracketed span apply. The macros receive no arguments.

```
1517 \def\markdownRendererBracketedSpanAttributeContextBegin{%
1518   \markdownRendererBracketedSpanAttributeContextBeginPrototype}%
1519 \ExplSyntaxOn
1520 \seq_gput_right:Nn
1521   \g_@@_renderers_seq
1522   { bracketedSpanAttributeContextBegin }
1523 \prop_gput:Nnn
1524   \g_@@_renderer_arities_prop
1525   { bracketedSpanAttributeContextBegin }
1526   { 0 }
1527 \ExplSyntaxOff
```

```

1528 \def\markdownRendererBracketedSpanAttributeContextEnd{%
1529   \markdownRendererBracketedSpanAttributeContextEndPrototype}%
1530 \ExplSyntaxOn
1531 \seq_gput_right:Nn
1532   \g_@@_renderers_seq
1533   { bracketedSpanAttributeContextEnd }
1534 \prop_gput:Nnn
1535   \g_@@_renderer_arities_prop
1536   { bracketedSpanAttributeContextEnd }
1537   { 0 }
1538 \ExplSyntaxOff

```

#### 2.2.5.4 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.

```

1539 \def\markdownRendererUlBegin{%
1540   \markdownRendererUlBeginPrototype}%
1541 \ExplSyntaxOn
1542 \seq_gput_right:Nn
1543   \g_@@_renderers_seq
1544   { ulBegin }
1545 \prop_gput:Nnn
1546   \g_@@_renderer_arities_prop
1547   { ulBegin }
1548   { 0 }
1549 \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.

```

1550 \def\markdownRendererUlBeginTight{%
1551   \markdownRendererUlBeginTightPrototype}%
1552 \ExplSyntaxOn
1553 \seq_gput_right:Nn
1554   \g_@@_renderers_seq
1555   { ulBeginTight }
1556 \prop_gput:Nnn
1557   \g_@@_renderer_arities_prop
1558   { ulBeginTight }
1559   { 0 }
1560 \ExplSyntaxOff

```

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

```

1561 \def\markdownRendererUListItem{%
1562   \markdownRendererUListItemPrototype}%
1563 \ExplSyntaxOn
1564 \seq_gput_right:Nn
1565   \g_@@_renderers_seq
1566   { ulItem }
1567 \prop_gput:Nnn
1568   \g_@@_renderer_arities_prop
1569   { ulItem }
1570   { 0 }
1571 \ExplSyntaxOff

```

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

```

1572 \def\markdownRendererUListItemEnd{%
1573   \markdownRendererUListItemEndPrototype}%
1574 \ExplSyntaxOn
1575 \seq_gput_right:Nn
1576   \g_@@_renderers_seq
1577   { ulItemEnd }
1578 \prop_gput:Nnn
1579   \g_@@_renderer_arities_prop
1580   { ulItemEnd }
1581   { 0 }
1582 \ExplSyntaxOff

```

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

```

1583 \def\markdownRendererUListEnd{%
1584   \markdownRendererUListEndPrototype}%
1585 \ExplSyntaxOn
1586 \seq_gput_right:Nn
1587   \g_@@_renderers_seq
1588   { ulEnd }
1589 \prop_gput:Nnn
1590   \g_@@_renderer_arities_prop
1591   { ulEnd }
1592   { 0 }
1593 \ExplSyntaxOff

```

The `\markdownRendererUListEndTight` 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.

```

1594 \def\markdownRendererUListEndTight{%

```

```

1595 \markdownRendererUllEndTightPrototype}%
1596 \ExplSyntaxOn
1597 \seq_gput_right:Nn
1598 \g_@@_renderers_seq
1599 { ulEndTight }
1600 \prop_gput:Nnn
1601 \g_@@_render_arity_prop
1602 { ulEndTight }
1603 { 0 }
1604 \ExplSyntaxOff

```

### 2.2.5.5 Citation Renderers

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.

```

1605 \def\markdownRendererCite{%
1606 \markdownRendererCitePrototype}%
1607 \ExplSyntaxOn
1608 \seq_gput_right:Nn
1609 \g_@@_renderers_seq
1610 { cite }
1611 \prop_gput:Nnn
1612 \g_@@_render_arity_prop
1613 { cite }
1614 { 1 }
1615 \ExplSyntaxOff

```

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.

```

1616 \def\markdownRendererTextCite{%
1617 \markdownRendererTextCitePrototype}%
1618 \ExplSyntaxOn
1619 \seq_gput_right:Nn
1620 \g_@@_renderers_seq
1621 { textCite }
1622 \prop_gput:Nnn
1623 \g_@@_render_arity_prop
1624 { textCite }
1625 { 1 }
1626 \ExplSyntaxOff

```



### 2.2.5.6 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.

```
1627 \def\markdownRendererInputVerbatim{%
1628   \markdownRendererInputVerbatimPrototype}%
1629 \ExplSyntaxOn
1630 \seq_gput_right:Nn
1631   \g_@@_renderers_seq
1632   { inputVerbatim }
1633 \prop_gput:Nnn
1634   \g_@@_renderer_arities_prop
1635   { inputVerbatim }
1636   { 1 }
1637 \ExplSyntaxOff
```

The `\markdownRendererInputFencedCode` macro represents a fenced code block. This macro will only be produced, when the `fencedCode` option is enabled. The macro receives three arguments that correspond to the filename of a file containing the code block contents, the fully escaped code fence infostring that can be directly typeset, and the raw code fence infostring that can be used outside typesetting.

```
1638 \def\markdownRendererInputFencedCode{%
1639   \markdownRendererInputFencedCodePrototype}%
1640 \ExplSyntaxOn
1641 \seq_gput_right:Nn
1642   \g_@@_renderers_seq
1643   { inputFencedCode }
1644 \prop_gput:Nnn
1645   \g_@@_renderer_arities_prop
1646   { inputFencedCode }
1647   { 3 }
1648 \ExplSyntaxOff
```

### 2.2.5.7 Code Span Renderer

The `\markdownRendererCodeSpan` macro represents inline code span in the input text. It receives a single argument that corresponds to the inline code span.

```
1649 \def\markdownRendererCodeSpan{%
1650   \markdownRendererCodeSpanPrototype}%
1651 \ExplSyntaxOn
1652 \seq_gput_right:Nn
1653   \g_@@_renderers_seq
1654   { codeSpan }
1655 \prop_gput:Nnn
1656   \g_@@_renderer_arities_prop
1657   { codeSpan }
```

```

1658 { 1 }
1659 \ExplSyntaxOff

```

### 2.2.5.8 Code Span Attribute Context Renderers

The following macros are only produced, when the `inlineCodeAttributes` option is enabled.

The `\markdownRendererCodeSpanAttributeContextBegin` and `\markdownRendererCodeSpanAttributeContextEnd` macros represent the beginning and the end of a context in which the attributes of an inline code span apply. The macros receive no arguments.

```

1660 \def\markdownRendererCodeSpanAttributeContextBegin{%
1661   \markdownRendererCodeSpanAttributeContextBeginPrototype}%
1662 \ExplSyntaxOn
1663 \seq_gput_right:Nn
1664   \g_@@_renderers_seq
1665   { codeSpanAttributeContextBegin }
1666 \prop_gput:Nnn
1667   \g_@@_renderer_arities_prop
1668   { codeSpanAttributeContextBegin }
1669   { 0 }
1670 \ExplSyntaxOff
1671 \def\markdownRendererCodeSpanAttributeContextEnd{%
1672   \markdownRendererCodeSpanAttributeContextEndPrototype}%
1673 \ExplSyntaxOn
1674 \seq_gput_right:Nn
1675   \g_@@_renderers_seq
1676   { codeSpanAttributeContextEnd }
1677 \prop_gput:Nnn
1678   \g_@@_renderer_arities_prop
1679   { codeSpanAttributeContextEnd }
1680   { 0 }
1681 \ExplSyntaxOff

```

### 2.2.5.9 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.

```

1682 \def\markdownRendererContentBlock{%
1683   \markdownRendererContentBlockPrototype}%
1684 \ExplSyntaxOn
1685 \seq_gput_right:Nn
1686   \g_@@_renderers_seq
1687   { contentBlock }
1688 \prop_gput:Nnn

```

```

1689 \g_@@_renderer_arities_prop
1690 { contentBlock }
1691 { 4 }
1692 \ExplSyntaxOff

```

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

```

1693 \def\markdownRendererContentBlockOnlineImage{%
1694 \markdownRendererContentBlockOnlineImagePrototype}%
1695 \ExplSyntaxOn
1696 \seq_gput_right:Nn
1697 \g_@@_renderers_seq
1698 { contentBlockOnlineImage }
1699 \prop_gput:Nnn
1700 \g_@@_renderer_arities_prop
1701 { contentBlockOnlineImage }
1702 { 4 }
1703 \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>32</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  $v$ . The macro receives five arguments: the local file name extension  $s$  cast to the lower case, the language  $v$ , the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the content block.

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

```

1704 \def\markdownRendererContentBlockCode{%
1705 \markdownRendererContentBlockCodePrototype}%
1706 \ExplSyntaxOn
1707 \seq_gput_right:Nn
1708 \g_@@_renderers_seq
1709 { contentBlockCode }
1710 \prop_gput:Nnn
1711 \g_@@_renderer_arities_prop

```

---

<sup>32</sup>Filenames other than `markdown-languages.json` may be specified using the `contentBlocksLanguageMap` Lua option.

```

1712 { contentBlockCode }
1713 { 5 }
1714 \ExplSyntaxOff

```

### 2.2.5.10 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.

```

1715 \def\markdownRendererDlBegin{%
1716   \markdownRendererDlBeginPrototype}%
1717 \ExplSyntaxOn
1718 \seq_gput_right:Nn
1719   \g_@@_renderers_seq
1720   { dlBegin }
1721 \prop_gput:Nnn
1722   \g_@@_renderer_arities_prop
1723   { dlBegin }
1724   { 0 }
1725 \ExplSyntaxOff

```

The `\markdownRendererDlBeginTight` macro represents the beginning 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.

```

1726 \def\markdownRendererDlBeginTight{%
1727   \markdownRendererDlBeginTightPrototype}%
1728 \ExplSyntaxOn
1729 \seq_gput_right:Nn
1730   \g_@@_renderers_seq
1731   { dlBeginTight }
1732 \prop_gput:Nnn
1733   \g_@@_renderer_arities_prop
1734   { dlBeginTight }
1735   { 0 }
1736 \ExplSyntaxOff

```

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

```

1737 \def\markdownRendererDlItem{%
1738   \markdownRendererDlItemPrototype}%
1739 \ExplSyntaxOn
1740 \seq_gput_right:Nn
1741   \g_@@_renderers_seq

```

```

1742 { dlItem }
1743 \prop_gput:Nnn
1744 \g_@@_renderer_arities_prop
1745 { dlItem }
1746 { 1 }
1747 \ExplSyntaxOff

```

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

```

1748 \def\markdownRendererDlItemEnd{%
1749 \markdownRendererDlItemEndPrototype}%
1750 \ExplSyntaxOn
1751 \seq_gput_right:Nn
1752 \g_@@_renderers_seq
1753 { dlItemEnd }
1754 \prop_gput:Nnn
1755 \g_@@_renderer_arities_prop
1756 { dlItemEnd }
1757 { 0 }
1758 \ExplSyntaxOff

```

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

```

1759 \def\markdownRendererDlDefinitionBegin{%
1760 \markdownRendererDlDefinitionBeginPrototype}%
1761 \ExplSyntaxOn
1762 \seq_gput_right:Nn
1763 \g_@@_renderers_seq
1764 { dlDefinitionBegin }
1765 \prop_gput:Nnn
1766 \g_@@_renderer_arities_prop
1767 { dlDefinitionBegin }
1768 { 0 }
1769 \ExplSyntaxOff

```

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

```

1770 \def\markdownRendererDlDefinitionEnd{%
1771 \markdownRendererDlDefinitionEndPrototype}%
1772 \ExplSyntaxOn
1773 \seq_gput_right:Nn
1774 \g_@@_renderers_seq
1775 { dlDefinitionEnd }
1776 \prop_gput:Nnn
1777 \g_@@_renderer_arities_prop
1778 { dlDefinitionEnd }
1779 { 0 }

```

```
1780 \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.

```
1781 \def\markdownRendererDlEnd{%
1782   \markdownRendererDlEndPrototype}%
1783 \ExplSyntaxOn
1784 \seq_gput_right:Nn
1785   \g_@@_renderers_seq
1786   { dlEnd }
1787 \prop_gput:Nnn
1788   \g_@@_renderer_arities_prop
1789   { dlEnd }
1790   { 0 }
1791 \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.

```
1792 \def\markdownRendererDlEndTight{%
1793   \markdownRendererDlEndTightPrototype}%
1794 \ExplSyntaxOn
1795 \seq_gput_right:Nn
1796   \g_@@_renderers_seq
1797   { dlEndTight }
1798 \prop_gput:Nnn
1799   \g_@@_renderer_arities_prop
1800   { dlEndTight }
1801   { 0 }
1802 \ExplSyntaxOff
```

### 2.2.5.11 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.

```
1803 \def\markdownRendererEllipsis{%
1804   \markdownRendererEllipsisPrototype}%
1805 \ExplSyntaxOn
1806 \seq_gput_right:Nn
1807   \g_@@_renderers_seq
1808   { ellipsis }
1809 \prop_gput:Nnn
1810   \g_@@_renderer_arities_prop
```

```

1811 { ellipsis }
1812 { 0 }
1813 \ExplSyntaxOff

```

### 2.2.5.12 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.

```

1814 \def\markdownRendererEmphasis{%
1815   \markdownRendererEmphasisPrototype}%
1816 \ExplSyntaxOn
1817 \seq_gput_right:Nn
1818   \g_@@_renderers_seq
1819   { emphasis }
1820 \prop_gput:Nnn
1821   \g_@@_renderer_arities_prop
1822   { emphasis }
1823   { 1 }
1824 \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.

```

1825 \def\markdownRendererStrongEmphasis{%
1826   \markdownRendererStrongEmphasisPrototype}%
1827 \ExplSyntaxOn
1828 \seq_gput_right:Nn
1829   \g_@@_renderers_seq
1830   { strongEmphasis }
1831 \prop_gput:Nnn
1832   \g_@@_renderer_arities_prop
1833   { strongEmphasis }
1834   { 1 }
1835 \ExplSyntaxOff

```

### 2.2.5.13 Fenced Code Attribute Context Renderers

The following macros are only produced, when the `fencedCode` and `fencedCodeAttributes` options are enabled.

The `\markdownRendererFencedCodeAttributeContextBegin` and `\markdownRendererFencedCodeAttributeContextEnd` macros represent the beginning and the end of a context in which the attributes of a fenced code apply. The macros receive no arguments.

```

1836 \def\markdownRendererFencedCodeAttributeContextBegin{%
1837   \markdownRendererFencedCodeAttributeContextBeginPrototype}%
1838 \ExplSyntaxOn

```

```

1839 \seq_gput_right:Nn
1840   \g_@@_renderers_seq
1841   { fencedCodeAttributeContextBegin }
1842 \prop_gput:Nnn
1843   \g_@@_renderer_arities_prop
1844   { fencedCodeAttributeContextBegin }
1845   { 0 }
1846 \ExplSyntaxOff
1847 \def\markdownRendererFencedCodeAttributeContextEnd{%
1848   \markdownRendererFencedCodeAttributeContextEndPrototype}%
1849 \ExplSyntaxOn
1850 \seq_gput_right:Nn
1851   \g_@@_renderers_seq
1852   { fencedCodeAttributeContextEnd }
1853 \prop_gput:Nnn
1854   \g_@@_renderer_arities_prop
1855   { fencedCodeAttributeContextEnd }
1856   { 0 }
1857 \ExplSyntaxOff

```

#### 2.2.5.14 Fenced Div Attribute Context Renderers

The following macros are only produced, when the `fencedDiv` option is enabled.

The `\markdownRendererFencedDivAttributeContextBegin` and `\markdownRendererFencedDivAttributeContextEnd` macros represent the beginning and the end of a context in which the attributes of a div apply. The macros receive no arguments.

```

1858 \def\markdownRendererFencedDivAttributeContextBegin{%
1859   \markdownRendererFencedDivAttributeContextBeginPrototype}%
1860 \ExplSyntaxOn
1861 \seq_gput_right:Nn
1862   \g_@@_renderers_seq
1863   { fencedDivAttributeContextBegin }
1864 \prop_gput:Nnn
1865   \g_@@_renderer_arities_prop
1866   { fencedDivAttributeContextBegin }
1867   { 0 }
1868 \ExplSyntaxOff
1869 \def\markdownRendererFencedDivAttributeContextEnd{%
1870   \markdownRendererFencedDivAttributeContextEndPrototype}%
1871 \ExplSyntaxOn
1872 \seq_gput_right:Nn
1873   \g_@@_renderers_seq
1874   { fencedDivAttributeContextEnd }
1875 \prop_gput:Nnn
1876   \g_@@_renderer_arities_prop
1877   { fencedDivAttributeContextEnd }
1878   { 0 }

```



1879 \ExplSyntaxOff

### 2.2.5.15 Header Attribute Context Renderers

The following macros are only produced, when the `autoIdentifiers`, `gfmAutoIdentifiers`, or `headerAttributes` options are enabled.

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

```
1880 \def\markdownRendererHeaderAttributeContextBegin{%
1881   \markdownRendererHeaderAttributeContextBeginPrototype}%
1882 \ExplSyntaxOn
1883 \seq_gput_right:Nn
1884   \g_@@_renderers_seq
1885   { headerAttributeContextBegin }
1886 \prop_gput:Nnn
1887   \g_@@_renderer_arities_prop
1888   { headerAttributeContextBegin }
1889   { 0 }
1890 \ExplSyntaxOff
1891 \def\markdownRendererHeaderAttributeContextEnd{%
1892   \markdownRendererHeaderAttributeContextEndPrototype}%
1893 \ExplSyntaxOn
1894 \seq_gput_right:Nn
1895   \g_@@_renderers_seq
1896   { headerAttributeContextEnd }
1897 \prop_gput:Nnn
1898   \g_@@_renderer_arities_prop
1899   { headerAttributeContextEnd }
1900   { 0 }
1901 \ExplSyntaxOff
```

### 2.2.5.16 Heading Renderers

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

```
1902 \def\markdownRendererHeadingOne{%
1903   \markdownRendererHeadingOnePrototype}%
1904 \ExplSyntaxOn
1905 \seq_gput_right:Nn
1906   \g_@@_renderers_seq
1907   { headingOne }
1908 \prop_gput:Nnn
1909   \g_@@_renderer_arities_prop
1910   { headingOne }
1911   { 1 }
1912 \ExplSyntaxOff
```

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

```
1913 \def\markdownRendererHeadingTwo{%
1914   \markdownRendererHeadingTwoPrototype}%
1915 \ExplSyntaxOn
1916 \seq_gput_right:Nn
1917   \g_@@_renderers_seq
1918   { headingTwo }
1919 \prop_gput:Nnn
1920   \g_@@_renderer_arities_prop
1921   { headingTwo }
1922   { 1 }
1923 \ExplSyntaxOff
```

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

```
1924 \def\markdownRendererHeadingThree{%
1925   \markdownRendererHeadingThreePrototype}%
1926 \ExplSyntaxOn
1927 \seq_gput_right:Nn
1928   \g_@@_renderers_seq
1929   { headingThree }
1930 \prop_gput:Nnn
1931   \g_@@_renderer_arities_prop
1932   { headingThree }
1933   { 1 }
1934 \ExplSyntaxOff
```

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

```
1935 \def\markdownRendererHeadingFour{%
1936   \markdownRendererHeadingFourPrototype}%
1937 \ExplSyntaxOn
1938 \seq_gput_right:Nn
1939   \g_@@_renderers_seq
1940   { headingFour }
1941 \prop_gput:Nnn
1942   \g_@@_renderer_arities_prop
1943   { headingFour }
1944   { 1 }
1945 \ExplSyntaxOff
```

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

```
1946 \def\markdownRendererHeadingFive{%
1947   \markdownRendererHeadingFivePrototype}%
```

```

1948 \ExplSyntaxOn
1949 \seq_gput_right:Nn
1950   \g_@@_renderers_seq
1951   { headingFive }
1952 \prop_gput:Nnn
1953   \g_@@_renderer_arities_prop
1954   { headingFive }
1955   { 1 }
1956 \ExplSyntaxOff

```

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

```

1957 \def\markdownRendererHeadingSix{%
1958   \markdownRendererHeadingSixPrototype}%
1959 \ExplSyntaxOn
1960 \seq_gput_right:Nn
1961   \g_@@_renderers_seq
1962   { headingSix }
1963 \prop_gput:Nnn
1964   \g_@@_renderer_arities_prop
1965   { headingSix }
1966   { 1 }
1967 \ExplSyntaxOff

```

#### 2.2.5.17 Inline HTML Comment Renderer

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.

```

1968 \def\markdownRendererInlineHtmlComment{%
1969   \markdownRendererInlineHtmlCommentPrototype}%
1970 \ExplSyntaxOn
1971 \seq_gput_right:Nn
1972   \g_@@_renderers_seq
1973   { inlineHtmlComment }
1974 \prop_gput:Nnn
1975   \g_@@_renderer_arities_prop
1976   { inlineHtmlComment }
1977   { 1 }
1978 \ExplSyntaxOff

```

#### 2.2.5.18 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.

```

1979 \def\markdownRendererInlineHtmlTag{%
1980   \markdownRendererInlineHtmlTagPrototype}%
1981 \ExplSyntaxOn
1982 \seq_gput_right:Nn
1983   \g_@@_renderers_seq
1984   { inlineHtmlTag }
1985 \prop_gput:Nnn
1986   \g_@@_renderer_arities_prop
1987   { inlineHtmlTag }
1988   { 1 }
1989 \ExplSyntaxOff
1990 \def\markdownRendererInputBlockHtmlElement{%
1991   \markdownRendererInputBlockHtmlElementPrototype}%
1992 \ExplSyntaxOn
1993 \seq_gput_right:Nn
1994   \g_@@_renderers_seq
1995   { inputBlockHtmlElement }
1996 \prop_gput:Nnn
1997   \g_@@_renderer_arities_prop
1998   { inputBlockHtmlElement }
1999   { 1 }
2000 \ExplSyntaxOff

```

### 2.2.5.19 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.

```

2001 \def\markdownRendererImage{%
2002   \markdownRendererImagePrototype}%
2003 \ExplSyntaxOn
2004 \seq_gput_right:Nn
2005   \g_@@_renderers_seq
2006   { image }
2007 \prop_gput:Nnn
2008   \g_@@_renderer_arities_prop
2009   { image }
2010   { 4 }
2011 \ExplSyntaxOff

```

### 2.2.5.20 Image Attribute Context Renderers

The following macros are only produced, when the `linkAttributes` option is enabled.

The `\markdownRendererImageAttributeContextBegin` and `\markdownRendererImageAttributeContextEnd` macros represent the beginning and the end of a context in which the attributes of an image apply. The macros receive no arguments.

```
2012 \def\markdownRendererImageAttributeContextBegin{%
2013   \markdownRendererImageAttributeContextBeginPrototype}%
2014 \ExplSyntaxOn
2015 \seq_gput_right:Nn
2016   \g_@@_renderers_seq
2017   { imageAttributeContextBegin }
2018 \prop_gput:Nnn
2019   \g_@@_renderer_arities_prop
2020   { imageAttributeContextBegin }
2021   { 0 }
2022 \ExplSyntaxOff
2023 \def\markdownRendererImageAttributeContextEnd{%
2024   \markdownRendererImageAttributeContextEndPrototype}%
2025 \ExplSyntaxOn
2026 \seq_gput_right:Nn
2027   \g_@@_renderers_seq
2028   { imageAttributeContextEnd }
2029 \prop_gput:Nnn
2030   \g_@@_renderer_arities_prop
2031   { imageAttributeContextEnd }
2032   { 0 }
2033 \ExplSyntaxOff
```

### 2.2.5.21 Interblock Separator Renderers

The `\markdownRendererInterblockSeparator` macro represents an interblock separator between two markdown block elements. The macro receives no arguments.

```
2034 \def\markdownRendererInterblockSeparator{%
2035   \markdownRendererInterblockSeparatorPrototype}%
2036 \ExplSyntaxOn
2037 \seq_gput_right:Nn
2038   \g_@@_renderers_seq
2039   { interblockSeparator }
2040 \prop_gput:Nnn
2041   \g_@@_renderer_arities_prop
2042   { interblockSeparator }
2043   { 0 }
2044 \ExplSyntaxOff
```

Users can use more than one blank line to delimit two block to indicate the end of a series of blocks that make up a logical paragraph. This produces a paragraph separator instead of an interblock separator. Between some blocks, such as markdown paragraphs, a paragraph separator is always produced.

The `\markdownRendererParagraphSeparator` macro represents a paragraph separator. The macro receives no arguments.

```

2045 \def\markdownRendererParagraphSeparator{%
2046   \markdownRendererParagraphSeparatorPrototype}%
2047 \ExplSyntaxOn
2048 \seq_gput_right:Nn
2049   \g_@@_renderers_seq
2050   { paragraphSeparator }
2051 \prop_gput:Nnn
2052   \g_@@_renderer_arities_prop
2053   { paragraphSeparator }
2054   { 0 }
2055 \ExplSyntaxOff

```

### 2.2.5.22 Line Block Renderers

The following macros are only produced, when the `lineBlocks` option is enabled.

The `\markdownRendererLineBlockBegin` and `\markdownRendererLineBlockEnd` macros represent the beginning and the end of a line block. The macros receive no arguments.

```

2056 \def\markdownRendererLineBlockBegin{%
2057   \markdownRendererLineBlockBeginPrototype}%
2058 \ExplSyntaxOn
2059 \seq_gput_right:Nn
2060   \g_@@_renderers_seq
2061   { lineBlockBegin }
2062 \prop_gput:Nnn
2063   \g_@@_renderer_arities_prop
2064   { lineBlockBegin }
2065   { 0 }
2066 \ExplSyntaxOff
2067 \def\markdownRendererLineBlockEnd{%
2068   \markdownRendererLineBlockEndPrototype}%
2069 \ExplSyntaxOn
2070 \seq_gput_right:Nn
2071   \g_@@_renderers_seq
2072   { lineBlockEnd }
2073 \prop_gput:Nnn
2074   \g_@@_renderer_arities_prop
2075   { lineBlockEnd }
2076   { 0 }
2077 \ExplSyntaxOff

```

### 2.2.5.23 Line Break Renderers

The `\markdownRendererSoftLineBreak` macro represents a soft line break. The macro receives no arguments.

```
2078 \def\markdownRendererSoftLineBreak{%
2079   \markdownRendererSoftLineBreakPrototype}%
2080 \ExplSyntaxOn
2081 \seq_gput_right:Nn
2082   \g_@@_renderers_seq
2083   { softLineBreak }
2084 \prop_gput:Nnn
2085   \g_@@_renderer_arities_prop
2086   { softLineBreak }
2087   { 0 }
2088 \ExplSyntaxOff
```

The `\markdownRendererHardLineBreak` macro represents a hard line break. The macro receives no arguments.

```
2089 \def\markdownRendererHardLineBreak{%
2090   \markdownRendererHardLineBreakPrototype}%
2091 \ExplSyntaxOn
2092 \seq_gput_right:Nn
2093   \g_@@_renderers_seq
2094   { hardLineBreak }
2095 \prop_gput:Nnn
2096   \g_@@_renderer_arities_prop
2097   { hardLineBreak }
2098   { 0 }
2099 \ExplSyntaxOff
```

### 2.2.5.24 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.

```
2100 \def\markdownRendererLink{%
2101   \markdownRendererLinkPrototype}%
2102 \ExplSyntaxOn
2103 \seq_gput_right:Nn
2104   \g_@@_renderers_seq
2105   { link }
2106 \prop_gput:Nnn
2107   \g_@@_renderer_arities_prop
2108   { link }
2109   { 4 }
2110 \ExplSyntaxOff
```

### 2.2.5.25 Link Attribute Context Renderers

The following macros are only produced, when the `linkAttributes` option is enabled.

The `\markdownRendererLinkAttributeContextBegin` and `\markdownRendererLinkAttributeContextEnd` macros represent the beginning and the end of a context in which the attributes of a hyperlink apply. The macros receive no arguments.

```
2111 \def\markdownRendererLinkAttributeContextBegin{%
2112   \markdownRendererLinkAttributeContextBeginPrototype}%
2113 \ExplSyntaxOn
2114 \seq_gput_right:Nn
2115   \g_@@_renderers_seq
2116   { linkAttributeContextBegin }
2117 \prop_gput:Nnn
2118   \g_@@_renderer_arities_prop
2119   { linkAttributeContextBegin }
2120   { 0 }
2121 \ExplSyntaxOff
2122 \def\markdownRendererLinkAttributeContextEnd{%
2123   \markdownRendererLinkAttributeContextEndPrototype}%
2124 \ExplSyntaxOn
2125 \seq_gput_right:Nn
2126   \g_@@_renderers_seq
2127   { linkAttributeContextEnd }
2128 \prop_gput:Nnn
2129   \g_@@_renderer_arities_prop
2130   { linkAttributeContextEnd }
2131   { 0 }
2132 \ExplSyntaxOff
```

### 2.2.5.26 Marked Text Renderer

The following macro is only produced, when the `mark` option is enabled.

The `\markdownRendererMark` macro represents a span of marked or highlighted text. The macro receives a single argument that corresponds to the marked text.

```
2133 \def\markdownRendererMark{%
2134   \markdownRendererMarkPrototype}%
2135 \ExplSyntaxOn
2136 \seq_gput_right:Nn
2137   \g_@@_renderers_seq
2138   { mark }
2139 \prop_gput:Nnn
2140   \g_@@_renderer_arities_prop
2141   { mark }
2142   { 1 }
2143 \ExplSyntaxOff
```



### 2.2.5.27 Markdown Document Renderers

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

A  $\TeX$  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.

```
2144 \def\markdownRendererDocumentBegin{%
2145   \markdownRendererDocumentBeginPrototype}%
2146 \ExplSyntaxOn
2147 \seq_gput_right:Nn
2148   \g_@@_renderers_seq
2149   { documentBegin }
2150 \prop_gput:Nnn
2151   \g_@@_renderer_arities_prop
2152   { documentBegin }
2153   { 0 }
2154 \ExplSyntaxOff
2155 \def\markdownRendererDocumentEnd{%
2156   \markdownRendererDocumentEndPrototype}%
2157 \ExplSyntaxOn
2158 \seq_gput_right:Nn
2159   \g_@@_renderers_seq
2160   { documentEnd }
2161 \prop_gput:Nnn
2162   \g_@@_renderer_arities_prop
2163   { documentEnd }
2164   { 0 }
2165 \ExplSyntaxOff
```

### 2.2.5.28 Non-Breaking Space Renderer

The `\markdownRendererNbsp` macro represents a non-breaking space.

```
2166 \def\markdownRendererNbsp{%
2167   \markdownRendererNbspPrototype}%
2168 \ExplSyntaxOn
2169 \seq_gput_right:Nn
2170   \g_@@_renderers_seq
2171   { nbsp }
2172 \prop_gput:Nnn
2173   \g_@@_renderer_arities_prop
2174   { nbsp }
2175   { 0 }
2176 \ExplSyntaxOff
```

### 2.2.5.29 Note Renderer

The `\markdownRendererNote` macro represents a note. This macro will only be produced, when the `notes` option is enabled. The macro receives a single argument that corresponds to the note text.

```
2177 \def\markdownRendererNote{%
2178   \markdownRendererNotePrototype}%
2179 \ExplSyntaxOn
2180 \seq_gput_right:Nn
2181   \g_@@_renderers_seq
2182   { note }
2183 \prop_gput:Nnn
2184   \g_@@_renderer_arities_prop
2185   { note }
2186   { 1 }
2187 \ExplSyntaxOff
```

### 2.2.5.30 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.

```
2188 \def\markdownRendererOlBegin{%
2189   \markdownRendererOlBeginPrototype}%
2190 \ExplSyntaxOn
2191 \seq_gput_right:Nn
2192   \g_@@_renderers_seq
2193   { olBegin }
2194 \prop_gput:Nnn
2195   \g_@@_renderer_arities_prop
2196   { olBegin }
2197   { 0 }
2198 \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.

```
2199 \def\markdownRendererOlBeginTight{%
2200   \markdownRendererOlBeginTightPrototype}%
2201 \ExplSyntaxOn
2202 \seq_gput_right:Nn
2203   \g_@@_renderers_seq
2204   { olBeginTight }
2205 \prop_gput:Nnn
```

```

2206 \g_@@_renderer_arities_prop
2207 { olBeginTight }
2208 { 0 }
2209 \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`).

```

2210 \def\markdownRendererFancyOlBegin{%
2211 \markdownRendererFancyOlBeginPrototype}%
2212 \ExplSyntaxOn
2213 \seq_gput_right:Nn
2214 \g_@@_renderers_seq
2215 { fancyOlBegin }
2216 \prop_gput:Nnn
2217 \g_@@_renderer_arities_prop
2218 { fancyOlBegin }
2219 { 2 }
2220 \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.

```

2221 \def\markdownRendererFancyOlBeginTight{%
2222 \markdownRendererFancyOlBeginTightPrototype}%
2223 \ExplSyntaxOn
2224 \seq_gput_right:Nn
2225 \g_@@_renderers_seq
2226 { fancyOlBeginTight }
2227 \prop_gput:Nnn
2228 \g_@@_renderer_arities_prop
2229 { fancyOlBeginTight }
2230 { 2 }
2231 \ExplSyntaxOff

```

The `\markdownRendererOlItem` 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.

```

2232 \def\markdownRendererOlItem{%

```

```

2233 \markdownRendererOlItemPrototype}%
2234 \ExplSyntaxOn
2235 \seq_gput_right:Nn
2236 \g_@@_renderers_seq
2237 { olItem }
2238 \prop_gput:Nnn
2239 \g_@@_renderer_arities_prop
2240 { olItem }
2241 { 0 }
2242 \ExplSyntaxOff

```

The `\markdownRendererOlItemEnd` 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.

```

2243 \def\markdownRendererOlItemEnd{%
2244 \markdownRendererOlItemEndPrototype}%
2245 \ExplSyntaxOn
2246 \seq_gput_right:Nn
2247 \g_@@_renderers_seq
2248 { olItemEnd }
2249 \prop_gput:Nnn
2250 \g_@@_renderer_arities_prop
2251 { olItemEnd }
2252 { 0 }
2253 \ExplSyntaxOff

```

The `\markdownRendererOlItemWithNumber` 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.

```

2254 \def\markdownRendererOlItemWithNumber{%
2255 \markdownRendererOlItemWithNumberPrototype}%
2256 \ExplSyntaxOn
2257 \seq_gput_right:Nn
2258 \g_@@_renderers_seq
2259 { olItemWithNumber }
2260 \prop_gput:Nnn
2261 \g_@@_renderer_arities_prop
2262 { olItemWithNumber }
2263 { 1 }
2264 \ExplSyntaxOff

```

The `\markdownRendererFancyOlItem` 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.

```

2265 \def\markdownRendererFancyOlItem{%

```

```

2266 \markdownRendererFancyO1ItemPrototype}%
2267 \ExplSyntaxOn
2268 \seq_gput_right:Nn
2269 \g_@@_renderers_seq
2270 { fancyO1Item }
2271 \prop_gput:Nnn
2272 \g_@@_renderer_arities_prop
2273 { fancyO1Item }
2274 { 0 }
2275 \ExplSyntaxOff

```

The `\markdownRendererFancyO1ItemEnd` 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.

```

2276 \def\markdownRendererFancyO1ItemEnd{%
2277 \markdownRendererFancyO1ItemEndPrototype}%
2278 \ExplSyntaxOn
2279 \seq_gput_right:Nn
2280 \g_@@_renderers_seq
2281 { fancyO1ItemEnd }
2282 \prop_gput:Nnn
2283 \g_@@_renderer_arities_prop
2284 { fancyO1ItemEnd }
2285 { 0 }
2286 \ExplSyntaxOff

```

The `\markdownRendererFancyO1ItemWithNumber` 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.

```

2287 \def\markdownRendererFancyO1ItemWithNumber{%
2288 \markdownRendererFancyO1ItemWithNumberPrototype}%
2289 \ExplSyntaxOn
2290 \seq_gput_right:Nn
2291 \g_@@_renderers_seq
2292 { fancyO1ItemWithNumber }
2293 \prop_gput:Nnn
2294 \g_@@_renderer_arities_prop
2295 { fancyO1ItemWithNumber }
2296 { 1 }
2297 \ExplSyntaxOff

```

The `\markdownRendererO1End` 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.

```

2298 \def\markdownRendererO1End{%
2299   \markdownRendererO1EndPrototype}%
2300 \ExplSyntaxOn
2301 \seq_gput_right:Nn
2302   \g_@@_renderers_seq
2303   { olEnd }
2304 \prop_gput:Nnn
2305   \g_@@_renderer_arities_prop
2306   { olEnd }
2307   { 0 }
2308 \ExplSyntaxOff

```

The `\markdownRendererO1EndTight` 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.

```

2309 \def\markdownRendererO1EndTight{%
2310   \markdownRendererO1EndTightPrototype}%
2311 \ExplSyntaxOn
2312 \seq_gput_right:Nn
2313   \g_@@_renderers_seq
2314   { olEndTight }
2315 \prop_gput:Nnn
2316   \g_@@_renderer_arities_prop
2317   { olEndTight }
2318   { 0 }
2319 \ExplSyntaxOff

```

The `\markdownRendererFancyO1End` 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.

```

2320 \def\markdownRendererFancyO1End{%
2321   \markdownRendererFancyO1EndPrototype}%
2322 \ExplSyntaxOn
2323 \seq_gput_right:Nn
2324   \g_@@_renderers_seq
2325   { fancyO1End }
2326 \prop_gput:Nnn
2327   \g_@@_renderer_arities_prop
2328   { fancyO1End }
2329   { 0 }
2330 \ExplSyntaxOff

```

The `\markdownRendererFancyO1EndTight` 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.

```

2331 \def\markdownRendererFancy01EndTight{%
2332   \markdownRendererFancy01EndTightPrototype}%
2333 \ExplSyntaxOn
2334 \seq_gput_right:Nn
2335   \g_@@_renderers_seq
2336   { fancy01EndTight }
2337 \prop_gput:Nnn
2338   \g_@@_renderer_arities_prop
2339   { fancy01EndTight }
2340   { 0 }
2341 \ExplSyntaxOff

```

### 2.2.5.31 Raw Content Renderers

The `\markdownRendererInputRawInline` macro represents an inline raw span. The macro receives two arguments: the filename of a file containing the inline raw span contents and the raw attribute that designates the format of the inline raw span. This macro will only be produced, when the `rawAttribute` option is enabled.

```

2342 \def\markdownRendererInputRawInline{%
2343   \markdownRendererInputRawInlinePrototype}%
2344 \ExplSyntaxOn
2345 \seq_gput_right:Nn
2346   \g_@@_renderers_seq
2347   { inputRawInline }
2348 \prop_gput:Nnn
2349   \g_@@_renderer_arities_prop
2350   { inputRawInline }
2351   { 2 }
2352 \ExplSyntaxOff

```

The `\markdownRendererInputRawBlock` macro represents a raw block. The macro receives two arguments: the filename of a file containing the raw block and the raw attribute that designates the format of the raw block. This macro will only be produced, when the `rawAttribute` and `fencedCode` options are enabled.

```

2353 \def\markdownRendererInputRawBlock{%
2354   \markdownRendererInputRawBlockPrototype}%
2355 \ExplSyntaxOn
2356 \seq_gput_right:Nn
2357   \g_@@_renderers_seq
2358   { inputRawBlock }
2359 \prop_gput:Nnn
2360   \g_@@_renderer_arities_prop
2361   { inputRawBlock }
2362   { 2 }

```

```
2363 \ExplSyntaxOff
```

### 2.2.5.32 Section Renderers

The `\markdownRendererSectionBegin` and `\markdownRendererSectionEnd` macros represent the beginning and the end of a section based on headings.

```
2364 \def\markdownRendererSectionBegin{%
2365   \markdownRendererSectionBeginPrototype}%
2366 \ExplSyntaxOn
2367 \seq_gput_right:Nn
2368   \g_@@_renderers_seq
2369   { sectionBegin }
2370 \prop_gput:Nnn
2371   \g_@@_renderer_arities_prop
2372   { sectionBegin }
2373   { 0 }
2374 \ExplSyntaxOff
2375 \def\markdownRendererSectionEnd{%
2376   \markdownRendererSectionEndPrototype}%
2377 \ExplSyntaxOn
2378 \seq_gput_right:Nn
2379   \g_@@_renderers_seq
2380   { sectionEnd }
2381 \prop_gput:Nnn
2382   \g_@@_renderer_arities_prop
2383   { sectionEnd }
2384   { 0 }
2385 \ExplSyntaxOff
```

### 2.2.5.33 Replacement Character Renderers

The `\markdownRendererReplacementCharacter` macro represents the U+0000 and U+FFFD Unicode characters. The macro receives no arguments.

```
2386 \def\markdownRendererReplacementCharacter{%
2387   \markdownRendererReplacementCharacterPrototype}%
2388 \ExplSyntaxOn
2389 \seq_gput_right:Nn
2390   \g_@@_renderers_seq
2391   { replacementCharacter }
2392 \prop_gput:Nnn
2393   \g_@@_renderer_arities_prop
2394   { replacementCharacter }
2395   { 0 }
2396 \ExplSyntaxOff
```

### 2.2.5.34 Special Character Renderers



The following macros replace any special plain T<sub>E</sub>X characters, including the active pipe character (|) of ConT<sub>E</sub>Xt, in the input text. These macros will only be produced, when the `hybrid` option is `false`.

```

2397 \def\markdownRendererLeftBrace{%
2398   \markdownRendererLeftBracePrototype}%
2399 \ExplSyntaxOn
2400 \seq_gput_right:Nn
2401   \g_@@_renderers_seq
2402   { leftBrace }
2403 \prop_gput:Nnn
2404   \g_@@_renderer_arities_prop
2405   { leftBrace }
2406   { 0 }
2407 \ExplSyntaxOff
2408 \def\markdownRendererRightBrace{%
2409   \markdownRendererRightBracePrototype}%
2410 \ExplSyntaxOn
2411 \seq_gput_right:Nn
2412   \g_@@_renderers_seq
2413   { rightBrace }
2414 \prop_gput:Nnn
2415   \g_@@_renderer_arities_prop
2416   { rightBrace }
2417   { 0 }
2418 \ExplSyntaxOff
2419 \def\markdownRendererDollarSign{%
2420   \markdownRendererDollarSignPrototype}%
2421 \ExplSyntaxOn
2422 \seq_gput_right:Nn
2423   \g_@@_renderers_seq
2424   { dollarSign }
2425 \prop_gput:Nnn
2426   \g_@@_renderer_arities_prop
2427   { dollarSign }
2428   { 0 }
2429 \ExplSyntaxOff
2430 \def\markdownRendererPercentSign{%
2431   \markdownRendererPercentSignPrototype}%
2432 \ExplSyntaxOn
2433 \seq_gput_right:Nn
2434   \g_@@_renderers_seq
2435   { percentSign }
2436 \prop_gput:Nnn
2437   \g_@@_renderer_arities_prop
2438   { percentSign }
2439   { 0 }

```

```

2440 \ExplSyntaxOff
2441 \def\markdownRendererAmpersand{%
2442   \markdownRendererAmpersandPrototype}%
2443 \ExplSyntaxOn
2444 \seq_gput_right:Nn
2445   \g_@@_renderers_seq
2446   { ampersand }
2447 \prop_gput:Nnn
2448   \g_@@_renderer_arities_prop
2449   { ampersand }
2450   { 0 }
2451 \ExplSyntaxOff
2452 \def\markdownRendererUnderscore{%
2453   \markdownRendererUnderscorePrototype}%
2454 \ExplSyntaxOn
2455 \seq_gput_right:Nn
2456   \g_@@_renderers_seq
2457   { underscore }
2458 \prop_gput:Nnn
2459   \g_@@_renderer_arities_prop
2460   { underscore }
2461   { 0 }
2462 \ExplSyntaxOff
2463 \def\markdownRendererHash{%
2464   \markdownRendererHashPrototype}%
2465 \ExplSyntaxOn
2466 \seq_gput_right:Nn
2467   \g_@@_renderers_seq
2468   { hash }
2469 \prop_gput:Nnn
2470   \g_@@_renderer_arities_prop
2471   { hash }
2472   { 0 }
2473 \ExplSyntaxOff
2474 \def\markdownRendererCircumflex{%
2475   \markdownRendererCircumflexPrototype}%
2476 \ExplSyntaxOn
2477 \seq_gput_right:Nn
2478   \g_@@_renderers_seq
2479   { circumflex }
2480 \prop_gput:Nnn
2481   \g_@@_renderer_arities_prop
2482   { circumflex }
2483   { 0 }
2484 \ExplSyntaxOff
2485 \def\markdownRendererBackslash{%
2486   \markdownRendererBackslashPrototype}%

```

```

2487 \ExplSyntaxOn
2488 \seq_gput_right:Nn
2489   \g_@@_renderers_seq
2490   { backslash }
2491 \prop_gput:Nnn
2492   \g_@@_renderer_arities_prop
2493   { backslash }
2494   { 0 }
2495 \ExplSyntaxOff
2496 \def\markdownRendererTilde{%
2497   \markdownRendererTildePrototype}%
2498 \ExplSyntaxOn
2499 \seq_gput_right:Nn
2500   \g_@@_renderers_seq
2501   { tilde }
2502 \prop_gput:Nnn
2503   \g_@@_renderer_arities_prop
2504   { tilde }
2505   { 0 }
2506 \ExplSyntaxOff
2507 \def\markdownRendererPipe{%
2508   \markdownRendererPipePrototype}%
2509 \ExplSyntaxOn
2510 \seq_gput_right:Nn
2511   \g_@@_renderers_seq
2512   { pipe }
2513 \prop_gput:Nnn
2514   \g_@@_renderer_arities_prop
2515   { pipe }
2516   { 0 }
2517 \ExplSyntaxOff

```

### 2.2.5.35 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.

```

2518 \def\markdownRendererStrikeThrough{%
2519   \markdownRendererStrikeThroughPrototype}%
2520 \ExplSyntaxOn
2521 \seq_gput_right:Nn
2522   \g_@@_renderers_seq
2523   { strikeThrough }
2524 \prop_gput:Nnn
2525   \g_@@_renderer_arities_prop
2526   { strikeThrough }

```

```
2527 { 1 }
2528 \ExplSyntaxOff
```

### 2.2.5.36 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.

```
2529 \def\markdownRendererSubscript{%
2530   \markdownRendererSubscriptPrototype}%
2531 \ExplSyntaxOn
2532 \seq_gput_right:Nn
2533   \g_@@_renderers_seq
2534   { subscript }
2535 \prop_gput:Nnn
2536   \g_@@_renderer_arities_prop
2537   { subscript }
2538   { 1 }
```

### 2.2.5.37 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.

```
2539 \def\markdownRendererSuperscript{%
2540   \markdownRendererSuperscriptPrototype}%
2541 \ExplSyntaxOn
2542 \seq_gput_right:Nn
2543   \g_@@_renderers_seq
2544   { superscript }
2545 \prop_gput:Nnn
2546   \g_@@_renderer_arities_prop
2547   { superscript }
2548   { 1 }
2549 \ExplSyntaxOff
```

### 2.2.5.38 Table Attribute Context Renderers

The following macros are only produced, when the `tableCaptions` and `tableAttributes` options are enabled.

The `\markdownRendererTableAttributeContextBegin` and `\markdownRendererTableAttributeContextEnd` macros represent the beginning and the end of a context in which the attributes of a table apply. The macros receive no arguments.

```
2550 \def\markdownRendererTableAttributeContextBegin{%
2551   \markdownRendererTableAttributeContextBeginPrototype}%
2552 \ExplSyntaxOn
```

```

2553 \seq_gput_right:Nn
2554   \g_@@_renderers_seq
2555   { tableAttributeContextBegin }
2556 \prop_gput:Nnn
2557   \g_@@_renderer_arities_prop
2558   { tableAttributeContextBegin }
2559   { 0 }
2560 \ExplSyntaxOff
2561 \def\markdownRendererTableAttributeContextEnd{%
2562   \markdownRendererTableAttributeContextEndPrototype}%
2563 \ExplSyntaxOn
2564 \seq_gput_right:Nn
2565   \g_@@_renderers_seq
2566   { tableAttributeContextEnd }
2567 \prop_gput:Nnn
2568   \g_@@_renderer_arities_prop
2569   { tableAttributeContextEnd }
2570   { 0 }
2571 \ExplSyntaxOff

```

### 2.2.5.39 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.

```

2572 \def\markdownRendererTable{%
2573   \markdownRendererTablePrototype}%
2574 \ExplSyntaxOn
2575 \seq_gput_right:Nn
2576   \g_@@_renderers_seq
2577   { table }
2578 \prop_gput:Nnn
2579   \g_@@_renderer_arities_prop
2580   { table }
2581   { 3 }
2582 \ExplSyntaxOff

```

#### 2.2.5.40 T<sub>E</sub>X Math Renderers

The `\markdownRendererInlineMath` and `\markdownRendererDisplayMath` macros represent inline and display T<sub>E</sub>X math. Both macros receive a single argument that corresponds to the T<sub>E</sub>X math content. These macros will only be produced, when the `texMathDollars`, `texMathSingleBackslash`, or `texMathDoubleBackslash` option are enabled.

```
2583 \def\markdownRendererInlineMath{%
2584   \markdownRendererInlineMathPrototype}%
2585 \ExplSyntaxOn
2586 \seq_gput_right:Nn
2587   \g_@@_renderers_seq
2588   { inlineMath }
2589 \prop_gput:Nnn
2590   \g_@@_renderer_arities_prop
2591   { inlineMath }
2592   { 1 }
2593 \ExplSyntaxOff
2594 \def\markdownRendererDisplayMath{%
2595   \markdownRendererDisplayMathPrototype}%
2596 \ExplSyntaxOn
2597 \seq_gput_right:Nn
2598   \g_@@_renderers_seq
2599   { displayMath }
2600 \prop_gput:Nnn
2601   \g_@@_renderer_arities_prop
2602   { displayMath }
2603   { 1 }
2604 \ExplSyntaxOff
```

#### 2.2.5.41 Thematic Break Renderer

The `\markdownRendererThematicBreak` macro represents a thematic break. The macro receives no arguments.

```
2605 \def\markdownRendererThematicBreak{%
2606   \markdownRendererThematicBreakPrototype}%
2607 \ExplSyntaxOn
2608 \seq_gput_right:Nn
2609   \g_@@_renderers_seq
2610   { thematicBreak }
2611 \prop_gput:Nnn
2612   \g_@@_renderer_arities_prop
2613   { thematicBreak }
2614   { 0 }
2615 \ExplSyntaxOff
```

#### 2.2.5.42 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.

```

2616 \def\markdownRendererTickedBox{%
2617   \markdownRendererTickedBoxPrototype}%
2618 \ExplSyntaxOn
2619 \seq_gput_right:Nn
2620   \g_@@_renderers_seq
2621   { tickedBox }
2622 \prop_gput:Nnn
2623   \g_@@_renderer_arities_prop
2624   { tickedBox }
2625   { 0 }
2626 \ExplSyntaxOff
2627 \def\markdownRendererHalfTickedBox{%
2628   \markdownRendererHalfTickedBoxPrototype}%
2629 \ExplSyntaxOn
2630 \seq_gput_right:Nn
2631   \g_@@_renderers_seq
2632   { halfTickedBox }
2633 \prop_gput:Nnn
2634   \g_@@_renderer_arities_prop
2635   { halfTickedBox }
2636   { 0 }
2637 \ExplSyntaxOff
2638 \def\markdownRendererUntickedBox{%
2639   \markdownRendererUntickedBoxPrototype}%
2640 \ExplSyntaxOn
2641 \seq_gput_right:Nn
2642   \g_@@_renderers_seq
2643   { untickedBox }
2644 \prop_gput:Nnn
2645   \g_@@_renderer_arities_prop
2646   { untickedBox }
2647   { 0 }
2648 \ExplSyntaxOff

```

#### 2.2.5.43 Warning and Error Renderers

The `\markdownRendererWarning` and `\markdownRendererError` macros represent warnings and errors produced by the markdown parser. Both macros receive a single parameter with the text of the warning or error.

```

2649 \def\markdownRendererWarning{%
2650   \markdownRendererWarningPrototype}%

```

```

2651 \def\markdownRendererError{%
2652   \markdownRendererErrorPrototype}%
2653 \ExplSyntaxOn
2654 \seq_gput_right:Nn
2655   \g_@@_renderers_seq
2656   { warning }
2657 \prop_gput:Nnn
2658   \g_@@_renderer_arities_prop
2659   { warning }
2660   { 1 }
2661 \seq_gput_right:Nn
2662   \g_@@_renderers_seq
2663   { error }
2664 \prop_gput:Nnn
2665   \g_@@_renderer_arities_prop
2666   { error }
2667   { 1 }
2668 \ExplSyntaxOff

```

#### 2.2.5.44 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.

```

2669 \def\markdownRendererJekyllDataBegin{%
2670   \markdownRendererJekyllDataBeginPrototype}%
2671 \ExplSyntaxOn
2672 \seq_gput_right:Nn
2673   \g_@@_renderers_seq
2674   { jekyllDataBegin }
2675 \prop_gput:Nnn
2676   \g_@@_renderer_arities_prop
2677   { jekyllDataBegin }
2678   { 0 }
2679 \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.

```

2680 \def\markdownRendererJekyllDataEnd{%
2681   \markdownRendererJekyllDataEndPrototype}%
2682 \ExplSyntaxOn
2683 \seq_gput_right:Nn
2684   \g_@@_renderers_seq
2685   { jekyllDataEnd }
2686 \prop_gput:Nnn
2687   \g_@@_renderer_arities_prop

```



```

2688 { jekyllDataEnd }
2689 { 0 }
2690 \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.

```

2691 \def\markdownRendererJekyllDataMappingBegin{%
2692   \markdownRendererJekyllDataMappingBeginPrototype}%
2693 \ExplSyntaxOn
2694 \seq_gput_right:Nn
2695   \g_@@_renderers_seq
2696   { jekyllDataMappingBegin }
2697 \prop_gput:Nnn
2698   \g_@@_renderer_arities_prop
2699   { jekyllDataMappingBegin }
2700   { 2 }
2701 \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.

```

2702 \def\markdownRendererJekyllDataMappingEnd{%
2703   \markdownRendererJekyllDataMappingEndPrototype}%
2704 \ExplSyntaxOn
2705 \seq_gput_right:Nn
2706   \g_@@_renderers_seq
2707   { jekyllDataMappingEnd }
2708 \prop_gput:Nnn
2709   \g_@@_renderer_arities_prop
2710   { jekyllDataMappingEnd }
2711   { 0 }
2712 \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.

```

2713 \def\markdownRendererJekyllDataSequenceBegin{%
2714   \markdownRendererJekyllDataSequenceBeginPrototype}%
2715 \ExplSyntaxOn
2716 \seq_gput_right:Nn
2717   \g_@@_renderers_seq

```

```

2718 { jekyllDataSequenceBegin }
2719 \prop_gput:Nnn
2720 \g_@@_renderer_arities_prop
2721 { jekyllDataSequenceBegin }
2722 { 2 }
2723 \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.

```

2724 \def\markdownRendererJekyllDataSequenceEnd{%
2725   \markdownRendererJekyllDataSequenceEndPrototype}%
2726 \ExplSyntaxOn
2727 \seq_gput_right:Nn
2728   \g_@@_renderers_seq
2729   { jekyllDataSequenceEnd }
2730 \prop_gput:Nnn
2731   \g_@@_renderer_arities_prop
2732   { jekyllDataSequenceEnd }
2733   { 0 }
2734 \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.

```

2735 \def\markdownRendererJekyllDataBoolean{%
2736   \markdownRendererJekyllDataBooleanPrototype}%
2737 \ExplSyntaxOn
2738 \seq_gput_right:Nn
2739   \g_@@_renderers_seq
2740   { jekyllDataBoolean }
2741 \prop_gput:Nnn
2742   \g_@@_renderer_arities_prop
2743   { jekyllDataBoolean }
2744   { 2 }
2745 \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.

```

2746 \def\markdownRendererJekyllDataNumber{%
2747   \markdownRendererJekyllDataNumberPrototype}%

```

```

2748 \ExplSyntaxOn
2749 \seq_gput_right:Nn
2750   \g_@@_renderers_seq
2751   { jekyllDataNumber }
2752 \prop_gput:Nnn
2753   \g_@@_renderer_arities_prop
2754   { jekyllDataNumber }
2755   { 2 }
2756 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataTypographicString` and `\markdownRendererJekyllDataProgrammaticString` macros represent string scalar values 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.

For each string scalar value, both macros are produced. Whereas `\markdownRendererJekyllDataProgrammaticString` receives the scalar value after all markdown markup and special  $\TeX$  characters in the string have been replaced by  $\TeX$  macros, `\markdownRendererJekyllDataTypographicString` receives the raw scalar value. Therefore, whereas the `\markdownRendererJekyllDataTypographicString` macro is more appropriate for texts that are supposed to be typeset with  $\TeX$ , such as document titles, author names, or exam questions, the `\markdownRendererJekyllDataProgrammaticString` macro is more appropriate for identifiers and other programmatic text that won't be typeset by  $\TeX$ .

```

2757 \def\markdownRendererJekyllDataTypographicString{%
2758   \markdownRendererJekyllDataTypographicStringPrototype}%
2759 \def\markdownRendererJekyllDataProgrammaticString{%
2760   \markdownRendererJekyllDataProgrammaticStringPrototype}%
2761 \ExplSyntaxOn
2762 \seq_gput_right:Nn
2763   \g_@@_renderers_seq
2764   { jekyllDataTypographicString }
2765 \prop_gput:Nnn
2766   \g_@@_renderer_arities_prop
2767   { jekyllDataTypographicString }
2768   { 2 }
2769 \seq_gput_right:Nn
2770   \g_@@_renderers_seq
2771   { jekyllDataProgrammaticString }
2772 \prop_gput:Nnn
2773   \g_@@_renderer_arities_prop
2774   { jekyllDataProgrammaticString }
2775   { 2 }
2776 \ExplSyntaxOff

```

Before Markdown 3.7.0, the `\markdownRendererJekyllDataTypographicString` macro was named `\markdownRendererJekyllDataString` and the `\markdownRendererJekyllDataProgrammaticString` macro was named `\markdownRendererJekyllDataProgrammaticString`.

macro was not produced. The `\markdownRendererJekyllDataString` has been deprecated and will be removed in Markdown 4.0.0.

```

2777 \ExplSyntaxOn
2778 \cs_gset:Npn
2779   \markdownRendererJekyllDataTypographicString
2780   {
2781     \cs_if_exist:NTF
2782       \markdownRendererJekyllDataString
2783       {
2784         \markdownWarning
2785         {
2786           The~jekyllDataString~renderer~has~been~deprecated,~
2787           to~be~removed~in~Markdown~4.0.0
2788         }
2789         \markdownRendererJekyllDataString
2790       }
2791     {
2792       \cs_if_exist:NTF
2793         \markdownRendererJekyllDataStringPrototype
2794         {
2795           \markdownWarning
2796           {
2797             The~jekyllDataString~renderer~prototype~
2798             has~been~deprecated,~
2799             to~be~removed~in~Markdown~4.0.0
2800           }
2801           \markdownRendererJekyllDataStringPrototype
2802         }
2803       {
2804         \markdownRendererJekyllDataTypographicStringPrototype
2805       }
2806     }
2807   }
2808 \seq_gput_right:Nn
2809   \g_@@_renderers_seq
2810   { jekyllDataString }
2811 \prop_gput:Nnn
2812   \g_@@_renderer_arities_prop
2813   { jekyllDataString }
2814   { 2 }
2815 \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 2.2.6.1 for the description of the high-level expl3 interface that you can also use to react to YAML metadata.

```

2816 \def\markdownRendererJekyllDataEmpty{%
2817   \markdownRendererJekyllDataEmptyPrototype}%
2818 \ExplSyntaxOn
2819 \seq_gput_right:Nn
2820   \g_@@_renderers_seq
2821   { jekyllDataEmpty }
2822 \prop_gput:Nnn
2823   \g_@@_renderer_arities_prop
2824   { jekyllDataEmpty }
2825   { 1 }
2826 \ExplSyntaxOff

```

### 2.2.5.45 Generating Plain T<sub>E</sub>X Token Renderer Macros and Key-Values

We define the command `\@@_define_renderers:` that defines plain T<sub>E</sub>X macros for token renderers. Furthermore, the `\markdownSetup` macro also accepts the `renderers` key, whose value must be a list of key-values, where the keys correspond to the markdown token renderer macros and the values are new definitions of these token renderers.

```

2827 \ExplSyntaxOn
2828 \cs_new:Nn \@@_define_renderers:
2829   {
2830     \seq_map_inline:Nn
2831       \g_@@_renderers_seq
2832       {

```

The `\markdownRendererJekyllDataString` has been deprecated and will be removed in Markdown 4.0.0.

```

2833     \str_if_eq:nnF
2834       { ##1 }
2835       { jekyllDataString }
2836     {
2837       \@@_define_renderer:n
2838         { ##1 }
2839     }
2840   }
2841 }
2842 \cs_new:Nn \@@_define_renderer:n
2843   {
2844     \@@_renderer_tl_to_csname:nN
2845       { #1 }
2846     \l_tmpa_tl
2847     \prop_get:NnN
2848       \g_@@_renderer_arities_prop
2849       { #1 }

```

```

2850     \l_tmpb_tl
2851     \@@_define_renderer:ncV
2852     { #1 }
2853     { \l_tmpa_tl }
2854     \l_tmpb_tl
2855   }
2856 \cs_new:Nn \@@_renderer_tl_to_csname:nN
2857 {
2858   \tl_set:Nn
2859     \l_tmpa_tl
2860     { \str_uppercase:n { #1 } }
2861   \tl_set:Nx
2862     #2
2863     {
2864       markdownRenderer
2865       \tl_head:f { \l_tmpa_tl }
2866       \tl_tail:n { #1 }
2867     }
2868   }
2869 \tl_new:N
2870   \l_@@_renderer_definition_tl
2871 \bool_new:N
2872   \g_@@_appending_renderer_bool
2873 \cs_new:Nn \@@_define_renderer:nNn
2874 {
2875   \keys_define:nn
2876     { markdown/options/renderers }
2877     {
2878       #1 .code:n = {
2879         \tl_set:Nn
2880           \l_@@_renderer_definition_tl
2881           { ##1 }
2882         \regex_replace_all:nnN
2883           { \cP\#0 }
2884           { #1 }
2885         \l_@@_renderer_definition_tl
2886         \bool_if:NT
2887           \g_@@_appending_renderer_bool
2888           {
2889             \@@_tl_set_from_cs:NNn
2890               \l_tmpa_tl
2891               #2
2892               { #3 }
2893             \tl_put_left:NV
2894               \l_@@_renderer_definition_tl
2895               \l_tmpa_tl
2896           }

```

```

2897         \cs_generate_from_arg_count:NNnV
2898         #2
2899         \cs_set:Npn
2900         { #3 }
2901         \l_@@_renderer_definition_tl
2902     },
2903 }
2904 }

```

We define the function `\@@_tl_set_from_cs:NNn` [9]. The function takes a token list, a control sequence with undelimited parameters, and the number of parameters the control sequence accepts, and locally assigns the replacement text of the control sequence to the token list.

```

2905 \cs_new_protected:Nn
2906 \@@_tl_set_from_cs:NNn
2907 {
2908     \tl_set:Nn
2909     \l_tmpa_tl
2910     { #2 }
2911     \int_step_inline:nn
2912     { #3 }
2913     {
2914         \exp_args:Nnc
2915         \tl_put_right:Nn
2916         \l_tmpa_tl
2917         { @@_tl_set_from_cs_parameter_ ##1 }
2918     }
2919     \exp_args:NNV
2920     \tl_set:No
2921     \l_tmpb_tl
2922     \l_tmpa_tl
2923     \regex_replace_all:nnN
2924     { \cP. }
2925     { \0\0 }
2926     \l_tmpb_tl
2927     \int_step_inline:nn
2928     { #3 }
2929     {
2930         \regex_replace_all:nnN
2931         { \c { @@_tl_set_from_cs_parameter_ ##1 } }
2932         { \cP\# ##1 }
2933         \l_tmpb_tl
2934     }
2935     \tl_set:NV
2936     #1
2937     \l_tmpb_tl
2938 }

```

```

2939 \cs_generate_variant:Nn
2940   \@@_define_renderer:nNn
2941   { ncV }
2942 \cs_generate_variant:Nn
2943   \cs_generate_from_arg_count:NNnn
2944   { NNnV }
2945 \cs_generate_variant:Nn
2946   \tl_put_left:Nn
2947   { Nv }
2948 \keys_define:nn
2949   { markdown/options }
2950   {
2951     renderers .code:n = {
2952       \keys_set:nn
2953         { markdown/options/renderers }
2954         { #1 }
2955     },
2956   }

```

The following example code showcases a possible configuration of the `\markdownRendererLink` and `\markdownRendererEmphasis` token renderer macros.

```

\markdownSetup{
  renderers = {
    link = {#4}, % Render links as the link title.
    emphasis = {{\it #1}}, % Render emphasized text using italics.
  }
}

```

```

2957 \tl_new:N
2958   \l_@@_renderer_glob_definition_tl
2959 \seq_new:N
2960   \l_@@_renderer_glob_results_seq
2961 \regex_const:Nn
2962   \c_@@_appending_key_regex
2963   { \s*+$ }
2964 \keys_define:nn
2965   { markdown/options/renderers }
2966   {
2967     unknown .code:n = {

```

Besides defining renderers at once, we can also define them incrementally using the appending operator (`+=`). This can be especially useful in defining rules for processing different HTML class names and identifiers:

```

\markdownSetup{
  renderers = {

```



```

% Start with empty renderers.
headerAttributeContextBegin = {},
attributeClassName = {},
attributeIdentifier = {},
% Define the processing of a single specific HTML class name.
headerAttributeContextBegin += {
    \markdownSetup{
        renderers = {
            attributeClassName += {...},
        },
    }
},
% Define the processing of a single specific HTML identifier.
headerAttributeContextBegin += {
    \markdownSetup{
        renderers = {
            attributeIdentifier += {...},
        },
    }
},
}
}

```

```

2968     \regex_match:NVTF
2969     \c_@@_appending_key_regex
2970     \l_keys_key_str
2971     {
2972         \bool_gset_true:N
2973         \g_@@_appending_renderer_bool
2974         \tl_set:NV
2975         \l_tmpa_tl
2976         \l_keys_key_str
2977         \regex_replace_once:NnN
2978         \c_@@_appending_key_regex
2979         { }
2980         \l_tmpa_tl
2981         \tl_set:Nx
2982         \l_tmpb_tl
2983         { { \l_tmpa_tl } = }
2984         \tl_put_right:Nn
2985         \l_tmpb_tl
2986         { { #1 } }
2987         \keys_set:nV
2988         { markdown/options/renderers }

```

```

2989         \l_tmpb_tl
2990         \bool_gset_false:N
2991         \g_@@_appending_renderer_bool
2992     }

```

In addition to exact token renderer names, we also support wildcards (\*) and enumerations (1) that match multiple token renderer names:

```

\markdownSetup{
  renderers = {
    heading* = {{\bf #1}},      % Render headings using the bold face.
    jekyllData(String|Number) = {% % Render YAML string and numbers
      {\it #2}%                % using italics.
    },
  }
}

```

Wildcards and enumerations can be combined:

```

\markdownSetup{
  renderers = {
    *1Item(|End) = {"},      % Quote ordered/bullet list items.
  }
}

```

To determine the current token renderer, you can use the pseudo-parameter #0:

```

\markdownSetup{
  renderers = {
    heading* = {#0: #1},      % Render headings as the renderer name
                                % followed by the heading text.
  }
}

```

```

2993     {
2994         \@@_glob_seq:VnN
2995         \l_keys_key_str
2996         { g_@@_renderers_seq }
2997         \l_@@_renderer_glob_results_seq
2998         \seq_if_empty:NTF
2999         \l_@@_renderer_glob_results_seq
3000         {
3001             \msg_error:nnV
3002             { markdown }

```

```

3003         { undefined-renderer }
3004         \l_keys_key_str
3005     }
3006     {
3007         \tl_set:Nn
3008         \l_@@_renderer_glob_definition_tl
3009         { \exp_not:n { #1 } }
3010         \seq_map_inline:Nn
3011         \l_@@_renderer_glob_results_seq
3012         {
3013             \tl_set:Nn
3014             \l_tmpa_tl
3015             { { ##1 } = }
3016             \tl_put_right:Nx
3017             \l_tmpa_tl
3018             { { \l_@@_renderer_glob_definition_tl } }
3019             \keys_set:nV
3020             { markdown/options/renderers }
3021             \l_tmpa_tl
3022         }
3023     }
3024 }
3025 },
3026 }
3027 \msg_new:nnn
3028 { markdown }
3029 { undefined-renderer }
3030 {
3031     Renderer~#1~is~undefined.
3032 }
3033 \cs_generate_variant:Nn
3034 \@@_glob_seq:nnN
3035 { VnN }
3036 \cs_generate_variant:Nn
3037 \cs_generate_from_arg_count:NNnn
3038 { cNVV }
3039 \cs_generate_variant:Nn
3040 \msg_error:nnn
3041 { nnV }
3042 \prg_generate_conditional_variant:Nnn
3043 \regex_match:Nn
3044 { NV }
3045 { TF }
3046 \prop_new:N
3047 \g_@@_glob_cache_prop
3048 \tl_new:N
3049 \l_@@_current_glob_tl

```

```

3050 \cs_new:Nn
3051   \@@_glob_seq:nnN
3052   {
3053     \tl_set:Nn
3054       \l_@@_current_glob_tl
3055       { ^ #1 $ }
3056     \prop_get:NeNTF
3057       \g_@@_glob_cache_prop
3058       { #2 / \l_@@_current_glob_tl }
3059     \l_tmpa_clist
3060     {
3061       \seq_set_from_clist:NN
3062         #3
3063         \l_tmpa_clist
3064     }
3065     {
3066       \seq_clear:N
3067         #3
3068       \regex_replace_all:nnN
3069         { \* }
3070         { .* }
3071         \l_@@_current_glob_tl
3072       \regex_set:NV
3073         \l_tmpa_regex
3074         \l_@@_current_glob_tl
3075       \seq_map_inline:cn
3076         { #2 }
3077         {
3078           \regex_match:NnT
3079             \l_tmpa_regex
3080             { ##1 }
3081             {
3082               \seq_put_right:Nn
3083                 #3
3084                 { ##1 }
3085             }
3086         }
3087       \clist_set_from_seq:NN
3088         \l_tmpa_clist
3089         #3
3090       \prop_gput:NeV
3091         \g_@@_glob_cache_prop
3092         { #2 / \l_@@_current_glob_tl }
3093         \l_tmpa_clist
3094     }
3095   }
3096 % TODO: Remove in TeX Live 2023.

```

```

3097 \prg_generate_conditional_variant:Nnn
3098   \prop_get:NnN
3099   { NeN }
3100   { TF }
3101 \cs_generate_variant:Nn
3102   \regex_set:Nn
3103   { NV }
3104 \cs_generate_variant:Nn
3105   \prop_gput:Nnn
3106   { NeV }

```

If plain  $\TeX$  is the top layer, we use the `\@@_define_renderers:` macro to define plain  $\TeX$  token renderer macros and key-values immediately. Otherwise, we postpone the definition until the upper layers have been loaded.

```

3107 \str_if_eq:VVT
3108   \c_@@_top_layer_tl
3109   \c_@@_option_layer_plain_tex_tl
3110   {
3111     \@@_define_renderers:
3112   }
3113 \ExplSyntaxOff

```

## 2.2.6 Token Renderer Prototypes

### 2.2.6.1 YAML Metadata Renderer Prototypes

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

```

3114 \ExplSyntaxOn
3115 \keys_define:nn
3116   { markdown/jekyllData }
3117   { }
3118 \ExplSyntaxOff

```

The `jekyllDataRenderers` key can be used as a syntactic sugar for setting the `markdown/jekyllData` key-values without using the `expl3` language.

```

3119 \ExplSyntaxOn
3120 \@@_with_various_cases:nn
3121   { jekyllDataRenderers }
3122   {
3123     \keys_define:nn
3124       { markdown/options }
3125       {
3126         #1 .code:n = {
3127           \tl_set:Nn
3128             \l_tmpa_tl
3129             { ##1 }

```

To ensure that keys containing forward slashes get passed correctly, we replace all forward slashes in the input with backslash tokens with category code letter and then undo the replacement. This means that if any unbraced backslash tokens with category code letter exist in the input, they will be replaced with forward slashes. However, this should be extremely rare.

```

3130         \tl_replace_all:NnV
3131         \l_tmpa_tl
3132         { / }
3133         \c_backslash_str
3134         \keys_set:nV
3135         { markdown/options/jekyll-data-renderers }
3136         \l_tmpa_tl
3137     },
3138 }
3139 }
3140 \keys_define:nn
3141 { markdown/options/jekyll-data-renderers }
3142 {
3143     unknown .code:n = {
3144         \tl_set_eq:NN
3145         \l_tmpa_tl
3146         \l_keys_key_str
3147         \tl_replace_all:NVn
3148         \l_tmpa_tl
3149         \c_backslash_str
3150         { / }
3151         \tl_put_right:Nn
3152         \l_tmpa_tl
3153         {
3154             .code:n = { #1 }
3155         }
3156         \keys_define:nV
3157         { markdown/jekyllData }
3158         \l_tmpa_tl
3159     }
3160 }
3161 \cs_generate_variant:Nn
3162 \keys_define:nn
3163 { nV }
3164 \ExplSyntaxOff

```

### 2.2.6.2 Generating Plain TeX Token Renderer Prototype Macros and Key-Values

We define the command `\@@_define_renderer_prototypes:` that defines plain TeX macros for token renderer prototypes. Furthermore, the `\markdownSetup` macro

also accepts the `rendererPrototype` key, whose value must be a list of key-values, where the keys correspond to the markdown token renderer prototype macros and the values are new definitions of these token renderer prototypes.

```

3165 \ExplSyntaxOn
3166 \cs_new:Nn \@@_define_renderer_prototypes:
3167   {
3168     \seq_map_inline:Nn
3169       \g_@@_renderers_seq
3170       {

```

The `\markdownRendererJekyllDataString` has been deprecated and will be removed in Markdown 4.0.0.

```

3171     \str_if_eq:nnF
3172       { ##1 }
3173       { jekyllDataString }
3174     {
3175       \@@_define_renderer_prototype:n
3176         { ##1 }
3177     }
3178   }
3179 }
3180 \cs_new:Nn \@@_define_renderer_prototype:n
3181   {
3182     \@@_renderer_prototype_tl_to_csname:nN
3183       { #1 }
3184     \l_tmpa_tl
3185     \prop_get:NnN
3186       \g_@@_renderer_arities_prop
3187       { #1 }
3188     \l_tmpb_tl
3189     \@@_define_renderer_prototype:ncV
3190       { #1 }
3191       { \l_tmpa_tl }
3192       \l_tmpb_tl
3193   }
3194 \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
3195   {
3196     \tl_set:Nn
3197       \l_tmpa_tl
3198       { \str_uppercase:n { #1 } }
3199     \tl_set:Nx
3200       #2
3201       {
3202         markdownRenderer
3203         \tl_head:f { \l_tmpa_tl }
3204         \tl_tail:n { #1 }
3205         Prototype

```

```

3206     }
3207 }
3208 \tl_new:N
3209   \l_@@_renderer_prototype_definition_tl
3210 \bool_new:N
3211   \g_@@_appending_renderer_prototype_bool
3212 \cs_new:Nn \@@_define_renderer_prototype:nNn
3213 {
3214   \keys_define:nn
3215     { markdown/options/renderer-prototypes }
3216     {
3217       #1 .code:n = {
3218         \tl_set:Nn
3219           \l_@@_renderer_prototype_definition_tl
3220           { ##1 }
3221         \regex_replace_all:nnN
3222           { \cP\#0 }
3223           { #1 }
3224         \l_@@_renderer_prototype_definition_tl
3225         \bool_if:NT
3226           \g_@@_appending_renderer_prototype_bool
3227           {
3228             \@@_tl_set_from_cs:NNn
3229             \l_tmpa_tl
3230             #2
3231             { #3 }
3232             \tl_put_left:NV
3233               \l_@@_renderer_prototype_definition_tl
3234               \l_tmpa_tl
3235           }
3236         \cs_generate_from_arg_count:NNnV
3237           #2
3238           \cs_set:Npn
3239             { #3 }
3240             \l_@@_renderer_prototype_definition_tl
3241       },
3242     }

```

Unless the token renderer prototype macro has already been defined, we provide an empty definition.

```

3243   \cs_if_free:NT
3244     #2
3245     {
3246       \cs_generate_from_arg_count:NNnn
3247         #2
3248         \cs_set:Npn
3249           { #3 }

```



```

3250         { }
3251     }
3252 }
3253 \cs_generate_variant:Nn
3254   \@_define_renderer_prototype:nNn
3255   { ncV }

```

The following example code showcases a possible configuration of the `\markdownRendererImagePrototype` and `\markdownRendererCodeSpanPrototype` token renderer prototype macros.

```

\markdownSetup{
  rendererPrototypes = {
    image = {\pdfximage{#2}},      % Embed PDF images in the document.
    codeSpan = {\tt #1},          % Render inline code using monospace.
  }
}

```

```

3256 \keys_define:nn
3257   { markdown/options/renderer-prototypes }
3258   {
3259     unknown .code:n = {

```

Besides defining renderer prototypes at once, we can also define them incrementally using the appending operator (`+=`). This can be especially useful in defining rules for processing different HTML class names and identifiers:

```

\markdownSetup{
  rendererPrototypes = {
    % Start with empty renderer prototypes.
    headerAttributeContextBegin = {},
    attributeClassName = {},
    attributeIdentifier = {},
    % Define the processing of a single specific HTML class name.
    headerAttributeContextBegin += {
      \markdownSetup{
        rendererPrototypes = {
          attributeClassName += {...},
        },
      },
    },
    % Define the processing of a single specific HTML identifier.
    headerAttributeContextBegin += {
      \markdownSetup{

```

```

    rendererPrototypes = {
      attributeIdentifier += {...},
    },
  },
},
}

```

```

3260   \regex_match:NVTF
3261   \c_@@_appending_key_regex
3262   \l_keys_key_str
3263   {
3264     \bool_gset_true:N
3265     \g_@@_appending_renderer_prototype_bool
3266     \tl_set:NV
3267     \l_tmpa_tl
3268     \l_keys_key_str
3269     \regex_replace_once:NnN
3270     \c_@@_appending_key_regex
3271     { }
3272     \l_tmpa_tl
3273     \tl_set:Nx
3274     \l_tmpb_tl
3275     { { \l_tmpa_tl } = }
3276     \tl_put_right:Nn
3277     \l_tmpb_tl
3278     { { #1 } }
3279     \keys_set:nV
3280     { markdown/options/renderer-prototypes }
3281     \l_tmpb_tl
3282     \bool_gset_false:N
3283     \g_@@_appending_renderer_prototype_bool
3284   }

```

In addition to exact token renderer prototype names, we also support wildcards (\*) and enumerations (|) that match multiple token renderer prototype names:

```

\markdownSetup{
  rendererPrototypes = {
    heading* = {{\bf #1}},      % Render headings using the bold face.
    jekyllData(String|Number) = { % Render YAML string and numbers
      {\it #2}%                % using italics.
    },
  }
}

```

Wildcards and enumerations can be combined:

```
\markdownSetup{
  rendererPrototypes = {
    *lItem(|End) = {""},           % Quote ordered/bullet list items.
  }
}
```

To determine the current token renderer prototype, you can use the pseudo-parameter #0:

```
\markdownSetup{
  rendererPrototypes = {
    heading* = {#0: #1}, % Render headings as the renderer prototype
  }                    % name followed by the heading text.
}
```

```
3285     {
3286         \@@_glob_seq:VnN
3287         \l_keys_key_str
3288         { g_@@_renderers_seq }
3289         \l_@@_renderer_glob_results_seq
3290     \seq_if_empty:NTF
3291     \l_@@_renderer_glob_results_seq
3292     {
3293         \msg_error:nnV
3294         { markdown }
3295         { undefined-renderer-prototype }
3296         \l_keys_key_str
3297     }
3298     {
3299         \tl_set:Nn
3300         \l_@@_renderer_glob_definition_tl
3301         { \exp_not:n { #1 } }
3302     \seq_map_inline:Nn
3303     \l_@@_renderer_glob_results_seq
3304     {
3305         \tl_set:Nn
3306         \l_tmpa_tl
3307         { { ##1 } = }
3308         \tl_put_right:Nx
3309         \l_tmpa_tl
3310         { { \l_@@_renderer_glob_definition_tl } }
3311     \keys_set:nV
3312     { markdown/options/renderer-prototypes }
```

```

3313             \l_tmpa_tl
3314         }
3315     }
3316 }
3317 },
3318 }
3319 \msg_new:nnn
3320 { markdown }
3321 { undefined-renderer-prototype }
3322 {
3323     Renderer~prototype~#1-is~undefined.
3324 }
3325 \@@_with_various_cases:nn
3326 { rendererPrototypes }
3327 {
3328     \keys_define:nn
3329     { markdown/options }
3330     {
3331         #1 .code:n = {
3332             \keys_set:nn
3333             { markdown/options/renderer-prototypes }
3334             { ##1 }
3335         },
3336     }
3337 }

```

If plain  $\text{T}_{\text{E}}\text{X}$  is the top layer, we use the `\@@_define_renderer_prototypes:` macro to define plain  $\text{T}_{\text{E}}\text{X}$  token renderer prototype macros and key-values immediately. Otherwise, we postpone the definition until the upper layers have been loaded.

```

3338 \str_if_eq:VVT
3339 \c_@@_top_layer_tl
3340 \c_@@_option_layer_plain_tex_tl
3341 {
3342     \@@_define_renderer_prototypes:
3343 }
3344 \ExplSyntaxOff

```

## 2.2.7 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.

The `\markdownInfo`, `\markdownWarning`, and `\markdownError` macros have been deprecated and will be removed in the next major version of the Markdown package.

## 2.2.8 Miscellanea

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

```
3345 \let\markdownMakeOther\relax
```

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

```
3346 \let\markdownReadAndConvert\relax
3347 \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*.

```
3348 \catcode`\|=0\catcode`\=12%
3349 |gdef|markdownBegin{%
3350   |markdownReadAndConvert{\markdownEnd}%
3351                               {\|markdownEnd}}%
3352 |endgroup
```

The macro is exposed in the interface, so that users can create their own markdown environments. Due to the way the arguments are passed to Lua, the first argument may not contain the string `]]` (regardless of the category code of the bracket symbol).

The `code` key, which can be used to immediately expand and execute code.

```
3353 \ExplSyntaxOn
3354 \keys_define:nn
3355   { markdown/options }
3356   {
3357     code .code:n = { #1 },
3358   }
3359 \ExplSyntaxOff
```

This can be especially useful in snippets.

## 2.3 $\LaTeX$ Interface

The  $\LaTeX$  interface provides  $\LaTeX$  environments for the typesetting of markdown input from within  $\LaTeX$ , facilities for setting Lua, plain  $\TeX$ , and  $\LaTeX$  options used during the conversion from markdown to plain  $\TeX$ , and facilities for changing

the way markdown tokens are rendered. The rest of the interface is inherited from the plain  $\text{\TeX}$  interface (see Section 2.2).

To determine whether  $\text{\LaTeX}$  is the top layer or if there are other layers above  $\text{\LaTeX}$ , we take a look on whether the `\c_@@_top_layer_tl` token list has already been defined. If not, we will assume that  $\text{\LaTeX}$  is the top layer.

```

3360 \ExplSyntaxOn
3361 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
3362 \cs_generate_variant:Nn
3363   \tl_const:Nn
3364   { NV }
3365 \tl_if_exist:NF
3366   \c_@@_top_layer_tl
3367   {
3368     \tl_const:NV
3369       \c_@@_top_layer_tl
3370       \c_@@_option_layer_latex_tl
3371   }
3372 \ExplSyntaxOff
3373 \input markdown/markdown

```

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

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

where *<options>* are the  $\text{\LaTeX}$  interface options (see Section 2.3.2). Note that *<options>* inside the `\usepackage` macro may not set the `markdownRenderers` (see Section 2.2.5.45) and `markdownRendererPrototypes` (see Section 2.2.6.2) keys. Furthermore, although the base variant of the `import` key that loads a single  $\text{\LaTeX}$  theme (see Section 2.3.3) can be used, the extended variant that can load multiple themes and import snippets from them (see Section 2.2.4) cannot. This limitation is due to the way  $\text{\LaTeX} 2_{\epsilon}$  parses package options.

## 2.3.1 Typesetting Markdown

The interface exposes the `markdown` and `markdown*`  $\text{\LaTeX}$  environments, and redefines the `\markinline` and `\markdownInput` commands.

### 2.3.1.1 The `markdown` and `markdown*` $\text{\LaTeX}$ environments

The `markdown` and `markdown*`  $\text{\LaTeX}$  environments are used to typeset markdown document fragments. Both  $\text{\LaTeX}$  environments accept  $\text{\LaTeX}$  interface options (see Section 2.3.2) as the only argument. This argument is optional for the `markdown` environment and mandatory for the `markdown*` environment.

The `markdown*` environment has been deprecated and will be removed in the next major version of the Markdown package.

```
3374 \newenvironment{markdown}\relax\relax
3375 \newenvironment{markdown*}[1]\relax\relax
```

You may prepend your own code to the `\markdown` macro and append your own code to the `\markdownEnd` 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}[smartEllipses] _Hello_ <b>world</b> ... \end{markdown} % ... \end{document}</pre>	<pre>\documentclass{article} \usepackage{markdown} \begin{document} % ... \begin{markdown*}{smartEllipses} _Hello_ <b>world</b> ... \end{markdown*} % ... \end{document}</pre>
--	--

You can't directly extend the `markdown` L<sup>A</sup>T<sub>E</sub>X environment by using it in other environments as follows:

```
\newenvironment{foo}%
    {code before \begin{markdown}[some, options]}%
    {\end{markdown} code after}
```

This is because the implementation looks for the literal string `\end{markdown}` to stop scanning the markdown text. However, you can work around this limitation by using the `\markdown` and `\markdownEnd` macros directly in the definition as follows:

```
\newenvironment{foo}%
    {code before \markdown[some, options]}%
    {\markdownEnd code after}
```

Specifically, the `\markdown` macro must appear at the end of the replacement text and must be followed by text that has not yet been ingested by T<sub>E</sub>X's input processor. Furthermore, using the `\markdownEnd` macro is optional and only makes a difference if you redefined it to produce special effects before and after the `markdown` L<sup>A</sup>T<sub>E</sub>X environment. Lastly, you can't nest the other environments. For example, the following definition is incorrect:

```
\newenvironment{bar}{\begin{foo}}{\end{foo}}
```

In this example, you should use the `\markdown` macro directly in the definition of the environment `bar`:

```
\newenvironment{bar}{\markdown[some, options]}{\markdownEnd}
```

### 2.3.1.2 The `\markinline` and `\markdownInput` macros

The `\markinline` macro accepts a single mandatory parameter containing inline markdown content and expands to the result of the conversion of the input markdown document to plain `TEX`. Unlike the `\markinline` macro provided by the plain `TEX` interface, this macro also accepts `LATEX` interface options (see Section 2.3.2) as its optional argument. These options will only influence this markdown content.

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 `TEX`. Unlike the `\markdownInput` macro provided by the plain `TEX` interface, this macro also accepts `LATEX` interface options (see Section 2.3.2) as its optional argument. These options will only influence this markdown document.

The following example `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 `LATEX` options are represented by a comma-delimited list of `<key>=<value>` pairs. For boolean options, the `=<value>` part is optional, and `<key>` will be interpreted as `<key>=true` if the `=<value>` part has been omitted.

`LATEX` options map directly to the options recognized by the plain `TEX` interface (see Section 2.2.2) and to the markdown token renderers and their prototypes recognized by the plain `TEX` interface (see Sections 2.2.5 and 2.2.6).

The `LATEX` options may be specified when loading the `LATEX` package, when using the `markdown*` `LATEX` environment or the `\markdownInput` macro (see Section 2.3), or via the `\markdownSetup` macro.



### 2.3.2.1 Finalizing and Freezing the Cache

To ensure compatibility with the `minted` package [10, Section 5.1], which supports the `finalizcache` and `frozenscache` package options with similar semantics to the `finalizeCache` and `frozenCache` plain  $\TeX$  options, the Markdown package also recognizes these as aliases and accepts 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  $\LaTeX$  document sources for distribution.

```
3376 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
3377 \DeclareOption{frozenscache}{\markdownSetup{frozenCache}}
```

### 2.3.2.2 Generating Plain $\TeX$ Option, Token Renderer, and Token Renderer Prototype Macros and Key-Values

If  $\LaTeX$  is the top layer, we use the `\@@_define_option_commands_and_keyvals:`, `\@@_define_renderers:`, and `\@@_define_renderer_prototypes:` macro to define plain  $\TeX$  option, token renderer, and token renderer prototype macros and key-values immediately. Otherwise, we postpone the definition until the upper layers have been loaded.

```
3378 \ExplSyntaxOn
3379 \str_if_eq:VVT
3380   \c_@@_top_layer_tl
3381   \c_@@_option_layer_latex_tl
3382   {
3383     \@@_define_option_commands_and_keyvals:
3384     \@@_define_renderers:
3385     \@@_define_renderer_prototypes:
3386   }
3387 \ExplSyntaxOff
```

The following example  $\LaTeX$  code showcases a possible configuration of plain  $\TeX$  interface options `hybrid`, `smartEllipses`, and `cacheDir`.

```
\markdownSetup{
  hybrid,
  smartEllipses,
```

```
    cacheDir = /tmp,  
  }
```

### 2.3.3 Themes

In Section 2.2.3, we described the concept of themes. In L<sup>A</sup>T<sub>E</sub>X, we expand on the concept of themes by allowing a theme to be a full-blown L<sup>A</sup>T<sub>E</sub>X package. Specifically, the key-values `theme=<theme name>` and `import=<theme name>` load a L<sup>A</sup>T<sub>E</sub>X package named `markdowntheme<munged theme name>.sty` if it exists and a T<sub>E</sub>X document named `markdowntheme<munged theme name>.tex` otherwise.

Having the Markdown package automatically load either the generic `.tex theme file` or the L<sup>A</sup>T<sub>E</sub>X-specific `.sty theme file` allows developers to have a single *theme file*, when the theme is small or the difference between T<sub>E</sub>X formats is unimportant, and scale up to separate theme files native to different T<sub>E</sub>X formats for large multi-format themes, where different code is needed for different T<sub>E</sub>X formats. To enable code reuse, developers can load the `.tex theme file` from the `.sty theme file` using the `\markdownLoadPlainTeXTheme` macro.

If the L<sup>A</sup>T<sub>E</sub>X option with keys `theme` or `import` 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 L<sup>A</sup>T<sub>E</sub>X 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_bamer_MU.sty` L<sup>A</sup>T<sub>E</sub>X package, and finally the `markdownthemewitiko_dot.sty` L<sup>A</sup>T<sub>E</sub>X package:

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

```
3388 \newif\ifmarkdownLaTeXLoaded  
3389 \markdownLaTeXLoadedfalse
```

Due to limitations of L<sup>A</sup>T<sub>E</sub>X, themes may not be loaded after the beginning of a L<sup>A</sup>T<sub>E</sub>X document.

Built-in L<sup>A</sup>T<sub>E</sub>X 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[import=witiko/dot]{markdown}
\setkeys{Gin}{
  width = \columnwidth,
  height = 0.65\paperheight,
  keepaspectratio}
\begin{document}
\begin{markdown}
``` dot Various formats of mathematical formulae
digraph tree {
  margin = 0;
  rankdir = "LR";

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

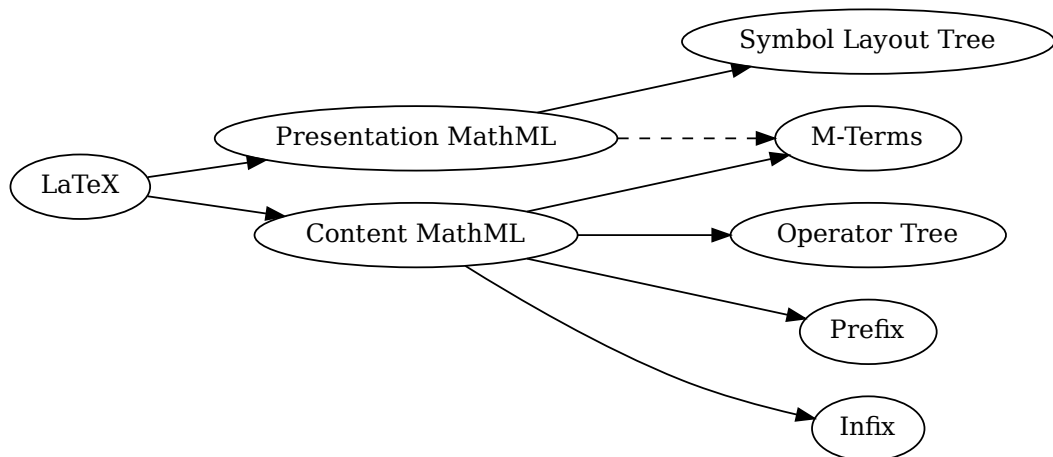
  latex [label = "LaTeX"];
  pmml [label = "Presentation MathML"];
  cmml [label = "Content MathML"];
  slt [label = "Symbol Layout Tree"];
  opt [label = "Operator Tree"];
  prefix [label = "Prefix"];
  infix [label = "Infix"];
  mterms [label = "M-Terms"];
}
```
\end{markdown}
\end{document}

```

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

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

```
3390 \ProvidesPackage{markdownthemewitiko_dot}[2021/03/09]%
```



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

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

```

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

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

```
3391 \ProvidesPackage{markdownthemewitiko_graphicx_http}[2021/03/22]%
```

**witiko/markdown/defaults** A  $\LaTeX$  theme with the default definitions of token renderer prototypes for plain  $\TeX$ . This theme is loaded automatically together with the package and explicitly loading it has no effect.

```
3392 \AtEndOfPackage{
3393   \markdownLaTeXLoadedtrue
```

```

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

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

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

```



# Chapter 1

## Introduction

1.1 Section  
1.1.1 Subsection  
Hello *Markdown!*

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

Table 1.1: Table

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

At the end of the  $\LaTeX$  module, we load the `witiko/markdown/defaults`  $\LaTeX$  theme (see Section 2.2.3) with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.3).

```

3394 \markdownIfOption{noDefaults}{-}{
3395   \markdownSetup{theme=witiko/markdown/defaults}
3396 }
3397 }

3398 \ProvidesPackage{markdownthemewitiko_markdown_defaults}[2024/01/03]%

```

Please, see Section 3.3.3 for implementation details of the built-in  $\LaTeX$  themes.

## 2.4 ConTeXt Interface

To determine whether ConTeXt is the top layer or if there are other layers above ConTeXt, we take a look on whether the `\c_@@_top_layer_tl` token list has already been defined. If not, we will assume that ConTeXt is the top layer.

```

3399 \ExplSyntaxOn
3400 \tl_const:Nn \c_@@_option_layer_context_tl { context }
3401 \cs_generate_variant:Nn
3402   \tl_const:Nn
3403   { NV }
3404 \tl_if_exist:NF

```

```

3405 \c_@@_top_layer_tl
3406 {
3407   \tl_const:NV
3408     \c_@@_top_layer_tl
3409     \c_@@_option_layer_context_tl
3410 }
3411 \ExplSyntaxOff

```

The ConTeXt interface provides a start-stop macro pair for the typesetting of markdown input from within ConTeXt and facilities for setting Lua, plain TeX, and ConTeXt options used during the conversion from markdown to plain TeX. The rest of the interface is inherited from the plain TeX interface (see Section 2.2).

```

3412 \writestatus{loading}{ConTeXt User Module / markdown}%
3413 \startmodule[markdown]
3414 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
3415   \do#\do\^\do\_do\%do\~}%
3416 \input markdown/markdown

```

The ConTeXt interface is implemented by the `t-markdown.tex` ConTeXt module file that can be loaded as follows:

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

It is expected that the special plain TeX 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` macro.

```

3417 \let\startmarkdown\relax
3418 \let\stopmarkdown\relax
3419 \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 TeX interface.

The following example ConTeXt 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  $\text{T}_{\text{E}}\text{X}$ . Unlike the `\markdownInput` macro provided by the plain  $\text{T}_{\text{E}}\text{X}$  interface, this macro also accepts Con $\text{T}_{\text{E}}\text{X}$ t interface options (see Section 2.4.2) as its optional argument. These options will only influence this markdown document.

The following example  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  code showcases the usage of the `\markdownInput` macro:

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

## 2.4.2 Options

The Con $\text{T}_{\text{E}}\text{X}$ t 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.

Con $\text{T}_{\text{E}}\text{X}$ t options map directly to the options recognized by the plain  $\text{T}_{\text{E}}\text{X}$  interface (see Section 2.2.2).

The Con $\text{T}_{\text{E}}\text{X}$ t options may be specified when using the `\inputmarkdown` macro (see Section 2.4), via the `\markdownSetup` macro, or via the `\setupmarkdown[#1]` macro, which is an alias for `\markdownSetup{#1}`.

```
3420 \ExplSyntaxOn  
3421 \cs_new:Npn  
3422   \setupmarkdown  
3423   [ #1 ]  
3424   {  
3425     \@@_setup:n  
3426     { #1 }  
3427   }  
3428 \ExplSyntaxOff
```

### 2.4.2.1 Generating Plain $\text{T}_{\text{E}}\text{X}$ Option Macros and Key-Values

Unlike plain  $\text{T}_{\text{E}}\text{X}$ , we also accept caseless variants of options in line with the style of Con $\text{T}_{\text{E}}\text{X}$ t.

```
3429 \ExplSyntaxOn
```

```

3430 \cs_new:Nn \@@_caseless:N
3431 {
3432   \regex_replace_all:nnN
3433     { ([a-z])([A-Z]) }
3434     { \1 \c { str_lowercase:n } \cB\{ \2 \cE\} }
3435     #1
3436   \tl_set:Nx
3437     #1
3438     { #1 }
3439 }
3440 \seq_gput_right:Nn \g_@@_cases_seq { @@_caseless:N }

```

If ConTEXt is the top layer, we use the `\@@_define_option_commands_and_keyvals:`, `\@@_define_renderers:`, and `\@@_define_renderer_prototypes:` macro to define plain TEX option, token renderer, and token renderer prototype macros and key-values immediately. Otherwise, we postpone the definition until the upper layers have been loaded.

```

3441 \str_if_eq:VVT
3442   \c_@@_top_layer_tl
3443   \c_@@_option_layer_context_tl
3444   {
3445     \@@_define_option_commands_and_keyvals:
3446     \@@_define_renderers:
3447     \@@_define_renderer_prototypes:
3448   }
3449 \ExplSyntaxOff

```

### 2.4.3 Themes

In Section 2.2.3, we described the concept of themes. In ConTEXt, we expand on the concept of themes by allowing a theme to be a full-blown ConTEXt module. Specifically, the key-values `theme=<theme name>` and `import=<theme name>` load a ConTEXt module named `t-markdowntheme<munged theme name>.tex` if it exists and a TEX document named `markdowntheme<munged theme name>.tex` otherwise.

Having the Markdown package automatically load either the generic `.tex theme file` or the ConTEXt-specific `t-*.tex` theme file allows developers to have a single *theme file*, when the theme is small or the difference between TEX formats is unimportant, and scale up to separate theme files native to different TEX formats for large multi-format themes, where different code is needed for different TEX formats. To enable code reuse, developers can load the `.tex` theme file from the `t-*.tex` theme file using the `\markdownLoadPlainTeXTheme` macro.

For example, to load a theme named `witiko/tilde` in your document:

```

\usemodule[t][markdown]
\setupmarkdown[import=witiko/tilde]

```



Built-in ConT<sub>E</sub>Xt themes provided with the Markdown package include:

**witiko/markdown/defaults** A ConT<sub>E</sub>Xt theme with the default definitions of token renderer prototypes for plain T<sub>E</sub>X. This theme is loaded automatically together with the package and explicitly loading it has no effect.

```
3450 \startmodule[markdownthemewitiko_markdown_defaults]
3451 \unprotect
```

Please, see Section 3.4.2 for implementation details of the built-in ConT<sub>E</sub>Xt themes.

## 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 T<sub>E</sub>X *token renderers* is performed by the Lua layer. The plain T<sub>E</sub>X layer provides default definitions for the token renderers. The L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt layers correct idiosyncrasies of the respective T<sub>E</sub>X 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 T<sub>E</sub>X, 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 T<sub>E</sub>X writer were hidden behind the converter functions exposed by the Lua interface (see Section 2.1).

```
3452 local upper, format, length =
3453   string.upper, string.format, string.len
3454 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, Cp, any =
3455   lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
3456   lpeg.Cmt, lpeg.Cc, lpeg.Ct, lpeg.B, lpeg.Cs, lpeg.Cp, lpeg.P(1)
```

#### 3.1.1 Utility Functions

This section documents the utility functions used by the plain T<sub>E</sub>X writer and the markdown reader. These functions are encapsulated in the **util** object. The functions were originally located in the `lunamark/util.lua` file in the Lunamark Lua module.

```
3457 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.

```
3458 function util.err(msg, exit_code)
3459   io.stderr:write("markdown.lua: " .. msg .. "\n")
3460   os.exit(exit_code or 1)
3461 end
```

The `util.cache` method used `dir`, `string`, `salt`, and `suffix` to determine a pathname. If a file with such a pathname does not exist, it gets created with `transform(string)` as its content. Regardless, the pathname is then returned.

```
3462 function util.cache(dir, string, salt, transform, suffix)
3463   local digest = md5.sumhexa(string .. (salt or ""))
3464   local name = util.pathname(dir, digest .. suffix)
3465   local file = io.open(name, "r")
3466   if file == nil then -- If no cache entry exists, create a new one.
3467     file = assert(io.open(name, "w"),
3468       [[Could not open file ]] .. name .. [[ for writing]])
3469     local result = string
3470     if transform ~= nil then
3471       result = transform(result)
3472     end
3473     assert(file:write(result))
3474     assert(file:close())
3475   end
3476   return name
3477 end
```

The `util.cache_verbatim` method strips whitespaces from the end of `string` and calls `util.cache` with `dir`, `string`, no salt or transformations, and the `.verbatim` suffix.

```
3478 function util.cache_verbatim(dir, string)
3479   local name = util.cache(dir, string, nil, nil, ".verbatim")
3480   return name
3481 end
```

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

```
3482 function util.table_copy(t)
3483   local u = { }
3484   for k, v in pairs(t) do u[k] = v end
3485   return setmetatable(u, getmetatable(t))
3486 end
```

The `util.encode_json_string` method encodes a string `s` in JSON.

```
3487 function util.encode_json_string(s)
3488   s = s:gsub([[\\]], [[\\]])
3489   s = s:gsub([[\"]], [[\"]])
3490   return [[\"]] .. s .. [[\"]]
3491 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 [11, Chapter 21].

```

3492 function util.expand_tabs_in_line(s, tabstop)
3493     local tab = tabstop or 4
3494     local corr = 0
3495     return (s:gsub("\t", function(p)
3496         local sp = tab - (p - 1 + corr) % tab
3497         corr = corr - 1 + sp
3498         return string.rep(" ", sp)
3499     end))
3500 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.

```

3501 function util.walk(t, f)
3502     local typ = type(t)
3503     if typ == "string" then
3504         f(t)
3505     elseif typ == "table" then
3506         local i = 1
3507         local n
3508         n = t[i]
3509         while n do
3510             util.walk(n, f)
3511             i = i + 1
3512             n = t[i]
3513         end
3514     elseif typ == "function" then
3515         local ok, val = pcall(t)
3516         if ok then
3517             util.walk(val, f)
3518         end
3519     else
3520         f(tostring(t))
3521     end
3522 end

```

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

```

3523 function util.flatten(ary)
3524     local new = {}
3525     for _,v in ipairs(ary) do
3526         if type(v) == "table" then

```

```

3527     for _,w in ipairs(util.flatten(v)) do
3528         new[#new + 1] = w
3529     end
3530     else
3531         new[#new + 1] = v
3532     end
3533 end
3534 return new
3535 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.

```

3536 function util.rope_to_string(rope)
3537     local buffer = {}
3538     util.walk(rope, function(x) buffer[#buffer + 1] = x end)
3539     return table.concat(buffer)
3540 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.

```

3541 function util.rope_last(rope)
3542     if #rope == 0 then
3543         return nil
3544     else
3545         local l = rope[#rope]
3546         if type(l) == "table" then
3547             return util.rope_last(l)
3548         else
3549             return l
3550         end
3551     end
3552 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  $1 \leq i \leq \#ary$ .

```

3553 function util.intersperse(ary, x)
3554     local new = {}
3555     local l = #ary
3556     for i,v in ipairs(ary) do
3557         local n = #new
3558         new[n + 1] = v
3559         if i ~= l then
3560             new[n + 2] = x
3561         end
3562     end
3563     return new
3564 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  $1 \leq i \leq \#ary$ .

```
3565 function util.map(ary, f)
3566   local new = {}
3567   for i,v in ipairs(ary) do
3568     new[i] = f(v)
3569   end
3570   return new
3571 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.

```
3572 function util.escaper(char_escapes, string_escapes)
```

Build a string of escapable characters.

```
3573   local char_escapes_list = ""
3574   for i,_ in pairs(char_escapes) do
3575     char_escapes_list = char_escapes_list .. i
3576   end
```

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

```
3577   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.

```
3578   if string_escapes then
3579     for k,v in pairs(string_escapes) do
3580       escapable = P(k) / v + escapable
3581     end
3582   end
```

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

```
3583   local escape_string = Cs((escapable + any)^0)
```

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

```

3584 return function(s)
3585     return lpeg.match(escape_string, s)
3586 end
3587 end

```

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

```

3588 function util.pathname(dir, file)
3589     if #dir == 0 then
3590         return file
3591     else
3592         return dir .. "/" .. file
3593     end
3594 end

```

The `util.salt` method produces cryptographic salt out of a table of options `options`.

```

3595 function util.salt(options)
3596     local opt_string = {}
3597     for k, _ in pairs(defaultOptions) do
3598         local v = options[k]
3599         if type(v) == "table" then
3600             for _, i in ipairs(v) do
3601                 opt_string[#opt_string+1] = k .. "=" .. tostring(i)
3602             end

```

The `cacheDir` option is disregarded.

```

3603         elseif k ~= "cacheDir" then
3604             opt_string[#opt_string+1] = k .. "=" .. tostring(v)
3605         end
3606     end
3607     table.sort(opt_string)
3608     local salt = table.concat(opt_string, ",")
3609                 .. "," .. metadata.version
3610     return salt
3611 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.

```

3612 local entities = {}
3613
3614 local character_entities = {
3615     ["Tab"] = 9,
3616     ["NewLine"] = 10,

```

3617 ["excl"] = 33,  
3618 ["QUOT"] = 34,  
3619 ["quot"] = 34,  
3620 ["num"] = 35,  
3621 ["dollar"] = 36,  
3622 ["percent"] = 37,  
3623 ["AMP"] = 38,  
3624 ["amp"] = 38,  
3625 ["apos"] = 39,  
3626 ["lpar"] = 40,  
3627 ["rpar"] = 41,  
3628 ["ast"] = 42,  
3629 ["midast"] = 42,  
3630 ["plus"] = 43,  
3631 ["comma"] = 44,  
3632 ["period"] = 46,  
3633 ["sol"] = 47,  
3634 ["colon"] = 58,  
3635 ["semi"] = 59,  
3636 ["LT"] = 60,  
3637 ["lt"] = 60,  
3638 ["nvlt"] = {60, 8402},  
3639 ["bne"] = {61, 8421},  
3640 ["equals"] = 61,  
3641 ["GT"] = 62,  
3642 ["gt"] = 62,  
3643 ["nvgt"] = {62, 8402},  
3644 ["quest"] = 63,  
3645 ["commat"] = 64,  
3646 ["lbrack"] = 91,  
3647 ["lsqb"] = 91,  
3648 ["bsol"] = 92,  
3649 ["rbrack"] = 93,  
3650 ["rsqb"] = 93,  
3651 ["Hat"] = 94,  
3652 ["UnderBar"] = 95,  
3653 ["lowbar"] = 95,  
3654 ["DiacriticalGrave"] = 96,  
3655 ["grave"] = 96,  
3656 ["fjlig"] = {102, 106},  
3657 ["lbrace"] = 123,  
3658 ["lcub"] = 123,  
3659 ["VerticalLine"] = 124,  
3660 ["verbar"] = 124,  
3661 ["vert"] = 124,  
3662 ["rbrace"] = 125,  
3663 ["rcub"] = 125,

3664 ["NonBreakingSpace"] = 160,  
3665 ["nbsp"] = 160,  
3666 ["iexcl"] = 161,  
3667 ["cent"] = 162,  
3668 ["pound"] = 163,  
3669 ["curren"] = 164,  
3670 ["yen"] = 165,  
3671 ["brvbar"] = 166,  
3672 ["sect"] = 167,  
3673 ["Dot"] = 168,  
3674 ["DoubleDot"] = 168,  
3675 ["die"] = 168,  
3676 ["uml"] = 168,  
3677 ["COPY"] = 169,  
3678 ["copy"] = 169,  
3679 ["ordf"] = 170,  
3680 ["laquo"] = 171,  
3681 ["not"] = 172,  
3682 ["shy"] = 173,  
3683 ["REG"] = 174,  
3684 ["circledR"] = 174,  
3685 ["reg"] = 174,  
3686 ["macr"] = 175,  
3687 ["strns"] = 175,  
3688 ["deg"] = 176,  
3689 ["PlusMinus"] = 177,  
3690 ["plusmn"] = 177,  
3691 ["pm"] = 177,  
3692 ["sup2"] = 178,  
3693 ["sup3"] = 179,  
3694 ["DiacriticalAcute"] = 180,  
3695 ["acute"] = 180,  
3696 ["micro"] = 181,  
3697 ["para"] = 182,  
3698 ["CenterDot"] = 183,  
3699 ["centerdot"] = 183,  
3700 ["middot"] = 183,  
3701 ["Cedilla"] = 184,  
3702 ["cedil"] = 184,  
3703 ["sup1"] = 185,  
3704 ["ordm"] = 186,  
3705 ["raquo"] = 187,  
3706 ["frac14"] = 188,  
3707 ["frac12"] = 189,  
3708 ["half"] = 189,  
3709 ["frac34"] = 190,  
3710 ["iquest"] = 191,



3711 ["Agrave"] = 192,  
3712 ["Aacute"] = 193,  
3713 ["Acirc"] = 194,  
3714 ["Atilde"] = 195,  
3715 ["Auml"] = 196,  
3716 ["Aring"] = 197,  
3717 ["angst"] = 197,  
3718 ["AElig"] = 198,  
3719 ["Ccedil"] = 199,  
3720 ["Egrave"] = 200,  
3721 ["Eacute"] = 201,  
3722 ["Ecirc"] = 202,  
3723 ["Euml"] = 203,  
3724 ["Igrave"] = 204,  
3725 ["Iacute"] = 205,  
3726 ["Icirc"] = 206,  
3727 ["Iuml"] = 207,  
3728 ["ETH"] = 208,  
3729 ["Ntilde"] = 209,  
3730 ["Ograve"] = 210,  
3731 ["Oacute"] = 211,  
3732 ["Ocirc"] = 212,  
3733 ["Otilde"] = 213,  
3734 ["Ouml"] = 214,  
3735 ["times"] = 215,  
3736 ["Oslash"] = 216,  
3737 ["Ugrave"] = 217,  
3738 ["Uacute"] = 218,  
3739 ["Ucirc"] = 219,  
3740 ["Uuml"] = 220,  
3741 ["Yacute"] = 221,  
3742 ["THORN"] = 222,  
3743 ["szlig"] = 223,  
3744 ["agrave"] = 224,  
3745 ["aacute"] = 225,  
3746 ["acirc"] = 226,  
3747 ["atilde"] = 227,  
3748 ["auml"] = 228,  
3749 ["aring"] = 229,  
3750 ["aelig"] = 230,  
3751 ["ccedil"] = 231,  
3752 ["egrave"] = 232,  
3753 ["eacute"] = 233,  
3754 ["ecirc"] = 234,  
3755 ["euml"] = 235,  
3756 ["igrave"] = 236,  
3757 ["iacute"] = 237,

3758 ["icirc"] = 238,  
3759 ["iuml"] = 239,  
3760 ["eth"] = 240,  
3761 ["ntilde"] = 241,  
3762 ["ograve"] = 242,  
3763 ["oacute"] = 243,  
3764 ["ocirc"] = 244,  
3765 ["otilde"] = 245,  
3766 ["ouml"] = 246,  
3767 ["div"] = 247,  
3768 ["divide"] = 247,  
3769 ["oslash"] = 248,  
3770 ["ugrave"] = 249,  
3771 ["uacute"] = 250,  
3772 ["ucirc"] = 251,  
3773 ["uuml"] = 252,  
3774 ["yacute"] = 253,  
3775 ["thorn"] = 254,  
3776 ["yuml"] = 255,  
3777 ["Amacr"] = 256,  
3778 ["amacr"] = 257,  
3779 ["Abreve"] = 258,  
3780 ["abreve"] = 259,  
3781 ["Aogon"] = 260,  
3782 ["aogon"] = 261,  
3783 ["Cacute"] = 262,  
3784 ["cacute"] = 263,  
3785 ["Ccirc"] = 264,  
3786 ["ccirc"] = 265,  
3787 ["Cdot"] = 266,  
3788 ["cdot"] = 267,  
3789 ["Ccaron"] = 268,  
3790 ["ccaron"] = 269,  
3791 ["Dcaron"] = 270,  
3792 ["dcaron"] = 271,  
3793 ["Dstrok"] = 272,  
3794 ["dstrok"] = 273,  
3795 ["Emacr"] = 274,  
3796 ["emacr"] = 275,  
3797 ["Edot"] = 278,  
3798 ["edot"] = 279,  
3799 ["Eogon"] = 280,  
3800 ["eogon"] = 281,  
3801 ["Ecaron"] = 282,  
3802 ["ecaron"] = 283,  
3803 ["Gcirc"] = 284,  
3804 ["gcirc"] = 285,

3805 ["Gbreve"] = 286,  
3806 ["gbreve"] = 287,  
3807 ["Gdot"] = 288,  
3808 ["gdot"] = 289,  
3809 ["Gcedil"] = 290,  
3810 ["Hcirc"] = 292,  
3811 ["hcirc"] = 293,  
3812 ["Hstrook"] = 294,  
3813 ["hstrook"] = 295,  
3814 ["Itilde"] = 296,  
3815 ["itilde"] = 297,  
3816 ["Imacr"] = 298,  
3817 ["imacr"] = 299,  
3818 ["Iogon"] = 302,  
3819 ["iogon"] = 303,  
3820 ["Idot"] = 304,  
3821 ["imath"] = 305,  
3822 ["inodot"] = 305,  
3823 ["IJlig"] = 306,  
3824 ["ijlig"] = 307,  
3825 ["Jcirc"] = 308,  
3826 ["jcirc"] = 309,  
3827 ["Kcedil"] = 310,  
3828 ["kcedil"] = 311,  
3829 ["kgreen"] = 312,  
3830 ["Lacute"] = 313,  
3831 ["lacute"] = 314,  
3832 ["Lcedil"] = 315,  
3833 ["lcedil"] = 316,  
3834 ["Lcaron"] = 317,  
3835 ["lcaron"] = 318,  
3836 ["Lmidot"] = 319,  
3837 ["lmidot"] = 320,  
3838 ["Lstrook"] = 321,  
3839 ["lstrook"] = 322,  
3840 ["Nacute"] = 323,  
3841 ["nacute"] = 324,  
3842 ["Ncedil"] = 325,  
3843 ["ncedil"] = 326,  
3844 ["Ncaron"] = 327,  
3845 ["ncaron"] = 328,  
3846 ["napos"] = 329,  
3847 ["ENG"] = 330,  
3848 ["eng"] = 331,  
3849 ["Omacr"] = 332,  
3850 ["omacr"] = 333,  
3851 ["Odblac"] = 336,

3852 ["odblac"] = 337,  
3853 ["OElig"] = 338,  
3854 ["oelig"] = 339,  
3855 ["Racute"] = 340,  
3856 ["racute"] = 341,  
3857 ["Rcedil"] = 342,  
3858 ["rcedil"] = 343,  
3859 ["Rcaron"] = 344,  
3860 ["rcaron"] = 345,  
3861 ["Sacute"] = 346,  
3862 ["sacute"] = 347,  
3863 ["Scirc"] = 348,  
3864 ["scirc"] = 349,  
3865 ["Scedil"] = 350,  
3866 ["scedil"] = 351,  
3867 ["Scaron"] = 352,  
3868 ["scaron"] = 353,  
3869 ["Tcedil"] = 354,  
3870 ["tcedil"] = 355,  
3871 ["Tcaron"] = 356,  
3872 ["tcaron"] = 357,  
3873 ["Tstrok"] = 358,  
3874 ["tstrok"] = 359,  
3875 ["Utilde"] = 360,  
3876 ["utilde"] = 361,  
3877 ["Umacr"] = 362,  
3878 ["umacr"] = 363,  
3879 ["Ubreve"] = 364,  
3880 ["ubreve"] = 365,  
3881 ["Uring"] = 366,  
3882 ["uring"] = 367,  
3883 ["Udblac"] = 368,  
3884 ["udblac"] = 369,  
3885 ["Uogon"] = 370,  
3886 ["uogon"] = 371,  
3887 ["Wcirc"] = 372,  
3888 ["wcirc"] = 373,  
3889 ["Ycirc"] = 374,  
3890 ["ycirc"] = 375,  
3891 ["Yuml"] = 376,  
3892 ["Zacute"] = 377,  
3893 ["zacute"] = 378,  
3894 ["Zdot"] = 379,  
3895 ["zdot"] = 380,  
3896 ["Zcaron"] = 381,  
3897 ["zcaron"] = 382,  
3898 ["fnof"] = 402,

3899 ["imped"] = 437,  
3900 ["gacute"] = 501,  
3901 ["jmath"] = 567,  
3902 ["circ"] = 710,  
3903 ["Hacek"] = 711,  
3904 ["caron"] = 711,  
3905 ["Breve"] = 728,  
3906 ["breve"] = 728,  
3907 ["DiacriticalDot"] = 729,  
3908 ["dot"] = 729,  
3909 ["ring"] = 730,  
3910 ["ogon"] = 731,  
3911 ["DiacriticalTilde"] = 732,  
3912 ["tilde"] = 732,  
3913 ["DiacriticalDoubleAcute"] = 733,  
3914 ["dblac"] = 733,  
3915 ["DownBreve"] = 785,  
3916 ["Alpha"] = 913,  
3917 ["Beta"] = 914,  
3918 ["Gamma"] = 915,  
3919 ["Delta"] = 916,  
3920 ["Epsilon"] = 917,  
3921 ["Zeta"] = 918,  
3922 ["Eta"] = 919,  
3923 ["Theta"] = 920,  
3924 ["Iota"] = 921,  
3925 ["Kappa"] = 922,  
3926 ["Lambda"] = 923,  
3927 ["Mu"] = 924,  
3928 ["Nu"] = 925,  
3929 ["Xi"] = 926,  
3930 ["Omicron"] = 927,  
3931 ["Pi"] = 928,  
3932 ["Rho"] = 929,  
3933 ["Sigma"] = 931,  
3934 ["Tau"] = 932,  
3935 ["Upsilon"] = 933,  
3936 ["Phi"] = 934,  
3937 ["Chi"] = 935,  
3938 ["Psi"] = 936,  
3939 ["Omega"] = 937,  
3940 ["ohm"] = 937,  
3941 ["alpha"] = 945,  
3942 ["beta"] = 946,  
3943 ["gamma"] = 947,  
3944 ["delta"] = 948,  
3945 ["epsi"] = 949,

3946 ["epsilon"] = 949,  
3947 ["zeta"] = 950,  
3948 ["eta"] = 951,  
3949 ["theta"] = 952,  
3950 ["iota"] = 953,  
3951 ["kappa"] = 954,  
3952 ["lambda"] = 955,  
3953 ["mu"] = 956,  
3954 ["nu"] = 957,  
3955 ["xi"] = 958,  
3956 ["omicron"] = 959,  
3957 ["pi"] = 960,  
3958 ["rho"] = 961,  
3959 ["sigmaf"] = 962,  
3960 ["sigmav"] = 962,  
3961 ["varsigma"] = 962,  
3962 ["sigma"] = 963,  
3963 ["tau"] = 964,  
3964 ["upsilon"] = 965,  
3965 ["upsilon"] = 965,  
3966 ["phi"] = 966,  
3967 ["chi"] = 967,  
3968 ["psi"] = 968,  
3969 ["omega"] = 969,  
3970 ["thetasym"] = 977,  
3971 ["thetav"] = 977,  
3972 ["vartheta"] = 977,  
3973 ["Upsilon"] = 978,  
3974 ["upsih"] = 978,  
3975 ["phiv"] = 981,  
3976 ["straightphi"] = 981,  
3977 ["varphi"] = 981,  
3978 ["piv"] = 982,  
3979 ["varpi"] = 982,  
3980 ["Gammad"] = 988,  
3981 ["digamma"] = 989,  
3982 ["gammad"] = 989,  
3983 ["kappav"] = 1008,  
3984 ["varkappa"] = 1008,  
3985 ["rhov"] = 1009,  
3986 ["varrho"] = 1009,  
3987 ["epsiv"] = 1013,  
3988 ["straightepsilon"] = 1013,  
3989 ["varepsilon"] = 1013,  
3990 ["backepsilon"] = 1014,  
3991 ["bepsi"] = 1014,  
3992 ["IOcy"] = 1025,

3993 ["DJcy"] = 1026,  
3994 ["GJcy"] = 1027,  
3995 ["Jukcy"] = 1028,  
3996 ["DScy"] = 1029,  
3997 ["Iukcy"] = 1030,  
3998 ["YIcy"] = 1031,  
3999 ["Jsercy"] = 1032,  
4000 ["LJcy"] = 1033,  
4001 ["NJcy"] = 1034,  
4002 ["TSHcy"] = 1035,  
4003 ["KJcy"] = 1036,  
4004 ["Ubrcy"] = 1038,  
4005 ["DZcy"] = 1039,  
4006 ["Acy"] = 1040,  
4007 ["Bcy"] = 1041,  
4008 ["Vcy"] = 1042,  
4009 ["Gcy"] = 1043,  
4010 ["Dcy"] = 1044,  
4011 ["IEcy"] = 1045,  
4012 ["ZHcy"] = 1046,  
4013 ["Zcy"] = 1047,  
4014 ["Icy"] = 1048,  
4015 ["Jcy"] = 1049,  
4016 ["Kcy"] = 1050,  
4017 ["Lcy"] = 1051,  
4018 ["Mcy"] = 1052,  
4019 ["Ncy"] = 1053,  
4020 ["Ocy"] = 1054,  
4021 ["Pcy"] = 1055,  
4022 ["Rcy"] = 1056,  
4023 ["Scy"] = 1057,  
4024 ["Tcy"] = 1058,  
4025 ["Ucy"] = 1059,  
4026 ["Fcy"] = 1060,  
4027 ["KHcy"] = 1061,  
4028 ["TScy"] = 1062,  
4029 ["CHcy"] = 1063,  
4030 ["SHcy"] = 1064,  
4031 ["SHCHcy"] = 1065,  
4032 ["HARDcy"] = 1066,  
4033 ["Ycy"] = 1067,  
4034 ["SOFTcy"] = 1068,  
4035 ["Ecy"] = 1069,  
4036 ["YUcy"] = 1070,  
4037 ["YAcy"] = 1071,  
4038 ["acy"] = 1072,  
4039 ["bcy"] = 1073,

4040 ["vcy"] = 1074,  
4041 ["gcy"] = 1075,  
4042 ["dcy"] = 1076,  
4043 ["iecy"] = 1077,  
4044 ["zhcy"] = 1078,  
4045 ["zcy"] = 1079,  
4046 ["icy"] = 1080,  
4047 ["jcy"] = 1081,  
4048 ["kcy"] = 1082,  
4049 ["lcy"] = 1083,  
4050 ["mcy"] = 1084,  
4051 ["ncy"] = 1085,  
4052 ["ocy"] = 1086,  
4053 ["pcy"] = 1087,  
4054 ["rcy"] = 1088,  
4055 ["scy"] = 1089,  
4056 ["tcy"] = 1090,  
4057 ["ucy"] = 1091,  
4058 ["fcy"] = 1092,  
4059 ["khcy"] = 1093,  
4060 ["tscy"] = 1094,  
4061 ["chcy"] = 1095,  
4062 ["shcy"] = 1096,  
4063 ["shchcy"] = 1097,  
4064 ["hardcy"] = 1098,  
4065 ["ycy"] = 1099,  
4066 ["softcy"] = 1100,  
4067 ["ecy"] = 1101,  
4068 ["yucy"] = 1102,  
4069 ["yacy"] = 1103,  
4070 ["iocy"] = 1105,  
4071 ["djcy"] = 1106,  
4072 ["gjcy"] = 1107,  
4073 ["jukcy"] = 1108,  
4074 ["dscy"] = 1109,  
4075 ["iukcy"] = 1110,  
4076 ["yicy"] = 1111,  
4077 ["jsercy"] = 1112,  
4078 ["ljcy"] = 1113,  
4079 ["njcy"] = 1114,  
4080 ["tshcy"] = 1115,  
4081 ["kjcy"] = 1116,  
4082 ["ubrcy"] = 1118,  
4083 ["dzcycy"] = 1119,  
4084 ["ensp"] = 8194,  
4085 ["emsp"] = 8195,  
4086 ["emsp13"] = 8196,



4087 ["emsp14"] = 8197,  
4088 ["numsp"] = 8199,  
4089 ["puncsp"] = 8200,  
4090 ["ThinSpace"] = 8201,  
4091 ["thinsp"] = 8201,  
4092 ["VeryThinSpace"] = 8202,  
4093 ["hairsp"] = 8202,  
4094 ["NegativeMediumSpace"] = 8203,  
4095 ["NegativeThickSpace"] = 8203,  
4096 ["NegativeThinSpace"] = 8203,  
4097 ["NegativeVeryThinSpace"] = 8203,  
4098 ["ZeroWidthSpace"] = 8203,  
4099 ["zwnj"] = 8204,  
4100 ["zwj"] = 8205,  
4101 ["lrm"] = 8206,  
4102 ["rlm"] = 8207,  
4103 ["dash"] = 8208,  
4104 ["hyphen"] = 8208,  
4105 ["ndash"] = 8211,  
4106 ["mdash"] = 8212,  
4107 ["horbar"] = 8213,  
4108 ["Verbar"] = 8214,  
4109 ["Vert"] = 8214,  
4110 ["OpenCurlyQuote"] = 8216,  
4111 ["lsquo"] = 8216,  
4112 ["CloseCurlyQuote"] = 8217,  
4113 ["rsquo"] = 8217,  
4114 ["rsquor"] = 8217,  
4115 ["lsquor"] = 8218,  
4116 ["sbquo"] = 8218,  
4117 ["OpenCurlyDoubleQuote"] = 8220,  
4118 ["ldquo"] = 8220,  
4119 ["CloseCurlyDoubleQuote"] = 8221,  
4120 ["rdquo"] = 8221,  
4121 ["rdquor"] = 8221,  
4122 ["bdquo"] = 8222,  
4123 ["ldquor"] = 8222,  
4124 ["dagger"] = 8224,  
4125 ["Dagger"] = 8225,  
4126 ["ddagger"] = 8225,  
4127 ["bull"] = 8226,  
4128 ["bullet"] = 8226,  
4129 ["nldr"] = 8229,  
4130 ["hellip"] = 8230,  
4131 ["mldr"] = 8230,  
4132 ["permil"] = 8240,  
4133 ["pertenk"] = 8241,

```

4134 ["prime"] = 8242,
4135 ["Prime"] = 8243,
4136 ["tprime"] = 8244,
4137 ["backprime"] = 8245,
4138 ["bprime"] = 8245,
4139 ["lsaquo"] = 8249,
4140 ["rsaquo"] = 8250,
4141 ["OverBar"] = 8254,
4142 ["oline"] = 8254,
4143 ["caret"] = 8257,
4144 ["hybull"] = 8259,
4145 ["frasl"] = 8260,
4146 ["bsemi"] = 8271,
4147 ["qprime"] = 8279,
4148 ["MediumSpace"] = 8287,
4149 ["ThickSpace"] = {8287, 8202},
4150 ["NoBreak"] = 8288,
4151 ["ApplyFunction"] = 8289,
4152 ["af"] = 8289,
4153 ["InvisibleTimes"] = 8290,
4154 ["it"] = 8290,
4155 ["InvisibleComma"] = 8291,
4156 ["ic"] = 8291,
4157 ["euro"] = 8364,
4158 ["TripleDot"] = 8411,
4159 ["tdot"] = 8411,
4160 ["DotDot"] = 8412,
4161 ["Copf"] = 8450,
4162 ["complexes"] = 8450,
4163 ["incare"] = 8453,
4164 ["gscr"] = 8458,
4165 ["HilbertSpace"] = 8459,
4166 ["Hscr"] = 8459,
4167 ["hamilt"] = 8459,
4168 ["Hfr"] = 8460,
4169 ["Poincareplane"] = 8460,
4170 ["Hopf"] = 8461,
4171 ["quaternions"] = 8461,
4172 ["planckh"] = 8462,
4173 ["hbar"] = 8463,
4174 ["hslash"] = 8463,
4175 ["planck"] = 8463,
4176 ["plankv"] = 8463,
4177 ["Iscr"] = 8464,
4178 ["imagline"] = 8464,
4179 ["Ifr"] = 8465,
4180 ["Im"] = 8465,

```

4181 ["image"] = 8465,  
4182 ["imagpart"] = 8465,  
4183 ["Laplacetrif"] = 8466,  
4184 ["Lscr"] = 8466,  
4185 ["lagran"] = 8466,  
4186 ["ell"] = 8467,  
4187 ["Nopf"] = 8469,  
4188 ["naturals"] = 8469,  
4189 ["numero"] = 8470,  
4190 ["copysr"] = 8471,  
4191 ["weierp"] = 8472,  
4192 ["wp"] = 8472,  
4193 ["Popf"] = 8473,  
4194 ["primes"] = 8473,  
4195 ["Qopf"] = 8474,  
4196 ["rationals"] = 8474,  
4197 ["Rscr"] = 8475,  
4198 ["realine"] = 8475,  
4199 ["Re"] = 8476,  
4200 ["Rfr"] = 8476,  
4201 ["real"] = 8476,  
4202 ["realpart"] = 8476,  
4203 ["Ropf"] = 8477,  
4204 ["reals"] = 8477,  
4205 ["rx"] = 8478,  
4206 ["TRADE"] = 8482,  
4207 ["trade"] = 8482,  
4208 ["Zopf"] = 8484,  
4209 ["integers"] = 8484,  
4210 ["mho"] = 8487,  
4211 ["Zfr"] = 8488,  
4212 ["zeetrif"] = 8488,  
4213 ["iiota"] = 8489,  
4214 ["Bernoullis"] = 8492,  
4215 ["Bscr"] = 8492,  
4216 ["bernou"] = 8492,  
4217 ["Cayleys"] = 8493,  
4218 ["Cfr"] = 8493,  
4219 ["escr"] = 8495,  
4220 ["Escr"] = 8496,  
4221 ["expectation"] = 8496,  
4222 ["Fouriertrif"] = 8497,  
4223 ["Fscr"] = 8497,  
4224 ["Mellintrif"] = 8499,  
4225 ["Mscr"] = 8499,  
4226 ["phmmat"] = 8499,  
4227 ["order"] = 8500,

4228 ["orderof"] = 8500,  
 4229 ["oscr"] = 8500,  
 4230 ["alefsym"] = 8501,  
 4231 ["aleph"] = 8501,  
 4232 ["beth"] = 8502,  
 4233 ["gimel"] = 8503,  
 4234 ["daleth"] = 8504,  
 4235 ["CapitalDifferentialD"] = 8517,  
 4236 ["DD"] = 8517,  
 4237 ["DifferentialD"] = 8518,  
 4238 ["dd"] = 8518,  
 4239 ["ExponentialE"] = 8519,  
 4240 ["ee"] = 8519,  
 4241 ["exponentiale"] = 8519,  
 4242 ["ImaginaryI"] = 8520,  
 4243 ["ii"] = 8520,  
 4244 ["frac13"] = 8531,  
 4245 ["frac23"] = 8532,  
 4246 ["frac15"] = 8533,  
 4247 ["frac25"] = 8534,  
 4248 ["frac35"] = 8535,  
 4249 ["frac45"] = 8536,  
 4250 ["frac16"] = 8537,  
 4251 ["frac56"] = 8538,  
 4252 ["frac18"] = 8539,  
 4253 ["frac38"] = 8540,  
 4254 ["frac58"] = 8541,  
 4255 ["frac78"] = 8542,  
 4256 ["LeftArrow"] = 8592,  
 4257 ["ShortLeftArrow"] = 8592,  
 4258 ["larr"] = 8592,  
 4259 ["leftarrow"] = 8592,  
 4260 ["slarr"] = 8592,  
 4261 ["ShortUpArrow"] = 8593,  
 4262 ["UpArrow"] = 8593,  
 4263 ["uarr"] = 8593,  
 4264 ["uparrow"] = 8593,  
 4265 ["RightArrow"] = 8594,  
 4266 ["ShortRightArrow"] = 8594,  
 4267 ["rarr"] = 8594,  
 4268 ["rightarrow"] = 8594,  
 4269 ["srarr"] = 8594,  
 4270 ["DownArrow"] = 8595,  
 4271 ["ShortDownArrow"] = 8595,  
 4272 ["darr"] = 8595,  
 4273 ["downarrow"] = 8595,  
 4274 ["LeftRightArrow"] = 8596,

4275 ["harr"] = 8596,  
 4276 ["leftrightarrow"] = 8596,  
 4277 ["UpDownArrow"] = 8597,  
 4278 ["updownarrow"] = 8597,  
 4279 ["varr"] = 8597,  
 4280 ["UpperLeftArrow"] = 8598,  
 4281 ["nwarr"] = 8598,  
 4282 ["nwarrow"] = 8598,  
 4283 ["UpperRightArrow"] = 8599,  
 4284 ["nearr"] = 8599,  
 4285 ["nearrow"] = 8599,  
 4286 ["LowerRightArrow"] = 8600,  
 4287 ["searr"] = 8600,  
 4288 ["searrow"] = 8600,  
 4289 ["LowerLeftArrow"] = 8601,  
 4290 ["swarr"] = 8601,  
 4291 ["swarrow"] = 8601,  
 4292 ["nlarr"] = 8602,  
 4293 ["nleftarrow"] = 8602,  
 4294 ["nrarr"] = 8603,  
 4295 ["nrightarrow"] = 8603,  
 4296 ["nrarrw"] = {8605, 824},  
 4297 ["rarrw"] = 8605,  
 4298 ["rightsquigarrow"] = 8605,  
 4299 ["Larr"] = 8606,  
 4300 ["twoheadleftarrow"] = 8606,  
 4301 ["Uarr"] = 8607,  
 4302 ["Rarr"] = 8608,  
 4303 ["twoheadrightarrow"] = 8608,  
 4304 ["Darr"] = 8609,  
 4305 ["larrtl"] = 8610,  
 4306 ["leftarrowtail"] = 8610,  
 4307 ["rarrtl"] = 8611,  
 4308 ["rightarrowtail"] = 8611,  
 4309 ["LeftTeeArrow"] = 8612,  
 4310 ["mapstoleft"] = 8612,  
 4311 ["UpTeeArrow"] = 8613,  
 4312 ["mapstoup"] = 8613,  
 4313 ["RightTeeArrow"] = 8614,  
 4314 ["map"] = 8614,  
 4315 ["mapsto"] = 8614,  
 4316 ["DownTeeArrow"] = 8615,  
 4317 ["mapstodown"] = 8615,  
 4318 ["hookleftarrow"] = 8617,  
 4319 ["larrhk"] = 8617,  
 4320 ["hookrightarrow"] = 8618,  
 4321 ["rarrhk"] = 8618,

4322 ["larrlp"] = 8619,  
 4323 ["looparrowleft"] = 8619,  
 4324 ["looparrowright"] = 8620,  
 4325 ["rarrlp"] = 8620,  
 4326 ["harrw"] = 8621,  
 4327 ["leftrightsquigarrow"] = 8621,  
 4328 ["nharr"] = 8622,  
 4329 ["nletrightarrow"] = 8622,  
 4330 ["Lsh"] = 8624,  
 4331 ["lsh"] = 8624,  
 4332 ["Rsh"] = 8625,  
 4333 ["rsh"] = 8625,  
 4334 ["ldsh"] = 8626,  
 4335 ["rdsh"] = 8627,  
 4336 ["crarr"] = 8629,  
 4337 ["cularr"] = 8630,  
 4338 ["curvearrowleft"] = 8630,  
 4339 ["curarr"] = 8631,  
 4340 ["curvearrowright"] = 8631,  
 4341 ["circlearrowleft"] = 8634,  
 4342 ["olarr"] = 8634,  
 4343 ["circlearrowright"] = 8635,  
 4344 ["orarr"] = 8635,  
 4345 ["LeftVector"] = 8636,  
 4346 ["leftharpoonup"] = 8636,  
 4347 ["lharu"] = 8636,  
 4348 ["DownLeftVector"] = 8637,  
 4349 ["leftharpoondown"] = 8637,  
 4350 ["lhard"] = 8637,  
 4351 ["RightUpVector"] = 8638,  
 4352 ["uharr"] = 8638,  
 4353 ["upharpoonright"] = 8638,  
 4354 ["LeftUpVector"] = 8639,  
 4355 ["uharl"] = 8639,  
 4356 ["upharpoonleft"] = 8639,  
 4357 ["RightVector"] = 8640,  
 4358 ["rharu"] = 8640,  
 4359 ["rightharpoonup"] = 8640,  
 4360 ["DownRightVector"] = 8641,  
 4361 ["rhard"] = 8641,  
 4362 ["rightharpoondown"] = 8641,  
 4363 ["RightDownVector"] = 8642,  
 4364 ["dharr"] = 8642,  
 4365 ["downharpoonright"] = 8642,  
 4366 ["LeftDownVector"] = 8643,  
 4367 ["dharl"] = 8643,  
 4368 ["downharpoonleft"] = 8643,

4369 ["RightArrowLeftArrow"] = 8644,  
 4370 ["rightleftarrows"] = 8644,  
 4371 ["rlarr"] = 8644,  
 4372 ["UpArrowDownArrow"] = 8645,  
 4373 ["udarr"] = 8645,  
 4374 ["LeftArrowRightArrow"] = 8646,  
 4375 ["leftrightharpoons"] = 8646,  
 4376 ["lrarr"] = 8646,  
 4377 ["leftleftarrows"] = 8647,  
 4378 ["llarr"] = 8647,  
 4379 ["upuparrows"] = 8648,  
 4380 ["uuarr"] = 8648,  
 4381 ["rightrightarrows"] = 8649,  
 4382 ["rrarr"] = 8649,  
 4383 ["ddarr"] = 8650,  
 4384 ["downdownarrows"] = 8650,  
 4385 ["ReverseEquilibrium"] = 8651,  
 4386 ["leftrightharpoons"] = 8651,  
 4387 ["lrhar"] = 8651,  
 4388 ["Equilibrium"] = 8652,  
 4389 ["rightleftharpoons"] = 8652,  
 4390 ["rlhar"] = 8652,  
 4391 ["nLeftarrow"] = 8653,  
 4392 ["nlArr"] = 8653,  
 4393 ["nLeftrightarrow"] = 8654,  
 4394 ["nhArr"] = 8654,  
 4395 ["nRightarrow"] = 8655,  
 4396 ["nrArr"] = 8655,  
 4397 ["DoubleLeftArrow"] = 8656,  
 4398 ["Leftarrow"] = 8656,  
 4399 ["lArr"] = 8656,  
 4400 ["DoubleUpArrow"] = 8657,  
 4401 ["Uparrow"] = 8657,  
 4402 ["uArr"] = 8657,  
 4403 ["DoubleRightArrow"] = 8658,  
 4404 ["Implies"] = 8658,  
 4405 ["Rightarrow"] = 8658,  
 4406 ["rArr"] = 8658,  
 4407 ["DoubleDownArrow"] = 8659,  
 4408 ["Downarrow"] = 8659,  
 4409 ["dArr"] = 8659,  
 4410 ["DoubleLeftRightArrow"] = 8660,  
 4411 ["Leftrightarrow"] = 8660,  
 4412 ["hArr"] = 8660,  
 4413 ["iff"] = 8660,  
 4414 ["DoubleUpDownArrow"] = 8661,  
 4415 ["Updownarrow"] = 8661,

4416 ["vArr"] = 8661,  
 4417 ["nwArr"] = 8662,  
 4418 ["neArr"] = 8663,  
 4419 ["seArr"] = 8664,  
 4420 ["swArr"] = 8665,  
 4421 ["Lleftarrow"] = 8666,  
 4422 ["lAarr"] = 8666,  
 4423 ["Rrightarrow"] = 8667,  
 4424 ["rAarr"] = 8667,  
 4425 ["zigrarr"] = 8669,  
 4426 ["LeftArrowBar"] = 8676,  
 4427 ["larrb"] = 8676,  
 4428 ["RightArrowBar"] = 8677,  
 4429 ["rarrb"] = 8677,  
 4430 ["DownArrowUpArrow"] = 8693,  
 4431 ["duarr"] = 8693,  
 4432 ["loarr"] = 8701,  
 4433 ["roarr"] = 8702,  
 4434 ["hoarr"] = 8703,  
 4435 ["ForAll"] = 8704,  
 4436 ["forall"] = 8704,  
 4437 ["comp"] = 8705,  
 4438 ["complement"] = 8705,  
 4439 ["PartialD"] = 8706,  
 4440 ["npart"] = {8706, 824},  
 4441 ["part"] = 8706,  
 4442 ["Exists"] = 8707,  
 4443 ["exist"] = 8707,  
 4444 ["NotExists"] = 8708,  
 4445 ["nexist"] = 8708,  
 4446 ["nexists"] = 8708,  
 4447 ["empty"] = 8709,  
 4448 ["emptyset"] = 8709,  
 4449 ["emptyv"] = 8709,  
 4450 ["varnothing"] = 8709,  
 4451 ["Del"] = 8711,  
 4452 ["nabla"] = 8711,  
 4453 ["Element"] = 8712,  
 4454 ["in"] = 8712,  
 4455 ["isin"] = 8712,  
 4456 ["isinv"] = 8712,  
 4457 ["NotElement"] = 8713,  
 4458 ["notin"] = 8713,  
 4459 ["notinva"] = 8713,  
 4460 ["ReverseElement"] = 8715,  
 4461 ["SuchThat"] = 8715,  
 4462 ["ni"] = 8715,



4463 ["niv"] = 8715,  
 4464 ["NotReverseElement"] = 8716,  
 4465 ["notni"] = 8716,  
 4466 ["notniva"] = 8716,  
 4467 ["Product"] = 8719,  
 4468 ["prod"] = 8719,  
 4469 ["Coproduct"] = 8720,  
 4470 ["coprod"] = 8720,  
 4471 ["Sum"] = 8721,  
 4472 ["sum"] = 8721,  
 4473 ["minus"] = 8722,  
 4474 ["MinusPlus"] = 8723,  
 4475 ["mnplus"] = 8723,  
 4476 ["mp"] = 8723,  
 4477 ["dotplus"] = 8724,  
 4478 ["plusdo"] = 8724,  
 4479 ["Backslash"] = 8726,  
 4480 ["setminus"] = 8726,  
 4481 ["setmn"] = 8726,  
 4482 ["smallsetminus"] = 8726,  
 4483 ["ssetmn"] = 8726,  
 4484 ["lowast"] = 8727,  
 4485 ["SmallCircle"] = 8728,  
 4486 ["compfn"] = 8728,  
 4487 ["Sqrt"] = 8730,  
 4488 ["radic"] = 8730,  
 4489 ["Proportional"] = 8733,  
 4490 ["prop"] = 8733,  
 4491 ["propto"] = 8733,  
 4492 ["varpropto"] = 8733,  
 4493 ["vprop"] = 8733,  
 4494 ["infin"] = 8734,  
 4495 ["angrt"] = 8735,  
 4496 ["ang"] = 8736,  
 4497 ["angle"] = 8736,  
 4498 ["nang"] = {8736, 8402},  
 4499 ["angmsd"] = 8737,  
 4500 ["measuredangle"] = 8737,  
 4501 ["angsph"] = 8738,  
 4502 ["VerticalBar"] = 8739,  
 4503 ["mid"] = 8739,  
 4504 ["shortmid"] = 8739,  
 4505 ["smid"] = 8739,  
 4506 ["NotVerticalBar"] = 8740,  
 4507 ["nmid"] = 8740,  
 4508 ["nshortmid"] = 8740,  
 4509 ["nsmid"] = 8740,

4510 ["DoubleVerticalBar"] = 8741,  
 4511 ["par"] = 8741,  
 4512 ["parallel"] = 8741,  
 4513 ["shortparallel"] = 8741,  
 4514 ["spar"] = 8741,  
 4515 ["NotDoubleVerticalBar"] = 8742,  
 4516 ["npar"] = 8742,  
 4517 ["nparallel"] = 8742,  
 4518 ["nshortparallel"] = 8742,  
 4519 ["nspar"] = 8742,  
 4520 ["and"] = 8743,  
 4521 ["wedge"] = 8743,  
 4522 ["or"] = 8744,  
 4523 ["vee"] = 8744,  
 4524 ["cap"] = 8745,  
 4525 ["caps"] = {8745, 65024},  
 4526 ["cup"] = 8746,  
 4527 ["cups"] = {8746, 65024},  
 4528 ["Integral"] = 8747,  
 4529 ["int"] = 8747,  
 4530 ["Int"] = 8748,  
 4531 ["iiint"] = 8749,  
 4532 ["tint"] = 8749,  
 4533 ["ContourIntegral"] = 8750,  
 4534 ["conint"] = 8750,  
 4535 ["oint"] = 8750,  
 4536 ["Conint"] = 8751,  
 4537 ["DoubleContourIntegral"] = 8751,  
 4538 ["Cconint"] = 8752,  
 4539 ["cwint"] = 8753,  
 4540 ["ClockwiseContourIntegral"] = 8754,  
 4541 ["cwconint"] = 8754,  
 4542 ["CounterClockwiseContourIntegral"] = 8755,  
 4543 ["awconint"] = 8755,  
 4544 ["Therefore"] = 8756,  
 4545 ["there4"] = 8756,  
 4546 ["therefore"] = 8756,  
 4547 ["Because"] = 8757,  
 4548 ["because"] = 8757,  
 4549 ["because"] = 8757,  
 4550 ["ratio"] = 8758,  
 4551 ["Colon"] = 8759,  
 4552 ["Proportion"] = 8759,  
 4553 ["dotminus"] = 8760,  
 4554 ["minusd"] = 8760,  
 4555 ["mDDot"] = 8762,  
 4556 ["homtht"] = 8763,

4557 ["Tilde"] = 8764,  
4558 ["nvsim"] = {8764, 8402},  
4559 ["sim"] = 8764,  
4560 ["thicksim"] = 8764,  
4561 ["thksim"] = 8764,  
4562 ["backsim"] = 8765,  
4563 ["bsim"] = 8765,  
4564 ["race"] = {8765, 817},  
4565 ["ac"] = 8766,  
4566 ["acE"] = {8766, 819},  
4567 ["mstpos"] = 8766,  
4568 ["acd"] = 8767,  
4569 ["VerticalTilde"] = 8768,  
4570 ["wr"] = 8768,  
4571 ["wreath"] = 8768,  
4572 ["NotTilde"] = 8769,  
4573 ["nsim"] = 8769,  
4574 ["EqualTilde"] = 8770,  
4575 ["NotEqualTilde"] = {8770, 824},  
4576 ["eqsim"] = 8770,  
4577 ["esim"] = 8770,  
4578 ["nesim"] = {8770, 824},  
4579 ["TildeEqual"] = 8771,  
4580 ["sime"] = 8771,  
4581 ["simeq"] = 8771,  
4582 ["NotTildeEqual"] = 8772,  
4583 ["nsime"] = 8772,  
4584 ["nsimeq"] = 8772,  
4585 ["TildeFullEqual"] = 8773,  
4586 ["cong"] = 8773,  
4587 ["simne"] = 8774,  
4588 ["NotTildeFullEqual"] = 8775,  
4589 ["ncong"] = 8775,  
4590 ["TildeTilde"] = 8776,  
4591 ["ap"] = 8776,  
4592 ["approx"] = 8776,  
4593 ["asymp"] = 8776,  
4594 ["thickapprox"] = 8776,  
4595 ["thkap"] = 8776,  
4596 ["NotTildeTilde"] = 8777,  
4597 ["nap"] = 8777,  
4598 ["napprox"] = 8777,  
4599 ["ape"] = 8778,  
4600 ["approxpeq"] = 8778,  
4601 ["apid"] = 8779,  
4602 ["napid"] = {8779, 824},  
4603 ["backcong"] = 8780,

4604 ["bcong"] = 8780,  
 4605 ["CupCap"] = 8781,  
 4606 ["asympeq"] = 8781,  
 4607 ["nvap"] = {8781, 8402},  
 4608 ["Bumpeq"] = 8782,  
 4609 ["HumpDownHump"] = 8782,  
 4610 ["NotHumpDownHump"] = {8782, 824},  
 4611 ["bump"] = 8782,  
 4612 ["nbump"] = {8782, 824},  
 4613 ["HumpEqual"] = 8783,  
 4614 ["NotHumpEqual"] = {8783, 824},  
 4615 ["bumpe"] = 8783,  
 4616 ["bumpeq"] = 8783,  
 4617 ["nbumpe"] = {8783, 824},  
 4618 ["DotEqual"] = 8784,  
 4619 ["doteq"] = 8784,  
 4620 ["esdot"] = 8784,  
 4621 ["nedot"] = {8784, 824},  
 4622 ["doteqdot"] = 8785,  
 4623 ["eDot"] = 8785,  
 4624 ["efDot"] = 8786,  
 4625 ["fallingdotseq"] = 8786,  
 4626 ["erDot"] = 8787,  
 4627 ["risingdotseq"] = 8787,  
 4628 ["Assign"] = 8788,  
 4629 ["colone"] = 8788,  
 4630 ["coloneq"] = 8788,  
 4631 ["ecolon"] = 8789,  
 4632 ["eqcolon"] = 8789,  
 4633 ["ecir"] = 8790,  
 4634 ["eqcirc"] = 8790,  
 4635 ["circeq"] = 8791,  
 4636 ["cire"] = 8791,  
 4637 ["wedgeq"] = 8793,  
 4638 ["veeeq"] = 8794,  
 4639 ["triangleq"] = 8796,  
 4640 ["trie"] = 8796,  
 4641 ["equest"] = 8799,  
 4642 ["questeq"] = 8799,  
 4643 ["NotEqual"] = 8800,  
 4644 ["ne"] = 8800,  
 4645 ["Congruent"] = 8801,  
 4646 ["bnequiv"] = {8801, 8421},  
 4647 ["equiv"] = 8801,  
 4648 ["NotCongruent"] = 8802,  
 4649 ["nequiv"] = 8802,  
 4650 ["le"] = 8804,

4651 ["leq"] = 8804,  
 4652 ["nvle"] = {8804, 8402},  
 4653 ["GreaterEqual"] = 8805,  
 4654 ["ge"] = 8805,  
 4655 ["geq"] = 8805,  
 4656 ["nvge"] = {8805, 8402},  
 4657 ["LessFullEqual"] = 8806,  
 4658 ["lE"] = 8806,  
 4659 ["leqq"] = 8806,  
 4660 ["nlE"] = {8806, 824},  
 4661 ["nleqq"] = {8806, 824},  
 4662 ["GreaterFullEqual"] = 8807,  
 4663 ["NotGreaterFullEqual"] = {8807, 824},  
 4664 ["gE"] = 8807,  
 4665 ["geqq"] = 8807,  
 4666 ["ngE"] = {8807, 824},  
 4667 ["ngeqq"] = {8807, 824},  
 4668 ["lnE"] = 8808,  
 4669 ["lneqq"] = 8808,  
 4670 ["lvertneqq"] = {8808, 65024},  
 4671 ["lvnE"] = {8808, 65024},  
 4672 ["gnE"] = 8809,  
 4673 ["gneqq"] = 8809,  
 4674 ["gvertneqq"] = {8809, 65024},  
 4675 ["gvnE"] = {8809, 65024},  
 4676 ["Lt"] = 8810,  
 4677 ["NestedLessLess"] = 8810,  
 4678 ["NotLessLess"] = {8810, 824},  
 4679 ["ll"] = 8810,  
 4680 ["nLt"] = {8810, 8402},  
 4681 ["nLtv"] = {8810, 824},  
 4682 ["Gt"] = 8811,  
 4683 ["NestedGreaterGreater"] = 8811,  
 4684 ["NotGreaterGreater"] = {8811, 824},  
 4685 ["gg"] = 8811,  
 4686 ["nGt"] = {8811, 8402},  
 4687 ["nGtv"] = {8811, 824},  
 4688 ["between"] = 8812,  
 4689 ["twixt"] = 8812,  
 4690 ["NotCupCap"] = 8813,  
 4691 ["NotLess"] = 8814,  
 4692 ["nless"] = 8814,  
 4693 ["nlt"] = 8814,  
 4694 ["NotGreater"] = 8815,  
 4695 ["ngt"] = 8815,  
 4696 ["ngtr"] = 8815,  
 4697 ["NotLessEqual"] = 8816,

4698 ["nle"] = 8816,  
 4699 ["nleq"] = 8816,  
 4700 ["NotGreaterEqual"] = 8817,  
 4701 ["nge"] = 8817,  
 4702 ["ngeq"] = 8817,  
 4703 ["LessTilde"] = 8818,  
 4704 ["lesssim"] = 8818,  
 4705 ["lsim"] = 8818,  
 4706 ["GreaterTilde"] = 8819,  
 4707 ["gsim"] = 8819,  
 4708 ["gtrsim"] = 8819,  
 4709 ["NotLessTilde"] = 8820,  
 4710 ["nlsim"] = 8820,  
 4711 ["NotGreaterTilde"] = 8821,  
 4712 ["ngsim"] = 8821,  
 4713 ["LessGreater"] = 8822,  
 4714 ["lessgtr"] = 8822,  
 4715 ["lg"] = 8822,  
 4716 ["GreaterLess"] = 8823,  
 4717 ["gl"] = 8823,  
 4718 ["gtrless"] = 8823,  
 4719 ["NotLessGreater"] = 8824,  
 4720 ["ntlg"] = 8824,  
 4721 ["NotGreaterLess"] = 8825,  
 4722 ["ntgl"] = 8825,  
 4723 ["Precedes"] = 8826,  
 4724 ["pr"] = 8826,  
 4725 ["prec"] = 8826,  
 4726 ["Succeeds"] = 8827,  
 4727 ["sc"] = 8827,  
 4728 ["succ"] = 8827,  
 4729 ["PrecedesSlantEqual"] = 8828,  
 4730 ["prcue"] = 8828,  
 4731 ["preccurlyeq"] = 8828,  
 4732 ["SucceedsSlantEqual"] = 8829,  
 4733 ["sccue"] = 8829,  
 4734 ["succcurlyeq"] = 8829,  
 4735 ["PrecedesTilde"] = 8830,  
 4736 ["precsim"] = 8830,  
 4737 ["prsim"] = 8830,  
 4738 ["NotSucceedsTilde"] = {8831, 824},  
 4739 ["SucceedsTilde"] = 8831,  
 4740 ["scsim"] = 8831,  
 4741 ["succsim"] = 8831,  
 4742 ["NotPrecedes"] = 8832,  
 4743 ["npr"] = 8832,  
 4744 ["nprec"] = 8832,

```

4745 ["NotSucceeds"] = 8833,
4746 ["nsc"] = 8833,
4747 ["nsucc"] = 8833,
4748 ["NotSubset"] = {8834, 8402},
4749 ["nsubset"] = {8834, 8402},
4750 ["sub"] = 8834,
4751 ["subset"] = 8834,
4752 ["vnsup"] = {8834, 8402},
4753 ["NotSuperset"] = {8835, 8402},
4754 ["Superset"] = 8835,
4755 ["nsupset"] = {8835, 8402},
4756 ["sup"] = 8835,
4757 ["supset"] = 8835,
4758 ["vnsup"] = {8835, 8402},
4759 ["nsub"] = 8836,
4760 ["nsup"] = 8837,
4761 ["SubsetEqual"] = 8838,
4762 ["sube"] = 8838,
4763 ["subseteq"] = 8838,
4764 ["SupersetEqual"] = 8839,
4765 ["supe"] = 8839,
4766 ["supseteq"] = 8839,
4767 ["NotSubsetEqual"] = 8840,
4768 ["nsube"] = 8840,
4769 ["nsubseteq"] = 8840,
4770 ["NotSupersetEqual"] = 8841,
4771 ["nsupe"] = 8841,
4772 ["nsupseteq"] = 8841,
4773 ["subne"] = 8842,
4774 ["subsetneq"] = 8842,
4775 ["varsubsetneq"] = {8842, 65024},
4776 ["vsubne"] = {8842, 65024},
4777 ["supne"] = 8843,
4778 ["supsetneq"] = 8843,
4779 ["varsupsetneq"] = {8843, 65024},
4780 ["vsupne"] = {8843, 65024},
4781 ["cupdot"] = 8845,
4782 ["UnionPlus"] = 8846,
4783 ["uplus"] = 8846,
4784 ["NotSquareSubset"] = {8847, 824},
4785 ["SquareSubset"] = 8847,
4786 ["sqsub"] = 8847,
4787 ["sqsubset"] = 8847,
4788 ["NotSquareSuperset"] = {8848, 824},
4789 ["SquareSuperset"] = 8848,
4790 ["sqsup"] = 8848,
4791 ["sqsupset"] = 8848,

```

4792 ["SquareSubsetEqual"] = 8849,  
4793 ["sqsube"] = 8849,  
4794 ["sqsubseteq"] = 8849,  
4795 ["SquareSupersetEqual"] = 8850,  
4796 ["sqsupe"] = 8850,  
4797 ["sqsupseteq"] = 8850,  
4798 ["SquareIntersection"] = 8851,  
4799 ["sqcap"] = 8851,  
4800 ["sqcaps"] = {8851, 65024},  
4801 ["SquareUnion"] = 8852,  
4802 ["sqcup"] = 8852,  
4803 ["sqcups"] = {8852, 65024},  
4804 ["CirclePlus"] = 8853,  
4805 ["oplus"] = 8853,  
4806 ["CircleMinus"] = 8854,  
4807 ["ominus"] = 8854,  
4808 ["CircleTimes"] = 8855,  
4809 ["otimes"] = 8855,  
4810 ["osol"] = 8856,  
4811 ["CircleDot"] = 8857,  
4812 ["odot"] = 8857,  
4813 ["circledcirc"] = 8858,  
4814 ["ocir"] = 8858,  
4815 ["circledast"] = 8859,  
4816 ["oast"] = 8859,  
4817 ["circleddash"] = 8861,  
4818 ["odash"] = 8861,  
4819 ["boxplus"] = 8862,  
4820 ["plusb"] = 8862,  
4821 ["boxminus"] = 8863,  
4822 ["minusb"] = 8863,  
4823 ["boxtimes"] = 8864,  
4824 ["timesb"] = 8864,  
4825 ["dotsquare"] = 8865,  
4826 ["sdotb"] = 8865,  
4827 ["RightTee"] = 8866,  
4828 ["vdash"] = 8866,  
4829 ["LeftTee"] = 8867,  
4830 ["dashv"] = 8867,  
4831 ["DownTee"] = 8868,  
4832 ["top"] = 8868,  
4833 ["UpTee"] = 8869,  
4834 ["bot"] = 8869,  
4835 ["bottom"] = 8869,  
4836 ["perp"] = 8869,  
4837 ["models"] = 8871,  
4838 ["DoubleRightTee"] = 8872,



4839 ["vDash"] = 8872,  
 4840 ["Vdash"] = 8873,  
 4841 ["Vvdash"] = 8874,  
 4842 ["VDash"] = 8875,  
 4843 ["nvdash"] = 8876,  
 4844 ["nvDash"] = 8877,  
 4845 ["nVdash"] = 8878,  
 4846 ["nVDash"] = 8879,  
 4847 ["prurel"] = 8880,  
 4848 ["LeftTriangle"] = 8882,  
 4849 ["vartriangleleft"] = 8882,  
 4850 ["vltri"] = 8882,  
 4851 ["RightTriangle"] = 8883,  
 4852 ["vartriangleright"] = 8883,  
 4853 ["vrtri"] = 8883,  
 4854 ["LeftTriangleEqual"] = 8884,  
 4855 ["ltrie"] = 8884,  
 4856 ["nvltrie"] = {8884, 8402},  
 4857 ["trianglelefteq"] = 8884,  
 4858 ["RightTriangleEqual"] = 8885,  
 4859 ["nvrtrie"] = {8885, 8402},  
 4860 ["rtrie"] = 8885,  
 4861 ["trianglerighteq"] = 8885,  
 4862 ["origof"] = 8886,  
 4863 ["imof"] = 8887,  
 4864 ["multimap"] = 8888,  
 4865 ["mumap"] = 8888,  
 4866 ["hercon"] = 8889,  
 4867 ["intcal"] = 8890,  
 4868 ["intercal"] = 8890,  
 4869 ["veebar"] = 8891,  
 4870 ["barvee"] = 8893,  
 4871 ["angrtvb"] = 8894,  
 4872 ["lrtri"] = 8895,  
 4873 ["Wedge"] = 8896,  
 4874 ["bigwedge"] = 8896,  
 4875 ["xwedge"] = 8896,  
 4876 ["Vee"] = 8897,  
 4877 ["bigvee"] = 8897,  
 4878 ["xvee"] = 8897,  
 4879 ["Intersection"] = 8898,  
 4880 ["bigcap"] = 8898,  
 4881 ["xcap"] = 8898,  
 4882 ["Union"] = 8899,  
 4883 ["bigcup"] = 8899,  
 4884 ["xcup"] = 8899,  
 4885 ["Diamond"] = 8900,

4886 ["diam"] = 8900,  
 4887 ["diamond"] = 8900,  
 4888 ["sdot"] = 8901,  
 4889 ["Star"] = 8902,  
 4890 ["sstarf"] = 8902,  
 4891 ["divideontimes"] = 8903,  
 4892 ["divonx"] = 8903,  
 4893 ["bowtie"] = 8904,  
 4894 ["ltimes"] = 8905,  
 4895 ["rtimes"] = 8906,  
 4896 ["leftthreetimes"] = 8907,  
 4897 ["lthree"] = 8907,  
 4898 ["rightthreetimes"] = 8908,  
 4899 ["rthree"] = 8908,  
 4900 ["backsimeq"] = 8909,  
 4901 ["bsime"] = 8909,  
 4902 ["curlyvee"] = 8910,  
 4903 ["cuvee"] = 8910,  
 4904 ["curlywedge"] = 8911,  
 4905 ["cuwed"] = 8911,  
 4906 ["Sub"] = 8912,  
 4907 ["Subset"] = 8912,  
 4908 ["Sup"] = 8913,  
 4909 ["Supset"] = 8913,  
 4910 ["Cap"] = 8914,  
 4911 ["Cup"] = 8915,  
 4912 ["fork"] = 8916,  
 4913 ["pitchfork"] = 8916,  
 4914 ["epar"] = 8917,  
 4915 ["lessdot"] = 8918,  
 4916 ["ltdot"] = 8918,  
 4917 ["gtdot"] = 8919,  
 4918 ["gtrdot"] = 8919,  
 4919 ["L1"] = 8920,  
 4920 ["nL1"] = {8920, 824},  
 4921 ["Gg"] = 8921,  
 4922 ["ggg"] = 8921,  
 4923 ["nGg"] = {8921, 824},  
 4924 ["LessEqualGreater"] = 8922,  
 4925 ["leg"] = 8922,  
 4926 ["lesg"] = {8922, 65024},  
 4927 ["lesseqgtr"] = 8922,  
 4928 ["GreaterEqualLess"] = 8923,  
 4929 ["gel"] = 8923,  
 4930 ["gesl"] = {8923, 65024},  
 4931 ["gtreqless"] = 8923,  
 4932 ["cuepr"] = 8926,

4933 ["curlyeqprec"] = 8926,  
 4934 ["cuesc"] = 8927,  
 4935 ["curlyeqsucc"] = 8927,  
 4936 ["NotPrecedesSlantEqual"] = 8928,  
 4937 ["nprcue"] = 8928,  
 4938 ["NotSucceedsSlantEqual"] = 8929,  
 4939 ["nsccue"] = 8929,  
 4940 ["NotSquareSubsetEqual"] = 8930,  
 4941 ["nsqsube"] = 8930,  
 4942 ["NotSquareSupersetEqual"] = 8931,  
 4943 ["nsqsupe"] = 8931,  
 4944 ["lnsim"] = 8934,  
 4945 ["gnsim"] = 8935,  
 4946 ["precnsim"] = 8936,  
 4947 ["prnsim"] = 8936,  
 4948 ["scnsim"] = 8937,  
 4949 ["succnsim"] = 8937,  
 4950 ["NotLeftTriangle"] = 8938,  
 4951 ["nltri"] = 8938,  
 4952 ["ntriangleleft"] = 8938,  
 4953 ["NotRightTriangle"] = 8939,  
 4954 ["nrtri"] = 8939,  
 4955 ["ntriangleright"] = 8939,  
 4956 ["NotLeftTriangleEqual"] = 8940,  
 4957 ["nltrie"] = 8940,  
 4958 ["ntrianglelefteq"] = 8940,  
 4959 ["NotRightTriangleEqual"] = 8941,  
 4960 ["nrtrie"] = 8941,  
 4961 ["ntrianglerighteq"] = 8941,  
 4962 ["vellip"] = 8942,  
 4963 ["ctdot"] = 8943,  
 4964 ["utdot"] = 8944,  
 4965 ["dtdot"] = 8945,  
 4966 ["disin"] = 8946,  
 4967 ["isinsv"] = 8947,  
 4968 ["isins"] = 8948,  
 4969 ["isindot"] = 8949,  
 4970 ["notin dot"] = {8949, 824},  
 4971 ["notinvc"] = 8950,  
 4972 ["notinvb"] = 8951,  
 4973 ["isinE"] = 8953,  
 4974 ["notinE"] = {8953, 824},  
 4975 ["nisd"] = 8954,  
 4976 ["xnis"] = 8955,  
 4977 ["nis"] = 8956,  
 4978 ["notnivc"] = 8957,  
 4979 ["notnivb"] = 8958,

4980 ["barwed"] = 8965,  
4981 ["barwedge"] = 8965,  
4982 ["Barwed"] = 8966,  
4983 ["doublebarwedge"] = 8966,  
4984 ["LeftCeiling"] = 8968,  
4985 ["lceil"] = 8968,  
4986 ["RightCeiling"] = 8969,  
4987 ["rceil"] = 8969,  
4988 ["LeftFloor"] = 8970,  
4989 ["lfloor"] = 8970,  
4990 ["RightFloor"] = 8971,  
4991 ["rfloor"] = 8971,  
4992 ["drcrop"] = 8972,  
4993 ["dlcrop"] = 8973,  
4994 ["urcrop"] = 8974,  
4995 ["ulcrop"] = 8975,  
4996 ["bnot"] = 8976,  
4997 ["profline"] = 8978,  
4998 ["profsurf"] = 8979,  
4999 ["telrec"] = 8981,  
5000 ["target"] = 8982,  
5001 ["ulcorn"] = 8988,  
5002 ["ulcorner"] = 8988,  
5003 ["urcorn"] = 8989,  
5004 ["urcorner"] = 8989,  
5005 ["dlcorn"] = 8990,  
5006 ["llcorner"] = 8990,  
5007 ["drcorn"] = 8991,  
5008 ["lrcorn"] = 8991,  
5009 ["frown"] = 8994,  
5010 ["sfrown"] = 8994,  
5011 ["smile"] = 8995,  
5012 ["ssmile"] = 8995,  
5013 ["cylcty"] = 9005,  
5014 ["profalar"] = 9006,  
5015 ["topbot"] = 9014,  
5016 ["ovbar"] = 9021,  
5017 ["solbar"] = 9023,  
5018 ["angzarr"] = 9084,  
5019 ["lmoust"] = 9136,  
5020 ["lmoustache"] = 9136,  
5021 ["rmoust"] = 9137,  
5022 ["rmoustache"] = 9137,  
5023 ["OverBracket"] = 9140,  
5024 ["tbrk"] = 9140,  
5025 ["UnderBracket"] = 9141,  
5026 ["bbrk"] = 9141,

5027 ["bbrktbrk"] = 9142,  
5028 ["OverParenthesis"] = 9180,  
5029 ["UnderParenthesis"] = 9181,  
5030 ["OverBrace"] = 9182,  
5031 ["UnderBrace"] = 9183,  
5032 ["trpezium"] = 9186,  
5033 ["elinters"] = 9191,  
5034 ["blank"] = 9251,  
5035 ["circledS"] = 9416,  
5036 ["oS"] = 9416,  
5037 ["HorizontalLine"] = 9472,  
5038 ["boxh"] = 9472,  
5039 ["boxv"] = 9474,  
5040 ["boxdr"] = 9484,  
5041 ["boxdl"] = 9488,  
5042 ["boxur"] = 9492,  
5043 ["boxul"] = 9496,  
5044 ["boxvr"] = 9500,  
5045 ["boxvl"] = 9508,  
5046 ["boxhd"] = 9516,  
5047 ["boxhu"] = 9524,  
5048 ["boxvh"] = 9532,  
5049 ["boxH"] = 9552,  
5050 ["boxV"] = 9553,  
5051 ["boxdR"] = 9554,  
5052 ["boxDr"] = 9555,  
5053 ["boxDR"] = 9556,  
5054 ["boxdL"] = 9557,  
5055 ["boxDL"] = 9558,  
5056 ["boxDL"] = 9559,  
5057 ["boxuR"] = 9560,  
5058 ["boxUr"] = 9561,  
5059 ["boxUR"] = 9562,  
5060 ["boxuL"] = 9563,  
5061 ["boxUL"] = 9564,  
5062 ["boxUL"] = 9565,  
5063 ["boxvR"] = 9566,  
5064 ["boxVr"] = 9567,  
5065 ["boxVR"] = 9568,  
5066 ["boxvL"] = 9569,  
5067 ["boxVl"] = 9570,  
5068 ["boxVL"] = 9571,  
5069 ["boxHd"] = 9572,  
5070 ["boxhD"] = 9573,  
5071 ["boxHD"] = 9574,  
5072 ["boxHu"] = 9575,  
5073 ["boxhU"] = 9576,

5074 ["boxHU"] = 9577,  
5075 ["boxvH"] = 9578,  
5076 ["boxVh"] = 9579,  
5077 ["boxVH"] = 9580,  
5078 ["uhblk"] = 9600,  
5079 ["lhblk"] = 9604,  
5080 ["block"] = 9608,  
5081 ["blk14"] = 9617,  
5082 ["blk12"] = 9618,  
5083 ["blk34"] = 9619,  
5084 ["Square"] = 9633,  
5085 ["squ"] = 9633,  
5086 ["square"] = 9633,  
5087 ["FilledVerySmallSquare"] = 9642,  
5088 ["blacksquare"] = 9642,  
5089 ["suarf"] = 9642,  
5090 ["squf"] = 9642,  
5091 ["EmptyVerySmallSquare"] = 9643,  
5092 ["rect"] = 9645,  
5093 ["marker"] = 9646,  
5094 ["fltns"] = 9649,  
5095 ["bigtriangleup"] = 9651,  
5096 ["xutri"] = 9651,  
5097 ["blacktriangle"] = 9652,  
5098 ["utrif"] = 9652,  
5099 ["triangle"] = 9653,  
5100 ["utri"] = 9653,  
5101 ["blacktriangleright"] = 9656,  
5102 ["rtrif"] = 9656,  
5103 ["rtri"] = 9657,  
5104 ["triangleright"] = 9657,  
5105 ["bigtriangledown"] = 9661,  
5106 ["xdtri"] = 9661,  
5107 ["blacktriangledown"] = 9662,  
5108 ["dtrif"] = 9662,  
5109 ["dtri"] = 9663,  
5110 ["triangledown"] = 9663,  
5111 ["blacktriangleleft"] = 9666,  
5112 ["ltrif"] = 9666,  
5113 ["ltri"] = 9667,  
5114 ["triangleleft"] = 9667,  
5115 ["loz"] = 9674,  
5116 ["lozenge"] = 9674,  
5117 ["cir"] = 9675,  
5118 ["tridot"] = 9708,  
5119 ["bigcirc"] = 9711,  
5120 ["xcirc"] = 9711,

5121 ["ultri"] = 9720,  
5122 ["urtri"] = 9721,  
5123 ["lltri"] = 9722,  
5124 ["EmptySmallSquare"] = 9723,  
5125 ["FilledSmallSquare"] = 9724,  
5126 ["bigstar"] = 9733,  
5127 ["starf"] = 9733,  
5128 ["star"] = 9734,  
5129 ["phone"] = 9742,  
5130 ["female"] = 9792,  
5131 ["male"] = 9794,  
5132 ["spades"] = 9824,  
5133 ["spadesuit"] = 9824,  
5134 ["clubs"] = 9827,  
5135 ["clubsuit"] = 9827,  
5136 ["hearts"] = 9829,  
5137 ["heartsuit"] = 9829,  
5138 ["diamondsuit"] = 9830,  
5139 ["diams"] = 9830,  
5140 ["sung"] = 9834,  
5141 ["flat"] = 9837,  
5142 ["natur"] = 9838,  
5143 ["natural"] = 9838,  
5144 ["sharp"] = 9839,  
5145 ["check"] = 10003,  
5146 ["checkmark"] = 10003,  
5147 ["cross"] = 10007,  
5148 ["malt"] = 10016,  
5149 ["maltese"] = 10016,  
5150 ["sext"] = 10038,  
5151 ["VerticalSeparator"] = 10072,  
5152 ["lbrk"] = 10098,  
5153 ["rbrk"] = 10099,  
5154 ["bsolhsub"] = 10184,  
5155 ["suphsol"] = 10185,  
5156 ["LeftDoubleBracket"] = 10214,  
5157 ["lbrk"] = 10214,  
5158 ["RightDoubleBracket"] = 10215,  
5159 ["robrk"] = 10215,  
5160 ["LeftAngleBracket"] = 10216,  
5161 ["lang"] = 10216,  
5162 ["langle"] = 10216,  
5163 ["RightAngleBracket"] = 10217,  
5164 ["rang"] = 10217,  
5165 ["rangle"] = 10217,  
5166 ["Lang"] = 10218,  
5167 ["Rang"] = 10219,

```

5168 ["loang"] = 10220,
5169 ["roang"] = 10221,
5170 ["LongLeftArrow"] = 10229,
5171 ["longleftarrow"] = 10229,
5172 ["xlarr"] = 10229,
5173 ["LongRightArrow"] = 10230,
5174 ["longrightarrow"] = 10230,
5175 ["xrarr"] = 10230,
5176 ["LongLeftRightArrow"] = 10231,
5177 ["longleftrightarrow"] = 10231,
5178 ["xharr"] = 10231,
5179 ["DoubleLongLeftArrow"] = 10232,
5180 ["Longleftarrow"] = 10232,
5181 ["xlArr"] = 10232,
5182 ["DoubleLongRightArrow"] = 10233,
5183 ["Longrightarrow"] = 10233,
5184 ["xrArr"] = 10233,
5185 ["DoubleLongLeftRightArrow"] = 10234,
5186 ["Longleftrightarrow"] = 10234,
5187 ["xhArr"] = 10234,
5188 ["longmapsto"] = 10236,
5189 ["xmap"] = 10236,
5190 ["dzigrarr"] = 10239,
5191 ["nvlArr"] = 10498,
5192 ["nvrArr"] = 10499,
5193 ["nvHarr"] = 10500,
5194 ["Map"] = 10501,
5195 ["lbarr"] = 10508,
5196 ["bkarow"] = 10509,
5197 ["rbarr"] = 10509,
5198 ["lBarr"] = 10510,
5199 ["dbkarow"] = 10511,
5200 ["rBarr"] = 10511,
5201 ["RBarr"] = 10512,
5202 ["drbkarow"] = 10512,
5203 ["DDottrahd"] = 10513,
5204 ["UpArrowBar"] = 10514,
5205 ["DownArrowBar"] = 10515,
5206 ["Rarrtl"] = 10518,
5207 ["latail"] = 10521,
5208 ["ratail"] = 10522,
5209 ["lAtail"] = 10523,
5210 ["rAtail"] = 10524,
5211 ["larrfs"] = 10525,
5212 ["rarrfs"] = 10526,
5213 ["larrbfs"] = 10527,
5214 ["rarrbfs"] = 10528,

```



5215 ["nwarhk"] = 10531,  
5216 ["nearhk"] = 10532,  
5217 ["hksearow"] = 10533,  
5218 ["searhk"] = 10533,  
5219 ["hkswarow"] = 10534,  
5220 ["swarhk"] = 10534,  
5221 ["nwnear"] = 10535,  
5222 ["nesear"] = 10536,  
5223 ["toea"] = 10536,  
5224 ["seswar"] = 10537,  
5225 ["tosa"] = 10537,  
5226 ["swnwar"] = 10538,  
5227 ["nrarrc"] = {10547, 824},  
5228 ["rarrc"] = 10547,  
5229 ["cudarr"] = 10549,  
5230 ["ldca"] = 10550,  
5231 ["rdca"] = 10551,  
5232 ["cudarrl"] = 10552,  
5233 ["larrpl"] = 10553,  
5234 ["curarrm"] = 10556,  
5235 ["cularrp"] = 10557,  
5236 ["rarrpl"] = 10565,  
5237 ["harrcir"] = 10568,  
5238 ["Uarrocir"] = 10569,  
5239 ["lurdshar"] = 10570,  
5240 ["ldrushar"] = 10571,  
5241 ["LeftRightVector"] = 10574,  
5242 ["RightUpDownVector"] = 10575,  
5243 ["DownLeftRightVector"] = 10576,  
5244 ["LeftUpDownVector"] = 10577,  
5245 ["LeftVectorBar"] = 10578,  
5246 ["RightVectorBar"] = 10579,  
5247 ["RightUpVectorBar"] = 10580,  
5248 ["RightDownVectorBar"] = 10581,  
5249 ["DownLeftVectorBar"] = 10582,  
5250 ["DownRightVectorBar"] = 10583,  
5251 ["LeftUpVectorBar"] = 10584,  
5252 ["LeftDownVectorBar"] = 10585,  
5253 ["LeftTeeVector"] = 10586,  
5254 ["RightTeeVector"] = 10587,  
5255 ["RightUpTeeVector"] = 10588,  
5256 ["RightDownTeeVector"] = 10589,  
5257 ["DownLeftTeeVector"] = 10590,  
5258 ["DownRightTeeVector"] = 10591,  
5259 ["LeftUpTeeVector"] = 10592,  
5260 ["LeftDownTeeVector"] = 10593,  
5261 ["lHar"] = 10594,

5262 ["uHar"] = 10595,  
5263 ["rHar"] = 10596,  
5264 ["dHar"] = 10597,  
5265 ["luruhar"] = 10598,  
5266 ["ldrdhar"] = 10599,  
5267 ["ruluhar"] = 10600,  
5268 ["rdldhar"] = 10601,  
5269 ["lharul"] = 10602,  
5270 ["llhard"] = 10603,  
5271 ["rharul"] = 10604,  
5272 ["lrhard"] = 10605,  
5273 ["UpEquilibrium"] = 10606,  
5274 ["udhar"] = 10606,  
5275 ["ReverseUpEquilibrium"] = 10607,  
5276 ["duhar"] = 10607,  
5277 ["RoundImplies"] = 10608,  
5278 ["erarr"] = 10609,  
5279 ["simrarr"] = 10610,  
5280 ["larrsim"] = 10611,  
5281 ["rarrsim"] = 10612,  
5282 ["rarrap"] = 10613,  
5283 ["ltlarr"] = 10614,  
5284 ["gtrarr"] = 10616,  
5285 ["subrarr"] = 10617,  
5286 ["suplarr"] = 10619,  
5287 ["lfisht"] = 10620,  
5288 ["rfisht"] = 10621,  
5289 ["ufisht"] = 10622,  
5290 ["dfisht"] = 10623,  
5291 ["lopar"] = 10629,  
5292 ["ropar"] = 10630,  
5293 ["lbrke"] = 10635,  
5294 ["rbrke"] = 10636,  
5295 ["lbrkslu"] = 10637,  
5296 ["rbrksld"] = 10638,  
5297 ["lbrksld"] = 10639,  
5298 ["rbrkslu"] = 10640,  
5299 ["langd"] = 10641,  
5300 ["rangd"] = 10642,  
5301 ["lparlt"] = 10643,  
5302 ["rpargt"] = 10644,  
5303 ["gtlPar"] = 10645,  
5304 ["ltrPar"] = 10646,  
5305 ["vzigzag"] = 10650,  
5306 ["vangrt"] = 10652,  
5307 ["angrtvbd"] = 10653,  
5308 ["ange"] = 10660,

5309 ["range"] = 10661,  
5310 ["dwangle"] = 10662,  
5311 ["uwangle"] = 10663,  
5312 ["angmsdaa"] = 10664,  
5313 ["angmsdab"] = 10665,  
5314 ["angmsdac"] = 10666,  
5315 ["angmsdad"] = 10667,  
5316 ["angmsdae"] = 10668,  
5317 ["angmsdaf"] = 10669,  
5318 ["angmsdag"] = 10670,  
5319 ["angmsdah"] = 10671,  
5320 ["bemptyv"] = 10672,  
5321 ["demptyv"] = 10673,  
5322 ["cemptyv"] = 10674,  
5323 ["raemptyv"] = 10675,  
5324 ["laemptyv"] = 10676,  
5325 ["ohbar"] = 10677,  
5326 ["omid"] = 10678,  
5327 ["opar"] = 10679,  
5328 ["operp"] = 10681,  
5329 ["olcross"] = 10683,  
5330 ["odsold"] = 10684,  
5331 ["olcir"] = 10686,  
5332 ["ofcir"] = 10687,  
5333 ["olt"] = 10688,  
5334 ["ogt"] = 10689,  
5335 ["cirscir"] = 10690,  
5336 ["cirE"] = 10691,  
5337 ["solb"] = 10692,  
5338 ["bsolb"] = 10693,  
5339 ["boxbox"] = 10697,  
5340 ["trisb"] = 10701,  
5341 ["rtriltri"] = 10702,  
5342 ["LeftTriangleBar"] = 10703,  
5343 ["NotLeftTriangleBar"] = {10703, 824},  
5344 ["NotRightTriangleBar"] = {10704, 824},  
5345 ["RightTriangleBar"] = 10704,  
5346 ["iinfin"] = 10716,  
5347 ["infintie"] = 10717,  
5348 ["nvinfin"] = 10718,  
5349 ["eparsl"] = 10723,  
5350 ["smeparsl"] = 10724,  
5351 ["eqvparsl"] = 10725,  
5352 ["blacklozenge"] = 10731,  
5353 ["lozf"] = 10731,  
5354 ["RuleDelayed"] = 10740,  
5355 ["dsol"] = 10742,

5356 ["bigodot"] = 10752,  
5357 ["xodot"] = 10752,  
5358 ["bigoplus"] = 10753,  
5359 ["xoplus"] = 10753,  
5360 ["bigotimes"] = 10754,  
5361 ["xotime"] = 10754,  
5362 ["biguplus"] = 10756,  
5363 ["xuplus"] = 10756,  
5364 ["bigsqcup"] = 10758,  
5365 ["xsqlcup"] = 10758,  
5366 ["iiiint"] = 10764,  
5367 ["qint"] = 10764,  
5368 ["fpartint"] = 10765,  
5369 ["cirfnint"] = 10768,  
5370 ["awint"] = 10769,  
5371 ["rppolint"] = 10770,  
5372 ["scpolint"] = 10771,  
5373 ["npolint"] = 10772,  
5374 ["pointint"] = 10773,  
5375 ["quatint"] = 10774,  
5376 ["intlarhk"] = 10775,  
5377 ["pluscir"] = 10786,  
5378 ["plusacir"] = 10787,  
5379 ["simplus"] = 10788,  
5380 ["plusdu"] = 10789,  
5381 ["plussim"] = 10790,  
5382 ["plustwo"] = 10791,  
5383 ["mcomma"] = 10793,  
5384 ["minusdu"] = 10794,  
5385 ["loplus"] = 10797,  
5386 ["roplus"] = 10798,  
5387 ["Cross"] = 10799,  
5388 ["timesd"] = 10800,  
5389 ["timesbar"] = 10801,  
5390 ["smashp"] = 10803,  
5391 ["lotimes"] = 10804,  
5392 ["rotimes"] = 10805,  
5393 ["otimesas"] = 10806,  
5394 ["Otimes"] = 10807,  
5395 ["odiv"] = 10808,  
5396 ["triplus"] = 10809,  
5397 ["triminus"] = 10810,  
5398 ["tritime"] = 10811,  
5399 ["intprod"] = 10812,  
5400 ["iproduct"] = 10812,  
5401 ["amalg"] = 10815,  
5402 ["capdot"] = 10816,

5403 ["ncup"] = 10818,  
5404 ["ncap"] = 10819,  
5405 ["capand"] = 10820,  
5406 ["cupor"] = 10821,  
5407 ["cupcap"] = 10822,  
5408 ["capcup"] = 10823,  
5409 ["cupbrcap"] = 10824,  
5410 ["capbrcup"] = 10825,  
5411 ["cupcup"] = 10826,  
5412 ["capcap"] = 10827,  
5413 ["ccups"] = 10828,  
5414 ["ccaps"] = 10829,  
5415 ["ccupssm"] = 10832,  
5416 ["And"] = 10835,  
5417 ["Or"] = 10836,  
5418 ["andand"] = 10837,  
5419 ["oror"] = 10838,  
5420 ["orslope"] = 10839,  
5421 ["andslope"] = 10840,  
5422 ["andv"] = 10842,  
5423 ["orv"] = 10843,  
5424 ["andd"] = 10844,  
5425 ["ord"] = 10845,  
5426 ["wedbar"] = 10847,  
5427 ["sdote"] = 10854,  
5428 ["simdot"] = 10858,  
5429 ["congdote"] = 10861,  
5430 ["ncongdote"] = {10861, 824},  
5431 ["easter"] = 10862,  
5432 ["apacir"] = 10863,  
5433 ["apE"] = 10864,  
5434 ["napE"] = {10864, 824},  
5435 ["eplus"] = 10865,  
5436 ["pluse"] = 10866,  
5437 ["Esim"] = 10867,  
5438 ["Colone"] = 10868,  
5439 ["Equal"] = 10869,  
5440 ["ddotseq"] = 10871,  
5441 ["eDDot"] = 10871,  
5442 ["equivDD"] = 10872,  
5443 ["ltcir"] = 10873,  
5444 ["gtcir"] = 10874,  
5445 ["ltquest"] = 10875,  
5446 ["gtquest"] = 10876,  
5447 ["LessSlantEqual"] = 10877,  
5448 ["NotLessSlantEqual"] = {10877, 824},  
5449 ["leqslant"] = 10877,

5450 ["les"] = 10877,  
5451 ["nleqslant"] = {10877, 824},  
5452 ["nles"] = {10877, 824},  
5453 ["GreaterSlantEqual"] = 10878,  
5454 ["NotGreaterSlantEqual"] = {10878, 824},  
5455 ["geqslant"] = 10878,  
5456 ["ges"] = 10878,  
5457 ["ngeqslant"] = {10878, 824},  
5458 ["nges"] = {10878, 824},  
5459 ["lesdot"] = 10879,  
5460 ["gesdot"] = 10880,  
5461 ["lesdoto"] = 10881,  
5462 ["gesdoto"] = 10882,  
5463 ["lesdotor"] = 10883,  
5464 ["gesdoto1"] = 10884,  
5465 ["lap"] = 10885,  
5466 ["lessapprox"] = 10885,  
5467 ["gap"] = 10886,  
5468 ["gtrapprox"] = 10886,  
5469 ["lne"] = 10887,  
5470 ["lneq"] = 10887,  
5471 ["gne"] = 10888,  
5472 ["gneq"] = 10888,  
5473 ["lnap"] = 10889,  
5474 ["lnapprox"] = 10889,  
5475 ["gnap"] = 10890,  
5476 ["gnapprox"] = 10890,  
5477 ["lEg"] = 10891,  
5478 ["lesseqqgtr"] = 10891,  
5479 ["gEl"] = 10892,  
5480 ["gtreqqless"] = 10892,  
5481 ["lsime"] = 10893,  
5482 ["gsime"] = 10894,  
5483 ["lsimg"] = 10895,  
5484 ["gsiml"] = 10896,  
5485 ["lgE"] = 10897,  
5486 ["glE"] = 10898,  
5487 ["lesges"] = 10899,  
5488 ["gesles"] = 10900,  
5489 ["els"] = 10901,  
5490 ["eqslantless"] = 10901,  
5491 ["egs"] = 10902,  
5492 ["eqslantgtr"] = 10902,  
5493 ["elsdot"] = 10903,  
5494 ["egsdot"] = 10904,  
5495 ["el"] = 10905,  
5496 ["eg"] = 10906,

5497 ["siml"] = 10909,  
5498 ["simg"] = 10910,  
5499 ["simlE"] = 10911,  
5500 ["simgE"] = 10912,  
5501 ["LessLess"] = 10913,  
5502 ["NotNestedLessLess"] = {10913, 824},  
5503 ["GreaterGreater"] = 10914,  
5504 ["NotNestedGreaterGreater"] = {10914, 824},  
5505 ["glj"] = 10916,  
5506 ["gla"] = 10917,  
5507 ["ltcc"] = 10918,  
5508 ["gtcc"] = 10919,  
5509 ["lescc"] = 10920,  
5510 ["gescc"] = 10921,  
5511 ["smt"] = 10922,  
5512 ["lat"] = 10923,  
5513 ["smte"] = 10924,  
5514 ["smtes"] = {10924, 65024},  
5515 ["late"] = 10925,  
5516 ["lates"] = {10925, 65024},  
5517 ["bumpE"] = 10926,  
5518 ["NotPrecedesEqual"] = {10927, 824},  
5519 ["PrecedesEqual"] = 10927,  
5520 ["npre"] = {10927, 824},  
5521 ["npreceq"] = {10927, 824},  
5522 ["pre"] = 10927,  
5523 ["preceq"] = 10927,  
5524 ["NotSucceedsEqual"] = {10928, 824},  
5525 ["SucceedsEqual"] = 10928,  
5526 ["nsce"] = {10928, 824},  
5527 ["nsucceq"] = {10928, 824},  
5528 ["sce"] = 10928,  
5529 ["succeq"] = 10928,  
5530 ["prE"] = 10931,  
5531 ["scE"] = 10932,  
5532 ["precneqq"] = 10933,  
5533 ["prnE"] = 10933,  
5534 ["scnE"] = 10934,  
5535 ["succneqq"] = 10934,  
5536 ["prap"] = 10935,  
5537 ["precapprox"] = 10935,  
5538 ["scap"] = 10936,  
5539 ["succapprox"] = 10936,  
5540 ["precnapprox"] = 10937,  
5541 ["prnap"] = 10937,  
5542 ["scnap"] = 10938,  
5543 ["succnapprox"] = 10938,

5544 ["Pr"] = 10939,  
5545 ["Sc"] = 10940,  
5546 ["subdot"] = 10941,  
5547 ["supdot"] = 10942,  
5548 ["subplus"] = 10943,  
5549 ["supplus"] = 10944,  
5550 ["submult"] = 10945,  
5551 ["supmult"] = 10946,  
5552 ["subedot"] = 10947,  
5553 ["supedot"] = 10948,  
5554 ["nsubE"] = {10949, 824},  
5555 ["nsubseteqq"] = {10949, 824},  
5556 ["subE"] = 10949,  
5557 ["subseteqq"] = 10949,  
5558 ["nsupE"] = {10950, 824},  
5559 ["nsupseteqq"] = {10950, 824},  
5560 ["supE"] = 10950,  
5561 ["supseteqq"] = 10950,  
5562 ["subsim"] = 10951,  
5563 ["supsim"] = 10952,  
5564 ["subnE"] = 10955,  
5565 ["subsetneqq"] = 10955,  
5566 ["varsubsetneqq"] = {10955, 65024},  
5567 ["vsubnE"] = {10955, 65024},  
5568 ["supnE"] = 10956,  
5569 ["supsetneqq"] = 10956,  
5570 ["varsupsetneqq"] = {10956, 65024},  
5571 ["vsupnE"] = {10956, 65024},  
5572 ["csub"] = 10959,  
5573 ["csup"] = 10960,  
5574 ["csube"] = 10961,  
5575 ["csupe"] = 10962,  
5576 ["subsup"] = 10963,  
5577 ["supsup"] = 10964,  
5578 ["subsub"] = 10965,  
5579 ["supsup"] = 10966,  
5580 ["suphsub"] = 10967,  
5581 ["supdsub"] = 10968,  
5582 ["forkv"] = 10969,  
5583 ["topfork"] = 10970,  
5584 ["mlcp"] = 10971,  
5585 ["Dashv"] = 10980,  
5586 ["DoubleLeftTee"] = 10980,  
5587 ["Vdashl"] = 10982,  
5588 ["Barv"] = 10983,  
5589 ["vBar"] = 10984,  
5590 ["vBarv"] = 10985,



5591 ["Vbar"] = 10987,  
5592 ["Not"] = 10988,  
5593 ["bNot"] = 10989,  
5594 ["rnmid"] = 10990,  
5595 ["cirmid"] = 10991,  
5596 ["midcir"] = 10992,  
5597 ["topcir"] = 10993,  
5598 ["nhpar"] = 10994,  
5599 ["parsim"] = 10995,  
5600 ["nparsl"] = {11005, 8421},  
5601 ["parsl"] = 11005,  
5602 ["fflig"] = 64256,  
5603 ["filig"] = 64257,  
5604 ["fllig"] = 64258,  
5605 ["ffilig"] = 64259,  
5606 ["ffllig"] = 64260,  
5607 ["Ascr"] = 119964,  
5608 ["Cscr"] = 119966,  
5609 ["Dscr"] = 119967,  
5610 ["Gscr"] = 119970,  
5611 ["Jscr"] = 119973,  
5612 ["Kscr"] = 119974,  
5613 ["Nscr"] = 119977,  
5614 ["Oscr"] = 119978,  
5615 ["Pscr"] = 119979,  
5616 ["Qscr"] = 119980,  
5617 ["Sscr"] = 119982,  
5618 ["Tscr"] = 119983,  
5619 ["Uscr"] = 119984,  
5620 ["Vscr"] = 119985,  
5621 ["Wscr"] = 119986,  
5622 ["Xscr"] = 119987,  
5623 ["Yscr"] = 119988,  
5624 ["Zscr"] = 119989,  
5625 ["ascr"] = 119990,  
5626 ["bscr"] = 119991,  
5627 ["cscr"] = 119992,  
5628 ["dscr"] = 119993,  
5629 ["fscr"] = 119995,  
5630 ["hscr"] = 119997,  
5631 ["iscr"] = 119998,  
5632 ["jscr"] = 119999,  
5633 ["kscr"] = 120000,  
5634 ["lscr"] = 120001,  
5635 ["mscr"] = 120002,  
5636 ["nscr"] = 120003,  
5637 ["pscr"] = 120005,

5638 ["qscr"] = 120006,  
5639 ["rscr"] = 120007,  
5640 ["sscr"] = 120008,  
5641 ["tscr"] = 120009,  
5642 ["uscr"] = 120010,  
5643 ["vscr"] = 120011,  
5644 ["wscr"] = 120012,  
5645 ["xscr"] = 120013,  
5646 ["yscr"] = 120014,  
5647 ["zscr"] = 120015,  
5648 ["Afr"] = 120068,  
5649 ["Bfr"] = 120069,  
5650 ["Dfr"] = 120071,  
5651 ["Efr"] = 120072,  
5652 ["Ffr"] = 120073,  
5653 ["Gfr"] = 120074,  
5654 ["Jfr"] = 120077,  
5655 ["Kfr"] = 120078,  
5656 ["Lfr"] = 120079,  
5657 ["Mfr"] = 120080,  
5658 ["Nfr"] = 120081,  
5659 ["Ofr"] = 120082,  
5660 ["Pfr"] = 120083,  
5661 ["Qfr"] = 120084,  
5662 ["Sfr"] = 120086,  
5663 ["Tfr"] = 120087,  
5664 ["Ufr"] = 120088,  
5665 ["Vfr"] = 120089,  
5666 ["Wfr"] = 120090,  
5667 ["Xfr"] = 120091,  
5668 ["Yfr"] = 120092,  
5669 ["afr"] = 120094,  
5670 ["bfr"] = 120095,  
5671 ["cfr"] = 120096,  
5672 ["dfr"] = 120097,  
5673 ["efr"] = 120098,  
5674 ["ffr"] = 120099,  
5675 ["gfr"] = 120100,  
5676 ["hfr"] = 120101,  
5677 ["ifr"] = 120102,  
5678 ["jfr"] = 120103,  
5679 ["kfr"] = 120104,  
5680 ["lfr"] = 120105,  
5681 ["mfr"] = 120106,  
5682 ["nfr"] = 120107,  
5683 ["ofr"] = 120108,  
5684 ["pfr"] = 120109,

5685 ["qfr"] = 120110,  
5686 ["rfr"] = 120111,  
5687 ["sfr"] = 120112,  
5688 ["tfr"] = 120113,  
5689 ["ufr"] = 120114,  
5690 ["vfr"] = 120115,  
5691 ["wfr"] = 120116,  
5692 ["xfr"] = 120117,  
5693 ["yfr"] = 120118,  
5694 ["zfr"] = 120119,  
5695 ["Aopf"] = 120120,  
5696 ["Bopf"] = 120121,  
5697 ["Dopf"] = 120123,  
5698 ["Eopf"] = 120124,  
5699 ["Fopf"] = 120125,  
5700 ["Gopf"] = 120126,  
5701 ["Iopf"] = 120128,  
5702 ["Jopf"] = 120129,  
5703 ["Kopf"] = 120130,  
5704 ["Lopf"] = 120131,  
5705 ["Mopf"] = 120132,  
5706 ["Oopf"] = 120134,  
5707 ["Sopf"] = 120138,  
5708 ["Topf"] = 120139,  
5709 ["Uopf"] = 120140,  
5710 ["Vopf"] = 120141,  
5711 ["Wopf"] = 120142,  
5712 ["Xopf"] = 120143,  
5713 ["Yopf"] = 120144,  
5714 ["aopf"] = 120146,  
5715 ["bopf"] = 120147,  
5716 ["copf"] = 120148,  
5717 ["dopf"] = 120149,  
5718 ["eopf"] = 120150,  
5719 ["fopf"] = 120151,  
5720 ["gopf"] = 120152,  
5721 ["hopf"] = 120153,  
5722 ["iopf"] = 120154,  
5723 ["jopf"] = 120155,  
5724 ["kopf"] = 120156,  
5725 ["lopf"] = 120157,  
5726 ["mopf"] = 120158,  
5727 ["nopf"] = 120159,  
5728 ["oopf"] = 120160,  
5729 ["popf"] = 120161,  
5730 ["qopf"] = 120162,  
5731 ["ropf"] = 120163,

```

5732 ["sopf"] = 120164,
5733 ["topf"] = 120165,
5734 ["uopf"] = 120166,
5735 ["vopf"] = 120167,
5736 ["wopf"] = 120168,
5737 ["xopf"] = 120169,
5738 ["yopf"] = 120170,
5739 ["zopf"] = 120171,
5740 }

```

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

```

5741 function entities.dec_entity(s)
5742   local n = tonumber(s)
5743   if n == nil then
5744     return "&#" .. s .. ";" -- fallback for unknown entities
5745   end
5746   return unicode.utf8.char(n)
5747 end

```

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

```

5748 function entities.hex_entity(s)
5749   local n = tonumber("0x"..s)
5750   if n == nil then
5751     return "&#x" .. s .. ";" -- fallback for unknown entities
5752   end
5753   return unicode.utf8.char(n)
5754 end

```

Given a captured character `x` and a string `s` of hexadecimal digits, the `entities.hex_entity_with_x_char` returns the corresponding UTF8-encoded Unicode codepoint or fallback with the `x` character.

```

5755 function entities.hex_entity_with_x_char(x, s)
5756   local n = tonumber("0x"..s)
5757   if n == nil then
5758     return "&#" .. x .. s .. ";" -- fallback for unknown entities
5759   end
5760   return unicode.utf8.char(n)
5761 end

```

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

```

5762 function entities.char_entity(s)
5763   local code_points = character_entities[s]
5764   if code_points == nil then
5765     return "&" .. s .. ";"
5766   end

```

```

5767 if type(code_points) ~= 'table' then
5768     code_points = {code_points}
5769 end
5770 local char_table = {}
5771 for _, code_point in ipairs(code_points) do
5772     table.insert(char_table, unicode.utf8.char(code_point))
5773 end
5774 return table.concat(char_table)
5775 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.

```

5776 M.writer = {}

```

The `writer.new` method creates and returns a new TeX writer object associated with the Lua interface options (see Section 2.1.3) `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.

```

5777 function M.writer.new(options)
5778     local self = {}

```

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

```

5779     self.options = options

```

Define `writer->flatten_inlines`, which indicates whether or not the writer should produce raw text rather than text in the output format for inline elements. The `writer->flatten_inlines` member variable is mutable.

```

5780     self.flatten_inlines = false

```

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

```

5781     local slice_specifiers = {}
5782     for specifier in options.slice:gmatch("[^%s]+") do

```

```

5783     table.insert(slice_specifiers, specifier)
5784 end
5785
5786 if #slice_specifiers == 2 then
5787     self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
5788     local slice_begin_type = self.slice_begin:sub(1, 1)
5789     if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
5790         self.slice_begin = "^" .. self.slice_begin
5791     end
5792     local slice_end_type = self.slice_end:sub(1, 1)
5793     if slice_end_type ~= "^" and slice_end_type ~= "$" then
5794         self.slice_end = "$" .. self.slice_end
5795     end
5796 elseif #slice_specifiers == 1 then
5797     self.slice_begin = "^" .. slice_specifiers[1]
5798     self.slice_end = "$" .. slice_specifiers[1]
5799 end
5800
5801 self.slice_begin_type = self.slice_begin:sub(1, 1)
5802 self.slice_begin_identifier = self.slice_begin:sub(2) or ""
5803 self.slice_end_type = self.slice_end:sub(1, 1)
5804 self.slice_end_identifier = self.slice_end:sub(2) or ""
5805
5806 if self.slice_begin == "^" and self.slice_end ~= "^" then
5807     self.is_writing = true
5808 else
5809     self.is_writing = false
5810 end

```

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

```
5811 self.space = " "
```

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

```
5812 self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
5813 function self.plain(s)
5814     return s
5815 end

```

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

```
5816 function self.paragraph(s)
5817     if not self.is_writing then return "" end
5818     return s
5819 end

```

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

```

5820 self.interblocksep_text = "\\markdownRendererInterblockSeparator\n{}"
5821 function self.interblocksep()
5822     if not self.is_writing then return "" end
5823     return self.interblocksep_text
5824 end

```

Define `writer->paragraphsep` as the output format of a paragraph separator. Users can use more than one blank line to delimit two blocks to indicate the end of a series of blocks that make up a paragraph. This produces a paragraph separator instead of an interblock separator.

```

5825 self.paragraphsep_text = "\\markdownRendererParagraphSeparator\n{}"
5826 function self.paragraphsep()
5827     if not self.is_writing then return "" end
5828     return self.paragraphsep_text
5829 end

```

Define `writer->undosep` as a function that will remove the output produced by an immediately preceding block element / paragraph separator.

```

5830 self.undosep_text = "\\markdownRendererUndoSeparator\n{}"
5831 function self.undosep()
5832     if not self.is_writing then return "" end
5833     return self.undosep_text
5834 end

```

Define `writer->soft_line_break` as the output format of a soft line break.

```

5835 self.soft_line_break = function()
5836     if self.flatten_inlines then return "\n" end
5837     return "\\markdownRendererSoftLineBreak\n{}"
5838 end

```

Define `writer->hard_line_break` as the output format of a hard line break.

```

5839 self.hard_line_break = function()
5840     if self.flatten_inlines then return "\n" end
5841     return "\\markdownRendererHardLineBreak\n{}"
5842 end

```

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

```

5843 self.ellipsis = "\\markdownRendererEllipsis{}"

```

Define `writer->thematic_break` as the output format of a thematic break.

```

5844 function self.thematic_break()
5845     if not self.is_writing then return "" end
5846     return "\\markdownRendererThematicBreak{}"
5847 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.

```

5848 self.escaped_uri_chars = {

```

```

5849     [{""] = "\\markdownRendererLeftBrace{}",
5850     ["}"] = "\\markdownRendererRightBrace{}",
5851     [{"\\"} = "\\markdownRendererBackslash{}",
5852     [{"r"} = " ",
5853     [{"n"} = " ",
5854   }
5855   self.escaped_minimal_strings = {
5856     [{"^"} = "\\markdownRendererCircumflex",
5857     .. "\\markdownRendererCircumflex ",
5858     [{"☒"} = "\\markdownRendererTickedBox{}",
5859     [{"◻"} = "\\markdownRendererHalfTickedBox{}",
5860     [{"□"} = "\\markdownRendererUntickedBox{}",
5861     [entities.hex_entity('FFFD')]
5862     = "\\markdownRendererReplacementCharacter{}",
5863   }

```

Define table `writer->escaped_strings` containing the mapping from character strings that need to be escaped in typeset content.

```

5864   self.escaped_strings = util.table_copy(self.escaped_minimal_strings)
5865   self.escaped_strings[entities.hex_entity('00A0')] = self.nbsp

```

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

```

5866   self.escaped_chars = {
5867     [{""] = "\\markdownRendererLeftBrace{}",
5868     ["}"] = "\\markdownRendererRightBrace{}",
5869     [{"%"} = "\\markdownRendererPercentSign{}",
5870     [{"\\"} = "\\markdownRendererBackslash{}",
5871     [{"#"} = "\\markdownRendererHash{}",
5872     [{"$"} = "\\markdownRendererDollarSign{}",
5873     [{"&"} = "\\markdownRendererAmpersand{}",
5874     [{"_"} = "\\markdownRendererUnderscore{}",
5875     [{"^"} = "\\markdownRendererCircumflex{}",
5876     [{"~"} = "\\markdownRendererTilde{}",
5877     [{"|"} = "\\markdownRendererPipe{}",
5878     [entities.hex_entity('0000')]
5879     = "\\markdownRendererReplacementCharacter{}",
5880   }

```

Use the `writer->escaped_chars`, `writer->escaped_uri_chars`, and `writer->escaped_minimal` tables to create the `escape_typographic_text`, `escape_programmatic_text`, and `escape_minimal` local escaper functions.

```

5881   local function create_escaper(char_escapes, string_escapes)
5882     local escape = util.escaper(char_escapes, string_escapes)
5883     return function(s)
5884       if self.flatten_inlines then return s end
5885       return escape(s)

```



```

5886     end
5887 end
5888 local escape_typographic_text = create_escaper(
5889     self.escaped_chars, self.escaped_strings)
5890 local escape_programmatic_text = create_escaper(
5891     self.escaped_uri_chars, self.escaped_minimal_strings)
5892 local escape_minimal = create_escaper(
5893     {}, self.escaped_minimal_strings)

```

Define the following semantic aliases for the escaper functions:

- `writer->escape` transforms a text string that should always be made printable.
- `writer->string` transforms a text string that should be made printable only when the `hybrid` Lua option is disabled. When `hybrid` is enabled, the text string should be kept as-is.
- `writer->math` transforms a math span.
- `writer->identifier` transforms an input programmatic identifier.
- `writer->uri` transforms an input URI.
- `writer->infostring` transforms a fence code infostring.

```

5894 self.escape = escape_typographic_text
5895 self.math = escape_minimal
5896 if options.hybrid then
5897     self.identifier = escape_minimal
5898     self.string = escape_minimal
5899     self.uri = escape_minimal
5900     self.infostring = escape_minimal
5901 else
5902     self.identifier = escape_programmatic_text
5903     self.string = escape_typographic_text
5904     self.uri = escape_programmatic_text
5905     self.infostring = escape_programmatic_text
5906 end

```

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

```

5907 function self.warning(t)
5908     return {"\\markdownRendererWarning{" , self.identifier(t), "}"}
5909 end

```

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

```

5910 function self.error(t)
5911     return {"\\markdownRendererError{" , self.identifier(t), "}"}
5912 end

```

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

```

5913 function self.code(s, attributes)
5914   if self.flatten_inlines then return s end
5915   local buf = {}
5916   if attributes ~= nil then
5917     table.insert(buf,
5918       "\\markdownRendererCodeSpanAttributeContextBegin\n")
5919     table.insert(buf, self.attributes(attributes))
5920   end
5921   table.insert(buf,
5922     {"\\markdownRendererCodeSpan{" , self.escape(s), "}"})
5923   if attributes ~= nil then
5924     table.insert(buf,
5925       "\\markdownRendererCodeSpanAttributeContextEnd{")
5926   end
5927   return buf
5928 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, `tit` to the title of the link, and `attributes` to optional attributes.

```

5929 function self.link(lab, src, tit, attributes)
5930   if self.flatten_inlines then return lab end
5931   local buf = {}
5932   if attributes ~= nil then
5933     table.insert(buf,
5934       "\\markdownRendererLinkAttributeContextBegin\n")
5935     table.insert(buf, self.attributes(attributes))
5936   end
5937   table.insert(buf, {"\\markdownRendererLink{" ,lab,"} ,
5938     {"",self.escape(src),"} ,
5939     {"",self.uri(src),"} ,
5940     {"",self.string(tit or ""),"}"}))
5941   if attributes ~= nil then
5942     table.insert(buf,
5943       "\\markdownRendererLinkAttributeContextEnd{")
5944   end
5945   return buf
5946 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, `tit` to the title of the image, and `attributes` to optional attributes.

```

5947 function self.image(lab, src, tit, attributes)
5948   if self.flatten_inlines then return lab end
5949   local buf = {}
5950   if attributes ~= nil then
5951     table.insert(buf,

```

```

5952             "\\markdownRendererImageAttributeContextBegin\n")
5953     table.insert(buf, self.attributes(attributes))
5954     end
5955     table.insert(buf, {"\\markdownRendererImage{",lab,"}",
5956                     "{",self.string(src),"}",
5957                     "{",self.uri(src),"}",
5958                     "{",self.string(tit or ""),"}}"})
5959     if attributes ~= nil then
5960         table.insert(buf,
5961             "\\markdownRendererImageAttributeContextEnd{")
5962     end
5963     return buf
5964 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.

```

5965 function self.bulletlist(items,tight)
5966     if not self.is_writing then return "" end
5967     local buffer = {}
5968     for _,item in ipairs(items) do
5969         if item ~= "" then
5970             buffer[#buffer + 1] = self.bulletitem(item)
5971         end
5972     end
5973     local contents = util.intersperse(buffer,"\n")
5974     if tight and options.tightLists then
5975         return {"\\markdownRendererUlBeginTight\n",contents,
5976             "\n\\markdownRendererUlEndTight "}
5977     else
5978         return {"\\markdownRendererUlBegin\n",contents,
5979             "\n\\markdownRendererUlEnd "}
5980     end
5981 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.

```

5982 function self.bulletitem(s)
5983     return {"\\markdownRendererUlItem ",s,
5984         "\\markdownRendererUlItemEnd "}
5985 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.

```

5986 function self.orderedlist(items,tight,startnum)
5987     if not self.is_writing then return "" end

```

```

5988     local buffer = {}
5989     local num = startnum
5990     for _,item in ipairs(items) do
5991         if item ~= "" then
5992             buffer[#buffer + 1] = self.ordereditem(item,num)
5993         end
5994         if num ~= nil and item ~= "" then
5995             num = num + 1
5996         end
5997     end
5998     local contents = util.intersperse(buffer,"\n")
5999     if tight and options.tightLists then
6000         return {"\markdownRenderer01BeginTight\n",contents,
6001             "\n\markdownRenderer01EndTight "}
6002     else
6003         return {"\markdownRenderer01Begin\n",contents,
6004             "\n\markdownRenderer01End "}
6005     end
6006 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.

```

6007     function self.ordereditem(s,num)
6008         if num ~= nil then
6009             return {"\markdownRenderer01ItemWithNumber{" ,num,"} ",s,
6010                 "\markdownRenderer01ItemEnd "}
6011         else
6012             return {"\markdownRenderer01Item ",s,
6013                 "\markdownRenderer01ItemEnd "}
6014         end
6015     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.

```

6016     function self.inline_html_comment(contents)
6017         if self.flatten_inlines then return contents end
6018         return {"\markdownRendererInlineHtmlComment{" ,contents,"}"}
6019     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.

```

6020     function self.inline_html_tag(contents)
6021         if self.flatten_inlines then return contents end
6022         return {"\markdownRendererInlineHtmlTag{" ,
6023             self.string(contents),"}"}

```

```
6024 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.

```
6025 function self.block_html_element(s)
6026   if not self.is_writing then return "" end
6027   local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
6028   return {"\\markdownRendererInputBlockHtmlElement{" ,name,"}"}
6029 end
```

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

```
6030 function self.emphasis(s)
6031   if self.flatten_inlines then return s end
6032   return {"\\markdownRendererEmphasis{" ,s,"}"}
6033 end
```

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

```
6034 function self.checkbox(f)
6035   if f == 1.0 then
6036     return "☒ "
6037   elseif f == 0.0 then
6038     return "☐ "
6039   else
6040     return "◻ "
6041   end
6042 end
```

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

```
6043 function self.strong(s)
6044   if self.flatten_inlines then return s end
6045   return {"\\markdownRendererStrongEmphasis{" ,s,"}"}
6046 end
```

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

```
6047 function self.blockquote(s)
6048   if not self.is_writing then return "" end
6049   return {"\\markdownRendererBlockQuoteBegin\n",s,
6050     "\\markdownRendererBlockQuoteEnd "}
6051 end
```

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

```
6052 function self.verbatim(s)
6053   if not self.is_writing then return "" end
```

```

6054     s = s:gsub("\n$", "")
6055     local name = util.cache_verbatim(options.cacheDir, s)
6056     return {"\\markdownRendererInputVerbatim{" ,name,"}"}
6057 end

```

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

```

6058 function self.document(d)
6059     local buf = {"\\markdownRendererDocumentBegin\n", d}
6060
6061     -- pop all attributes
6062     table.insert(buf, self.pop_attributes())
6063
6064     table.insert(buf, "\\markdownRendererDocumentEnd")
6065
6066     return buf
6067 end

```

Define `writer->attributes` as a function that will transform input attributes `attrs` to the output format.

```

6068 local seen_identifiers = {}
6069 local key_value_regex = "([~= ]+)%s*=%s*(.*)"
6070 local function normalize_attributes(attributes, auto_identifiers)
6071     -- normalize attributes
6072     local normalized_attributes = {}
6073     local has_explicit_identifiers = false
6074     local key, value
6075     for _, attribute in ipairs(attributes or {}) do
6076         if attribute:sub(1, 1) == "#" then
6077             table.insert(normalized_attributes, attribute)
6078             has_explicit_identifiers = true
6079             seen_identifiers[attribute:sub(2)] = true
6080         elseif attribute:sub(1, 1) == "." then
6081             table.insert(normalized_attributes, attribute)
6082         else
6083             key, value = attribute:match(key_value_regex)
6084             if key:lower() == "id" then
6085                 table.insert(normalized_attributes, "#" .. value)
6086             elseif key:lower() == "class" then
6087                 local classes = {}
6088                 for class in value:gmatch("%S+") do
6089                     table.insert(classes, class)
6090                 end
6091                 table.sort(classes)
6092                 for _, class in ipairs(classes) do
6093                     table.insert(normalized_attributes, "." .. class)
6094                 end
6095             else

```

```

6096         table.insert(normalized_attributes, attribute)
6097     end
6098     end
6099 end
6100
6101 -- if no explicit identifiers exist, add auto identifiers
6102 if not has_explicit_identifiers and auto_identifiers ~= nil then
6103     local seen_auto_identifiers = {}
6104     for _, auto_identifier in ipairs(auto_identifiers) do
6105         if seen_auto_identifiers[auto_identifier] == nil then
6106             seen_auto_identifiers[auto_identifier] = true
6107             if seen_identifiers[auto_identifier] == nil then
6108                 seen_identifiers[auto_identifier] = true
6109                 table.insert(normalized_attributes,
6110                     "#" .. auto_identifier)
6111             else
6112                 local auto_identifier_number = 1
6113                 while true do
6114                     local numbered_auto_identifier = auto_identifier .. "-"
6115                                             .. auto_identifier_number
6116                     if seen_identifiers[numbered_auto_identifier] == nil then
6117                         seen_identifiers[numbered_auto_identifier] = true
6118                         table.insert(normalized_attributes,
6119                             "#" .. numbered_auto_identifier)
6120                     break
6121                 end
6122                 auto_identifier_number = auto_identifier_number + 1
6123             end
6124         end
6125     end
6126     end
6127 end
6128
6129 -- sort and deduplicate normalized attributes
6130 table.sort(normalized_attributes)
6131 local seen_normalized_attributes = {}
6132 local deduplicated_normalized_attributes = {}
6133 for _, attribute in ipairs(normalized_attributes) do
6134     if seen_normalized_attributes[attribute] == nil then
6135         seen_normalized_attributes[attribute] = true
6136         table.insert(deduplicated_normalized_attributes, attribute)
6137     end
6138 end
6139
6140 return deduplicated_normalized_attributes
6141 end
6142

```

```

6143 function self.attributes(attributes, should_normalize_attributes)
6144   local normalized_attributes
6145   if should_normalize_attributes == false then
6146     normalized_attributes = attributes
6147   else
6148     normalized_attributes = normalize_attributes(attributes)
6149   end
6150
6151   local buf = {}
6152   local key, value
6153   for _, attribute in ipairs(normalized_attributes) do
6154     if attribute:sub(1, 1) == "#" then
6155       table.insert(buf, {"\\markdownRendererAttributeIdentifier{" ,
6156         attribute:sub(2), "}"}))
6157     elseif attribute:sub(1, 1) == "." then
6158       table.insert(buf, {"\\markdownRendererAttributeName{" ,
6159         attribute:sub(2), "}"}))
6160     else
6161       key, value = attribute:match(key_value_regex)
6162       table.insert(buf, {"\\markdownRendererAttributeKeyValue{" ,
6163         key, "}{" , value, "}"}))
6164     end
6165   end
6166
6167   return buf
6168 end

```

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

```

6169 self.active_attributes = {}

```

Define `writer->attribute_type_levels` as a hash table that maps attribute types to the number of attributes of said type in `writer->active_attributes`.

```

6170 self.attribute_type_levels = {}
6171 setmetatable(self.attribute_type_levels,
6172   { __index = function() return 0 end })

```

Define `writer->push_attributes` and `writer->pop_attributes` as functions that will add a new set of active block-level attributes or remove the most current attributes from `writer->active_attributes`.

```

6173 local function apply_attributes()
6174   local buf = {}
6175   for i = 1, #self.active_attributes do
6176     local start_output = self.active_attributes[i][3]
6177     if start_output ~= nil then
6178       table.insert(buf, start_output)
6179     end
6180   end

```



```

6181     return buf
6182 end
6183
6184 local function tear_down_attributes()
6185     local buf = {}
6186     for i = #self.active_attributes, 1, -1 do
6187         local end_output = self.active_attributes[i][4]
6188         if end_output ~= nil then
6189             table.insert(buf, end_output)
6190         end
6191     end
6192     return buf
6193 end

```

The `writer->push_attributes` method adds `attributes` of type `attribute_type` to `writer->active_attributes`. The `start_output` string is used to construct a rope that will be returned by this function, together with output produced as a result of slicing (see `slice`). The `end_output` string is stored together with `attributes` and is used to construct the return value of the `writer->pop_attributes` method.

```

6194 function self.push_attributes(attribute_type, attributes,
6195                             start_output, end_output)
6196     local attribute_type_level
6197         = self.attribute_type_levels[attribute_type]
6198     self.attribute_type_levels[attribute_type]
6199         = attribute_type_level + 1
6200
6201     -- index attributes in a hash table for easy lookup
6202     attributes = attributes or {}
6203     for i = 1, #attributes do
6204         attributes[attributes[i]] = true
6205     end
6206
6207     local buf = {}
6208     -- handle slicing
6209     if attributes["#" .. self.slice_end_identifiser] ~= nil and
6210        self.slice_end_type == "^" then
6211         if self.is_writing then
6212             table.insert(buf, self.undosep())
6213             table.insert(buf, tear_down_attributes())
6214         end
6215         self.is_writing = false
6216     end
6217     if attributes["#" .. self.slice_begin_identifiser] ~= nil and
6218        self.slice_begin_type == "^" then
6219         table.insert(buf, apply_attributes())
6220         self.is_writing = true
6221     end

```

```

6222     if self.is_writing and start_output ~= nil then
6223         table.insert(buf, start_output)
6224     end
6225     table.insert(self.active_attributes,
6226                 {attribute_type, attributes,
6227                 start_output, end_output})
6228     return buf
6229 end
6230

```

The `writer->pop_attributes` method removes the most current of active block-level attributes from `writer->active_attributes` until attributes of type `attribute_type` have been removed. The method returns a rope constructed from the `end_output` string specified in the calls of `writer->push_attributes` that produced the most current attributes, and also from output produced as a result of slicing (see `slice`).

```

6231 function self.pop_attributes(attribute_type)
6232     local buf = {}
6233     -- pop attributes until we find attributes of correct type
6234     -- or until no attributes remain
6235     local current_attribute_type = false
6236     while current_attribute_type ~= attribute_type and
6237           #self.active_attributes > 0 do
6238         local attributes, _, end_output
6239         current_attribute_type, attributes, _, end_output = table.unpack(
6240             self.active_attributes[#self.active_attributes])
6241         local attribute_type_level
6242             = self.attribute_type_levels[current_attribute_type]
6243         self.attribute_type_levels[current_attribute_type]
6244             = attribute_type_level - 1
6245         if self.is_writing and end_output ~= nil then
6246             table.insert(buf, end_output)
6247         end
6248         table.remove(self.active_attributes, #self.active_attributes)
6249         -- handle slicing
6250         if attributes["#" .. self.slice_end_identifier] ~= nil
6251             and self.slice_end_type == "$" then
6252             if self.is_writing then
6253                 table.insert(buf, self.undosep())
6254                 table.insert(buf, tear_down_attributes())
6255             end
6256             self.is_writing = false
6257         end
6258         if attributes["#" .. self.slice_begin_identifier] ~= nil and
6259             self.slice_begin_type == "$" then
6260             self.is_writing = true
6261             table.insert(buf, apply_attributes())

```

```

6262     end
6263     end
6264     return buf
6265 end

```

Create an auto identifier string by stripping and converting characters from string `s`.

```

6266 local function create_auto_identifier(s)
6267     local buffer = {}
6268     local prev_space = false
6269     local letter_found = false
6270     local normalized_s = s
6271     if not options.unicodeNormalization
6272         or options.unicodeNormalizationForm ~= "nfc" then
6273         normalized_s = uni_algos.normalize.NFC(normalized_s)
6274     end
6275
6276     for _, code in utf8.codes(normalized_s) do
6277         local char = utf8.char(code)
6278
6279         -- Remove everything up to the first letter.
6280         if not letter_found then
6281             local is_letter = unicode.utf8.match(char, "%a")
6282             if is_letter then
6283                 letter_found = true
6284             else
6285                 goto continue
6286             end
6287         end
6288
6289         -- Remove all non-alphanumeric characters, except underscores,
6290         -- hyphens, and periods.
6291         if not unicode.utf8.match(char, "[%w_-%.%s]") then
6292             goto continue
6293         end
6294
6295         -- Replace all spaces and newlines with hyphens.
6296         if unicode.utf8.match(char, "[%s\n]") then
6297             char = "-"
6298             if prev_space then
6299                 goto continue
6300             else
6301                 prev_space = true
6302             end
6303         else
6304             -- Convert all alphabetic characters to lowercase.
6305             char = unicode.utf8.lower(char)
6306             prev_space = false
6307         end

```

```

6308
6309     table.insert(buffer, char)
6310
6311     ::continue::
6312 end
6313
6314 if prev_space then
6315     table.remove(buffer)
6316 end
6317
6318 local identifier = #buffer == 0 and "section"
6319                 or table.concat(buffer, "")
6320 return identifier
6321 end

```

Create an GitHub-flavored auto identifier string by stripping and converting characters from string `s`.

```

6322 local function create_gfm_auto_identifier(s)
6323     local buffer = {}
6324     local prev_space = false
6325     local letter_found = false
6326     local normalized_s = s
6327     if not options.unicodeNormalization
6328         or options.unicodeNormalizationForm ~= "nfc" then
6329         normalized_s = uni_algos.normalize.NFC(normalized_s)
6330     end
6331
6332     for _, code in utf8.codes(normalized_s) do
6333         local char = utf8.char(code)
6334
6335         -- Remove everything up to the first non-space.
6336         if not letter_found then
6337             local is_letter = unicode.utf8.match(char, "%S")
6338             if is_letter then
6339                 letter_found = true
6340             else
6341                 goto continue
6342             end
6343         end
6344
6345         -- Remove all non-alphanumeric characters, except underscores
6346         -- and hyphens.
6347         if not unicode.utf8.match(char, "[%w_-%s]") then
6348             prev_space = false
6349             goto continue
6350         end
6351     end

```

```

6352     -- Replace all spaces and newlines with hyphens.
6353     if unicode.utf8.match(char, "[%s\n]") then
6354         char = "-"
6355         if prev_space then
6356             goto continue
6357         else
6358             prev_space = true
6359         end
6360     else
6361         -- Convert all alphabetic characters to lowercase.
6362         char = unicode.utf8.lower(char)
6363         prev_space = false
6364     end
6365
6366     table.insert(buffer, char)
6367
6368     ::continue::
6369 end
6370
6371 if prev_space then
6372     table.remove(buffer)
6373 end
6374
6375 local identifier = #buffer == 0 and "section"
6376                  or table.concat(buffer, "")
6377 return identifier
6378 end

```

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

```

6379 self.secbegin_text = "\\markdownRendererSectionBegin\n"
6380 self.secend_text = "\n\\markdownRendererSectionEnd "
6381 function self.heading(s, level, attributes)
6382     local buf = {}
6383     local flat_text, inlines = table.unpack(s)
6384
6385     -- push empty attributes for implied sections
6386     while self.attribute_type_levels["heading"] < level - 1 do
6387         table.insert(buf,
6388             self.push_attributes("heading",
6389                 nil,
6390                 self.secbegin_text,
6391                 self.secend_text))
6392     end
6393
6394     -- pop attributes for sections that have ended
6395     while self.attribute_type_levels["heading"] >= level do

```

```

6396     table.insert(buf, self.pop_attributes("heading"))
6397 end
6398
6399 -- construct attributes for the new section
6400 local auto_identifiers = {}
6401 if self.options.autoIdentifiers then
6402     table.insert(auto_identifiers, create_auto_identifier(flat_text))
6403 end
6404 if self.options.gfmAutoIdentifiers then
6405     table.insert(auto_identifiers,
6406                 create_gfm_auto_identifier(flat_text))
6407 end
6408 local normalized_attributes = normalize_attributes(attributes,
6409                                                  auto_identifiers)
6410
6411 -- push attributes for the new section
6412 local start_output = {}
6413 local end_output = {}
6414 table.insert(start_output, self.secbegin_text)
6415 table.insert(end_output, self.secend_text)
6416
6417 table.insert(buf, self.push_attributes("heading",
6418                                     normalized_attributes,
6419                                     start_output,
6420                                     end_output))
6421 assert(self.attribute_type_levels["heading"] == level)
6422
6423 -- render the heading and its attributes
6424 if self.is_writing and #normalized_attributes > 0 then
6425     table.insert(buf,
6426                 "\\markdownRendererHeaderAttributeContextBegin\n")
6427     table.insert(buf, self.attributes(normalized_attributes, false))
6428 end
6429
6430 local cmd
6431 level = level + options.shiftHeadings
6432 if level <= 1 then
6433     cmd = "\\markdownRendererHeadingOne"
6434 elseif level == 2 then
6435     cmd = "\\markdownRendererHeadingTwo"
6436 elseif level == 3 then
6437     cmd = "\\markdownRendererHeadingThree"
6438 elseif level == 4 then
6439     cmd = "\\markdownRendererHeadingFour"
6440 elseif level == 5 then
6441     cmd = "\\markdownRendererHeadingFive"
6442 elseif level >= 6 then

```

```

6443     cmd = "\\markdownRendererHeadingSix"
6444   else
6445     cmd = ""
6446   end
6447   if self.is_writing then
6448     table.insert(buf, {cmd, "{", inlines, "}"})
6449   end
6450
6451   if self.is_writing and #normalized_attributes > 0 then
6452     table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd{")
6453   end
6454
6455   return buf
6456 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.

```

6457 function self.get_state()
6458   return {
6459     is_writing=self.is_writing,
6460     flatten_inlines=self.flatten_inlines,
6461     active_attributes={table.unpack(self.active_attributes)},
6462   }
6463 end

```

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

```

6464 function self.set_state(s)
6465   local previous_state = self.get_state()
6466   for key, value in pairs(s) do
6467     self[key] = value
6468   end
6469   return previous_state
6470 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`.

```

6471 function self.defer_call(f)
6472   local previous_state = self.get_state()
6473   return function(...)
6474     local state = self.set_state(previous_state)
6475     local return_value = f(...)
6476     self.set_state(state)
6477     return return_value
6478   end
6479 end
6480

```

```
6481 return self
6482 end
```

### 3.1.4 Parsers

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

```
6483 local parsers = {}
```

#### 3.1.4.1 Basic Parsers

```
6484 parsers.percent = P("%")
6485 parsers.at = P("@")
6486 parsers.comma = P(",")
6487 parsers.asterisk = P("*")
6488 parsers.dash = P("-")
6489 parsers.plus = P("+")
6490 parsers.underscore = P("_")
6491 parsers.period = P(".")
6492 parsers.hash = P("#")
6493 parsers.dollar = P("$")
6494 parsers.ampersand = P("&")
6495 parsers.backtick = P("`")
6496 parsers.less = P("<")
6497 parsers.more = P(">")
6498 parsers.space = P(" ")
6499 parsers.squote = P("'")
6500 parsers.dquote = P('"')
6501 parsers.lparent = P("(")
6502 parsers.rparent = P(")")
6503 parsers.lbracket = P("[")
6504 parsers.rbracket = P("]")
6505 parsers.lbrace = P("{")
6506 parsers.rbrace = P("}")
6507 parsers.circumflex = P("^")
6508 parsers.slash = P("/")
6509 parsers.equal = P("=")
6510 parsers.colon = P(":")
6511 parsers.semicolon = P(";")
6512 parsers.exclamation = P("!")
6513 parsers.pipe = P("|")
6514 parsers.tilde = P("~")
6515 parsers.backslash = P("\\")
6516 parsers.tab = P("\t")
6517 parsers.newline = P("\n")
6518
6519 parsers.digit = R("09")
```



```

6520 parsers.hexdigit           = R("09","af","AF")
6521 parsers.letter            = R("AZ","az")
6522 parsers.alphanumeric      = R("AZ","az","09")
6523 parsers.keyword           = parsers.letter
6524                            * (parsers.alphanumeric + parsers.dash)^0
6525
6526 parsers.doubleasterisks    = P("**")
6527 parsers.doubleunderscores  = P("__")
6528 parsers.doubletildes       = P("~~")
6529 parsers.fourspace         = P("    ")
6530
6531 parsers.any                 = P(1)
6532 parsers.succeed            = P(true)
6533 parsers.fail               = P(false)
6534
6535 parsers.internal_punctuation = S(":,;.?.")
6536 parsers.ascii_punctuation  = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")

```

### 3.1.5 Unicode punctuation

This section documents the Unicode punctuation<sup>33</sup> recognized by the markdown reader. The punctuation is organized in the `parsers.punctuation` table according to the number of bytes occupied after conversion to UTF8.

(CommonMark Spec, Version 0.31.2 (2024-01-28))

```

6537 ;(function()
6538   local pathname = assert(kpse.find_file("UnicodeData.txt"),
6539     [[Could not locate file "UnicodeData.txt"]])
6540   local file = assert(io.open(pathname, "r"),
6541     [[Could not open file "UnicodeData.txt"]])

```

In order to minimize the size and speed of the parser, we will first construct a prefix tree of UTF-8 encodings for all codepoints of a given code length.

```

6542   local prefix_trees = {}
6543   for line in file:lines() do
6544     local codepoint, major_category = line:match("^(%x+);[^;]*;(%)a")
6545     if major_category == "P" or major_category == "S" then
6546       local code = unicode.utf8.char(tonumber(codepoint, 16))
6547       if prefix_trees[#code] == nil then
6548         prefix_trees[#code] = {}
6549       end
6550       local node = prefix_trees[#code]
6551       for i = 1, #code do

```

<sup>33</sup>See <https://spec.commonmark.org/0.31.2/#unicode-punctuation-character>.

```

6552     local byte = code:sub(i, i)
6553     if i < #code then
6554         if node[byte] == nil then
6555             node[byte] = {}
6556         end
6557         node = node[byte]
6558     else
6559         table.insert(node, byte)
6560     end
6561 end
6562 end
6563 end
6564 assert(file:close())

```

Next, we will construct a parser out of the prefix tree.

```

6565 local function depth_first_search(node, path, visit, leave)
6566     visit(node, path)
6567     for label, child in pairs(node) do
6568         if type(child) == "table" then
6569             depth_first_search(child, path .. label, visit, leave)
6570         else
6571             visit(child, path)
6572         end
6573     end
6574     leave(node, path)
6575 end
6576
6577 parsers.punctuation = {}
6578 for length, prefix_tree in pairs(prefix_trees) do
6579     local subparsers = {}
6580     depth_first_search(prefix_tree, "", function(node, path)
6581         if type(node) == "table" then
6582             subparsers[path] = parsers.fail
6583         else
6584             assert(type(node) == "string")
6585             subparsers[path] = subparsers[path] + S(node)
6586         end
6587     end, function(_, path)
6588         if #path > 0 then
6589             local byte = path:sub(#path, #path)
6590             local parent_path = path:sub(1, #path-1)
6591             subparsers[parent_path] = subparsers[parent_path]
6592                 + S(byte) * subparsers[path]
6593         else
6594             parsers.punctuation[length] = subparsers[path]
6595         end
6596     end)
6597     assert(parsers.punctuation[length] ~= nil)

```

```

6598 end
6599 end)()
6600
6601 parsers.escapable = parsers.ascii_punctuation
6602 parsers.anyescaped = parsers.backslash / ""
6603 * parsers.escapable
6604 + parsers.any
6605
6606 parsers.spacechar = S("\t ")
6607 parsers.spacing = S(" \n\r\t")
6608 parsers.nonpacechar = parsers.any - parsers.spacing
6609 parsers.optionalspace = parsers.spacechar^0
6610
6611 parsers.normalchar = parsers.any - (V("SpecialChar")
6612 + parsers.spacing)
6613 parsers.eof = -parsers.any
6614 parsers.nonindentspace = parsers.space^-3 * - parsers.spacechar
6615 parsers.indent = parsers.space^-3 * parsers.tab
6616 + parsers.fourspace / ""
6617 parsers.linechar = P(1 - parsers.newline)
6618
6619 parsers.blankline = parsers.optionalspace
6620 * parsers.newline / "\n"
6621 parsers.blanklines = parsers.blankline^0
6622 parsers.skipblanklines = ( parsers.optionalspace
6623 * parsers.newline)^0
6624 parsers.indentedline = parsers.indent /""
6625 * C( parsers.linechar^1
6626 * parsers.newline^-1)
6627 parsers.optionallyindentedline = parsers.indent^-1 /""
6628 * C( parsers.linechar^1
6629 * parsers.newline^-1)
6630 parsers.sp = parsers.spacing^0
6631 parsers.spnl = parsers.optionalspace
6632 * ( parsers.newline
6633 * parsers.optionalspace)^-1
6634 parsers.line = parsers.linechar^0 * parsers.newline
6635 parsers.nonemptyline = parsers.line - parsers.blankline

```

### 3.1.5.1 Parsers Used for Indentation

```

6636
6637 parsers.leader = parsers.space^-3
6638

```

Check if a trail exists and is non-empty in the indent table `indent_table`.

```

6639 local function has_trail(indent_table)
6640 return indent_table ~= nil and

```

```

6641     indent_table.trail ~= nil and
6642     next(indent_table.trail) ~= nil
6643 end
6644

```

Check if indent table `indent_table` has any indents.

```

6645 local function has_indents(indent_table)
6646     return indent_table ~= nil and
6647         indent_table.indents ~= nil and
6648         next(indent_table.indents) ~= nil
6649 end
6650

```

Add a trail `trail_info` to the indent table `indent_table`.

```

6651 local function add_trail(indent_table, trail_info)
6652     indent_table.trail = trail_info
6653     return indent_table
6654 end
6655

```

Remove a trail `trail_info` from the indent table `indent_table`.

```

6656 local function remove_trail(indent_table)
6657     indent_table.trail = nil
6658     return indent_table
6659 end
6660

```

Update the indent table `indent_table` by adding or removing a new indent `add`.

```

6661 local function update_indent_table(indent_table, new_indent, add)
6662     indent_table = remove_trail(indent_table)
6663
6664     if not has_indents(indent_table) then
6665         indent_table.indents = {}
6666     end
6667
6668
6669     if add then
6670         indent_table.indents[#indent_table.indents + 1] = new_indent
6671     else
6672         if indent_table.indents[#indent_table.indents].name
6673             == new_indent.name then
6674             indent_table.indents[#indent_table.indents] = nil
6675         end
6676     end
6677
6678     return indent_table
6679 end
6680

```

Remove an indent by its name `name`.

```
6681 local function remove_indent(name)
6682   local remove_indent_level =
6683     function(s, i, indent_table) -- luacheck: ignore s i
6684       indent_table = update_indent_table(indent_table, {name=name},
6685                                         false)
6686       return true, indent_table
6687     end
6688
6689   return Cg(Cmt(Cb("indent_info"), remove_indent_level), "indent_info")
6690 end
6691
```

Process the spacing of a string of spaces and tabs `spacing` with preceding indent width from the start of the line `indent` and strip up to `left_strip_length` spaces. Return the remainder `remainder` and whether there is enough spaces to produce a code `is_code`. Return how many spaces were stripped, as well as if the minimum was met `is_minimum` and what remainder it left `minimum_remainder`.

```
6692 local function process_starter_spacing(indent, spacing,
6693                                       minimum, left_strip_length)
6694   left_strip_length = left_strip_length or 0
6695
6696   local count = 0
6697   local tab_value = 4 - (indent) % 4
6698
6699   local code_started, minimum_found = false, false
6700   local code_start, minimum_remainder = "", ""
6701
6702   local left_total_stripped = 0
6703   local full_remainder = ""
6704
6705   if spacing ~= nil then
6706     for i = 1, #spacing do
6707       local character = spacing:sub(i, i)
6708
6709       if character == "\t" then
6710         count = count + tab_value
6711         tab_value = 4
6712       elseif character == " " then
6713         count = count + 1
6714         tab_value = 4 - (1 - tab_value) % 4
6715       end
6716
6717       if (left_strip_length ~= 0) then
6718         local possible_to_strip = math.min(count, left_strip_length)
6719         count = count - possible_to_strip
6720         left_strip_length = left_strip_length - possible_to_strip

```

```

6721     left_total_stripped = left_total_stripped + possible_to_strip
6722 else
6723     full_remainder = full_remainder .. character
6724 end
6725
6726 if (minimum_found) then
6727     minimum_remainder = minimum_remainder .. character
6728 elseif (count >= minimum) then
6729     minimum_found = true
6730     minimum_remainder = minimum_remainder
6731         .. string.rep(" ", count - minimum)
6732 end
6733
6734 if (code_started) then
6735     code_start = code_start .. character
6736 elseif (count >= minimum + 4) then
6737     code_started = true
6738     code_start = code_start
6739         .. string.rep(" ", count - (minimum + 4))
6740 end
6741 end
6742 end
6743
6744 local remainder
6745 if (code_started) then
6746     remainder = code_start
6747 else
6748     remainder = string.rep(" ", count - minimum)
6749 end
6750
6751 local is_minimum = count >= minimum
6752 return {
6753     is_code = code_started,
6754     remainder = remainder,
6755     left_total_stripped = left_total_stripped,
6756     is_minimum = is_minimum,
6757     minimum_remainder = minimum_remainder,
6758     total_length = count,
6759     full_remainder = full_remainder
6760 }
6761 end
6762

```

Count the total width of all indents in the indent table `indent_table`.

```

6763 local function count_indent_tab_level(indent_table)
6764     local count = 0
6765     if not has_indents(indent_table) then
6766         return count

```

```

6767 end
6768
6769 for i=1, #indent_table.indents do
6770     count = count + indent_table.indents[i].length
6771 end
6772 return count
6773 end
6774

```

Count the total width of a delimiter `delimiter`.

```

6775 local function total_delimiter_length(delimiter)
6776     local count = 0
6777     if type(delimiter) == "string" then return #delimiter end
6778     for _, value in pairs(delimiter) do
6779         count = count + total_delimiter_length(value)
6780     end
6781     return count
6782 end
6783

```

Process the container starter `starter` of a type `indent_type`. Adjust the width of the indent if the delimiter is followed only by whitespaces `is_blank`.

```

6784 local function process_starter_indent(_, _, indent_table, starter,
6785                                     is_blank, indent_type, breakable)
6786     local last_trail = starter[1]
6787     local delimiter = starter[2]
6788     local raw_new_trail = starter[3]
6789
6790     if indent_type == "bq" and not breakable then
6791         indent_table.ignore_blockquote_blank = true
6792     end
6793
6794     if has_trail(indent_table) then
6795         local trail = indent_table.trail
6796         if trail.is_code then
6797             return false
6798         end
6799         last_trail = trail.remainder
6800     else
6801         local sp = process_starter_spacing(0, last_trail, 0, 0)
6802
6803         if sp.is_code then
6804             return false
6805         end
6806         last_trail = sp.remainder
6807     end
6808
6809     local preceding_indentation = count_indent_tab_level(indent_table) % 4

```

```

6810 local last_trail_length = #last_trail
6811 local delimiter_length = total_delimiter_length(delimiter)
6812
6813 local total_indent_level = preceding_indentation + last_trail_length
6814                        + delimiter_length
6815
6816 local sp = {}
6817 if not is_blank then
6818     sp = process_starter_spacing(total_indent_level, raw_new_trail,
6819                               0, 1)
6820 end
6821
6822 local del_trail_length = sp.left_total_stripped
6823 if is_blank then
6824     del_trail_length = 1
6825 elseif not sp.is_code then
6826     del_trail_length = del_trail_length + #sp.remainder
6827 end
6828
6829 local indent_length = last_trail_length + delimiter_length
6830                    + del_trail_length
6831 local new_indent_info = {name=indent_type, length=indent_length}
6832
6833 indent_table = update_indent_table(indent_table, new_indent_info,
6834                                   true)
6835 indent_table = add_trail(indent_table,
6836                          {is_code=sp.is_code,
6837                           remainder=sp.remainder,
6838                           total_length=sp.total_length,
6839                           full_remainder=sp.full_remainder})
6840
6841 return true, indent_table
6842 end
6843

```

Return the pattern corresponding with the indent name `name`.

```

6844 local function decode_pattern(name)
6845     local delimiter = parsers.succeed
6846     if name == "bq" then
6847         delimiter = parsers.more
6848     end
6849
6850     return C(parsers.optionalspace) * C(delimiter)
6851            * C(parsers.optionalspace) * Cp()
6852 end
6853

```



Find the first blank-only indent of the indent table `indent_table` followed by blank-only indents.

```
6854 local function left_blank_starter(indent_table)
6855   local blank_starter_index
6856
6857   if not has_indents(indent_table) then
6858     return
6859   end
6860
6861   for i = #indent_table.indents,1,-1 do
6862     local value = indent_table.indents[i]
6863     if value.name == "li" then
6864       blank_starter_index = i
6865     else
6866       break
6867     end
6868   end
6869
6870   return blank_starter_index
6871 end
6872
```

Apply the patterns decoded from the indents of the indent table `indent_table` iteratively starting at position `index` of the string `s`. If the `is_optional` mode is selected, match as many patterns as possible, else match all or fail. With the option `is_blank`, the parsing behaves as optional after the position of a blank-only indent has been surpassed.

```
6873 local function traverse_indent(s, i, indent_table, is_optional,
6874                               is_blank, current_line_indents)
6875   local new_index = i
6876
6877   local preceding_indentation = 0
6878   local current_trail = {}
6879
6880   local blank_starter = left_blank_starter(indent_table)
6881
6882   if current_line_indents == nil then
6883     current_line_indents = {}
6884   end
6885
6886   for index = 1,#indent_table.indents do
6887     local value = indent_table.indents[index]
6888     local pattern = decode_pattern(value.name)
6889
6890     -- match decoded pattern
6891     local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
6892     if new_indent_info == nil then
```

```

6893     local blankline_end = lpeg.match(
6894         Ct(parsers.blankline * Cg(Cp(), "pos")), s, new_index)
6895     if is_optional or not indent_table.ignore_blockquote_blank
6896         or not blankline_end then
6897         return is_optional, new_index, current_trail,
6898             current_line_indents
6899     end
6900
6901     return traverse_indent(s, tonumber(blankline_end.pos),
6902         indent_table, is_optional, is_blank,
6903         current_line_indents)
6904 end
6905
6906 local raw_last_trail = new_indent_info[1]
6907 local delimiter = new_indent_info[2]
6908 local raw_new_trail = new_indent_info[3]
6909 local next_index = new_indent_info[4]
6910
6911 local space_only = delimiter == ""
6912
6913 -- check previous trail
6914 if not space_only and next(current_trail) == nil then
6915     local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
6916     current_trail = {is_code=sp.is_code, remainder=sp.remainder,
6917         total_length=sp.total_length,
6918         full_remainder=sp.full_remainder}
6919 end
6920
6921 if next(current_trail) ~= nil then
6922     if not space_only and current_trail.is_code then
6923         return is_optional, new_index, current_trail,
6924             current_line_indents
6925     end
6926     if current_trail.internal_remainder ~= nil then
6927         raw_last_trail = current_trail.internal_remainder
6928     end
6929 end
6930
6931 local raw_last_trail_length = 0
6932 local delimiter_length = 0
6933
6934 if not space_only then
6935     delimiter_length = #delimiter
6936     raw_last_trail_length = #raw_last_trail
6937 end
6938
6939 local total_indent_level = preceding_indentation

```

```

6940             + raw_last_trail_length + delimiter_length
6941
6942     local spacing_to_process
6943     local minimum = 0
6944     local left_strip_length = 0
6945
6946     if not space_only then
6947         spacing_to_process = raw_new_trail
6948         left_strip_length = 1
6949     else
6950         spacing_to_process = raw_last_trail
6951         minimum = value.length
6952     end
6953
6954     local sp = process_starter_spacing(total_indent_level,
6955                                     spacing_to_process, minimum,
6956                                     left_strip_length)
6957
6958     if space_only and not sp.is_minimum then
6959         return is_optional or (is_blank and blank_starter <= index),
6960             new_index, current_trail, current_line_indents
6961     end
6962
6963     local indent_length = raw_last_trail_length + delimiter_length
6964                       + sp.left_total_stripped
6965
6966     -- update info for the next pattern
6967     if not space_only then
6968         preceding_indentation = preceding_indentation + indent_length
6969     else
6970         preceding_indentation = preceding_indentation + value.length
6971     end
6972
6973     current_trail = {is_code=sp.is_code, remainder=sp.remainder,
6974                    internal_remainder=sp.minimum_remainder,
6975                    total_length=sp.total_length,
6976                    full_remainder=sp.full_remainder}
6977
6978     current_line_indents[#current_line_indents + 1] = new_indent_info
6979     new_index = next_index
6980 end
6981
6982 return true, new_index, current_trail, current_line_indents
6983 end
6984

```

Check if a code trail is expected.

```

6985 local function check_trail(expect_code, is_code)

```

```

6986 return (expect_code and is_code) or (not expect_code and not is_code)
6987 end
6988

```

Check if the current trail of the `indent_table` would produce code if it is expected `expect_code` or it would not if it is not. If there is no trail, process and check the current spacing `spacing`.

```

6989 local check_trail_joined =
6990 function(s, i, indent_table, -- luacheck: ignore s i
6991         spacing, expect_code, omit_remainder)
6992     local is_code
6993     local remainder
6994
6995     if has_trail(indent_table) then
6996         local trail = indent_table.trail
6997         is_code = trail.is_code
6998         if is_code then
6999             remainder = trail.remainder
7000         else
7001             remainder = trail.full_remainder
7002         end
7003     else
7004         local sp = process_starter_spacing(0, spacing, 0, 0)
7005         is_code = sp.is_code
7006         if is_code then
7007             remainder = sp.remainder
7008         else
7009             remainder = sp.full_remainder
7010         end
7011     end
7012
7013     local result = check_trail(expect_code, is_code)
7014     if omit_remainder then
7015         return result
7016     end
7017     return result, remainder
7018 end
7019

```

Check if the current trail of the `indent_table` is of length between `min` and `max`.

```

7020 local check_trail_length =
7021 function(s, i, indent_table, -- luacheck: ignore s i
7022         spacing, min, max)
7023     local trail
7024
7025     if has_trail(indent_table) then
7026         trail = indent_table.trail
7027     else

```

```

7028     trail = process_starter_spacing(0, spacing, 0, 0)
7029 end
7030
7031 local total_length = trail.total_length
7032 if total_length == nil then
7033     return false
7034 end
7035
7036 return min <= total_length and total_length <= max
7037 end
7038

```

Check the indentation of the continuation line, optionally with the mode `is_optional` selected. Check blank line exclusively with `is_blank`.

```

7039 local function check_continuation_indentation(s, i, indent_table,
7040                                             is_optional, is_blank)
7041     if not has_indents(indent_table) then
7042         return true
7043     end
7044
7045     local passes, new_index, current_trail, current_line_indents =
7046         traverse_indent(s, i, indent_table, is_optional, is_blank)
7047
7048     if passes then
7049         indent_table.current_line_indents = current_line_indents
7050         indent_table = add_trail(indent_table, current_trail)
7051         return new_index, indent_table
7052     end
7053     return false
7054 end
7055

```

Get name of the last indent from the `indent_table`.

```

7056 local function get_last_indent_name(indent_table)
7057     if has_indents(indent_table) then
7058         return indent_table.indents[#indent_table.indents].name
7059     end
7060 end
7061

```

Remove the remainder altogether if the last indent from the `indent_table` is blank-only.

```

7062 local function remove_remainder_if_blank(indent_table, remainder)
7063     if get_last_indent_name(indent_table) == "li" then
7064         return ""
7065     end
7066     return remainder
7067 end

```

7068

Take the trail `trail` or create a new one from `spacing` and compare it with the expected `trail_type`. On success return the index `i` and the remainder of the trail.

```
7069 local check_trail_type =
7070   function(s, i, -- luacheck: ignore s i
7071           trail, spacing, trail_type)
7072     if trail == nil then
7073       trail = process_starter_spacing(0, spacing, 0, 0)
7074     end
7075
7076     if trail_type == "non-code" then
7077       return check_trail(false, trail.is_code)
7078     end
7079     if trail_type == "code" then
7080       return check_trail(true, trail.is_code)
7081     end
7082     if trail_type == "full-code" then
7083       if (trail.is_code) then
7084         return i, trail.remainder
7085       end
7086       return i, ""
7087     end
7088     if trail_type == "full-any" then
7089       return i, trail.internal_remainder
7090     end
7091   end
7092
```

Stores or restores an `is_freezing` trail from indent table `indent_table`.

```
7093 local trail_freezing =
7094   function(s, i, -- luacheck: ignore s i
7095           indent_table, is_freezing)
7096     if is_freezing then
7097       if indent_table.is_trail_frozen then
7098         indent_table.trail = indent_table.frozen_trail
7099       else
7100         indent_table.frozen_trail = indent_table.trail
7101         indent_table.is_trail_frozen = true
7102       end
7103     else
7104       indent_table.frozen_trail = nil
7105       indent_table.is_trail_frozen = false
7106     end
7107     return true, indent_table
7108   end
7109
```

Check the indentation of the continuation line, optionally with the mode `is_optional` selected. Check blank line specifically with `is_blank`. Additionally, also directly check the new trail with a type `trail_type`.

```
7110 local check_continuation_indentation_and_trail =
7111   function (s, i, indent_table, is_optional, is_blank, trail_type,
7112             reset_rem, omit_remainder)
7113     if not has_indents(indent_table) then
7114       local spacing, new_index = lpeg.match( C(parsers.spacechar~0)
7115                                             * Cp(), s, i)
7116       local result, remainder = check_trail_type(s, i,
7117         indent_table.trail, spacing, trail_type)
7118       if remainder == nil then
7119         if result then
7120           return new_index
7121         end
7122         return false
7123       end
7124       if result then
7125         return new_index, remainder
7126       end
7127       return false
7128     end
7129
7130     local passes, new_index, current_trail = traverse_indent(s, i,
7131       indent_table, is_optional, is_blank)
7132
7133     if passes then
7134       local spacing
7135       if current_trail == nil then
7136         local newer_spacing, newer_index = lpeg.match(
7137           C(parsers.spacechar~0) * Cp(), s, i)
7138         current_trail = process_starter_spacing(0, newer_spacing, 0, 0)
7139         new_index = newer_index
7140         spacing = newer_spacing
7141       else
7142         spacing = current_trail.remainder
7143       end
7144       local result, remainder = check_trail_type(s, new_index,
7145         current_trail, spacing, trail_type)
7146       if remainder == nil or omit_remainder then
7147         if result then
7148           return new_index
7149         end
7150         return false
7151       end
7152     end
7153     if is_blank and reset_rem then
```

```

7154     remainder = remove_remainder_if_blank(indent_table, remainder)
7155     end
7156     if result then
7157         return new_index, remainder
7158     end
7159     return false
7160 end
7161 return false
7162 end
7163

```

The following patterns check whitespace indentation at the start of a block.

```

7164 parsers.check_trail = Cmt( Cb("indent_info") * C(parsers.spacechar^0)
7165     * Cc(false), check_trail_joined)
7166
7167 parsers.check_trail_no_rem = Cmt( Cb("indent_info")
7168     * C(parsers.spacechar^0) * Cc(false)
7169     * Cc(true), check_trail_joined)
7170
7171 parsers.check_code_trail = Cmt( Cb("indent_info")
7172     * C(parsers.spacechar^0)
7173     * Cc(true), check_trail_joined)
7174
7175 parsers.check_trail_length_range = function(min, max)
7176     return Cmt( Cb("indent_info") * C(parsers.spacechar^0) * Cc(min)
7177         * Cc(max), check_trail_length)
7178 end
7179
7180 parsers.check_trail_length = function(n)
7181     return parsers.check_trail_length_range(n, n)
7182 end
7183

```

The following patterns handle trail backup, to prevent a failing pattern to modify it before passing it to the next.

```

7184 parsers.freeze_trail = Cg( Cmt(Cb("indent_info")
7185     * Cc(true), trail_freezing), "indent_info")
7186
7187 parsers.unfreeze_trail = Cg(Cmt(Cb("indent_info") * Cc(false),
7188     trail_freezing), "indent_info")
7189

```

The following patterns check indentation in continuation lines as defined by the container start.

```

7190 parsers.check_minimal_indent = Cmt(Cb("indent_info") * Cc(false),
7191     check_continuation_indentation)
7192
7193 parsers.check_optional_indent = Cmt(Cb("indent_info") * Cc(true),

```



```

7194                                     check_continuation_indentation)
7195
7196 parsers.check_minimal_blank_indent
7197 = Cmt( Cb("indent_info") * Cc(false)
7198       * Cc(true)
7199       , check_continuation_indentation)
7200

```

The following patterns check indentation in continuation lines as defined by the container start. Additionally the subsequent trail is also directly checked.

```

7201
7202 parsers.check_minimal_indent_and_trail =
7203   Cmt( Cb("indent_info")
7204       * Cc(false) * Cc(false) * Cc("non-code") * Cc(true)
7205       , check_continuation_indentation_and_trail)
7206
7207 parsers.check_minimal_indent_and_code_trail =
7208   Cmt( Cb("indent_info")
7209       * Cc(false) * Cc(false) * Cc("code") * Cc(false)
7210       , check_continuation_indentation_and_trail)
7211
7212 parsers.check_minimal_blank_indent_and_full_code_trail =
7213   Cmt( Cb("indent_info")
7214       * Cc(false) * Cc(true) * Cc("full-code") * Cc(true)
7215       , check_continuation_indentation_and_trail)
7216
7217 parsers.check_minimal_indent_and_any_trail =
7218   Cmt( Cb("indent_info")
7219       * Cc(false) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
7220       , check_continuation_indentation_and_trail)
7221
7222 parsers.check_minimal_blank_indent_and_any_trail =
7223   Cmt( Cb("indent_info")
7224       * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
7225       , check_continuation_indentation_and_trail)
7226
7227 parsers.check_minimal_blank_indent_and_any_trail_no_rem =
7228   Cmt( Cb("indent_info")
7229       * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(true)
7230       , check_continuation_indentation_and_trail)
7231
7232 parsers.check_optional_indent_and_any_trail =
7233   Cmt( Cb("indent_info")
7234       * Cc(true) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
7235       , check_continuation_indentation_and_trail)
7236
7237 parsers.check_optional_blank_indent_and_any_trail =

```

```

7238 Cmt( Cb("indent_info")
7239     * Cc(true) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
7240     , check_continuation_indentation_and_trail)
7241

```

The following patterns specify behaviour around newlines.

```

7242
7243 parsers.spnlc_noexc = parsers.optionalspace
7244                      * ( parsers.newline
7245                        * parsers.check_minimal_indent_and_any_trail)^-1
7246
7247 parsers.spnlc = parsers.optionalspace
7248                * (V("EndlineNoSub"))^-1
7249
7250 parsers.spnlc_sep = parsers.optionalspace * V("EndlineNoSub")
7251                  + parsers.spacechar^1
7252
7253 parsers.only_blank = parsers.spacechar^0
7254                    * (parsers.newline + parsers.eof)
7255
7256 % \end{macrocode}
7257 % \begin{figure}
7258 % \hspace*{-0.1\textwidth}
7259 % \begin{minipage}{1.2\textwidth}
7260 % \centering
7261 % \begin{tikzpicture}[shorten >=1pt, line width=0.1mm, >={Stealth[length=2mm]}, node
7262 % \node[state, initial by diamond, accepting] (noop) {initial};
7263 % \node[state] (odd_backslash) [above right=of noop] {odd backslash};
7264 % \node[state] (even_backslash) [below right=of odd_backslash] {even backslash};
7265 % \node[state] (comment) [below=of noop] {comment};
7266 % \node[state] (leading_spaces) [below=of even_backslash, align=center] {leading tabs};
7267 % \node[state] (blank_line) [below right=of comment] {blank line};
7268 % \path[->]
7269 % (noop) edge [in=150, out=180, loop] node [align=center, yshift=-0.75cm] {match [^\
7270 %     edge [bend right=10] node [below right=-0.2cm] {match \textbackslash} (odd_b
7271 %     edge [bend left=30] node [left, align=center] {match \%\\capture \textbacksl
7272 % (comment) edge [in=305, out=325, loop] node [xshift=-1.2cm] {match [^\wedge$$\drsh$
7273 %     edge [bend left=10] node {match $\drsh$} (leading_spaces)
7274 % (leading_spaces) edge [loop below] node {match [\textvisiblespace$\rightleftarrows$
7275 %     edge [bend right=90] node [right] {match \textbackslash} (odd_back
7276 %     edge [bend left=10] node {match \%} (comment)
7277 %     edge [bend right=10] node {$\epsilon$} (blank_line)
7278 %     edge [bend left=10] node [align=center, right=0.3cm] {match [^\we
7279 % (blank_line) edge [loop below] node {match [\textvisiblespace$\rightleftarrows$}} (
7280 %     edge [bend left=90] node [align=center, below=1.2cm] {match $\drsh$\\
7281 % (odd_backslash) edge [bend right=10] node [align=center, xshift=-0.3cm, yshift=0.2c
7282 %     edge [bend right=10] node [align=center, above left=-
7283 % 0.3cm, xshift=0.1cm] {match [^\wedge$\textbackslash}\\for \%, capture \textbacksl

```

```

7283 % (even_backslash) edge [bend left=10] node {$\epsilon$} (noop);
7284 % \end{tikzpicture}
7285 % \caption{A pushdown automaton that recognizes \TeX{} comments}
7286 % \label{fig:commented_line}
7287 % \end{minipage}
7288 % \end{figure}
7289 % \begin{markdown}
7290 %
7291 % The \luamdef{parsers.commented_line}^1 parser recognizes the regular
7292 % language of \TeX{} comments, see an equivalent finite automaton in Figure
7293 % <#fig:commented_line>.
7294 %
7295 % \end{markdown}
7296 % \begin{macrocode}
7297 parsers.commented_line_letter = parsers.linechar
7298                               + parsers.newline
7299                               - parsers.backslash
7300                               - parsers.percent
7301 parsers.commented_line = Cg(Cc(""), "backslashes")
7302   * ((#(parsers.commented_line_letter
7303     - parsers.newline)
7304     * Cb("backslashes")
7305     * Cs(parsers.commented_line_letter
7306     - parsers.newline)^1 -- initial
7307     * Cg(Cc(""), "backslashes"))
7308   + #(parsers.backslash * parsers.backslash)
7309   * Cg((parsers.backslash -- even backslash
7310     * parsers.backslash)^1, "backslashes")
7311   + (parsers.backslash
7312     * (#parsers.percent
7313     * Cb("backslashes")
7314     / function(backslashes)
7315     return string.rep("\\", #backslashes / 2)
7316     end
7317     * C(parsers.percent)
7318     + #parsers.commented_line_letter
7319     * Cb("backslashes")
7320     * Cc("\\")
7321     * C(parsers.commented_line_letter))
7322     * Cg(Cc(""), "backslashes"))^0
7323   * (#parsers.percent
7324     * Cb("backslashes")
7325     / function(backslashes)
7326     return string.rep("\\", #backslashes / 2)
7327     end
7328     * ((parsers.percent -- comment
7329       * parsers.line

```

```

7330         * #parsers.blankline) -- blank line
7331         / "\n"
7332         + parsers.percent -- comment
7333         * parsers.line
7334         * parsers.optionalspace) -- leading spaces
7335         + #(parsers.newline)
7336         * Cb("backslashes")
7337         * C(parsers.newline))
7338
7339 parsers.chunk = parsers.line * (parsers.optionallyindentedline
7340                             - parsers.blankline)^0
7341
7342 parsers.attribute_key_char = parsers.alphanumeric + S("-_:.")
7343 parsers.attribute_raw_char = parsers.alphanumeric + S("-_")
7344 parsers.attribute_key = (parsers.attribute_key_char
7345                       - parsers.dash - parsers.digit)
7346                       * parsers.attribute_key_char^0
7347 parsers.attribute_value = ( (parsers.dquote / "\"")
7348                          * (parsers.anyescaped - parsers.dquote)^0
7349                          * (parsers.dquote / "\""))
7350                          + ( (parsers.squote / "\"")
7351                          * (parsers.anyescaped - parsers.squote)^0
7352                          * (parsers.squote / "\""))
7353                          + ( parsers.anyescaped
7354                          - parsers.dquote
7355                          - parsers.rbrace
7356                          - parsers.space)^0
7357 parsers.attribute_identifer = parsers.attribute_key_char^1
7358 parsers.attribute_classname = parsers.letter
7359                             * parsers.attribute_key_char^0
7360 parsers.attribute_raw = parsers.attribute_raw_char^1
7361
7362 parsers.attribute = (parsers.dash * Cc(".unnumbered"))
7363                   + C( parsers.hash
7364                       * parsers.attribute_identifer)
7365                   + C( parsers.period
7366                       * parsers.attribute_classname)
7367                   + Cs( parsers.attribute_key
7368                       * parsers.optionalspace
7369                       * parsers.equal
7370                       * parsers.optionalspace
7371                       * parsers.attribute_value)
7372 parsers.attributes = parsers.lbrace
7373                   * parsers.optionalspace
7374                   * parsers.attribute
7375                   * (parsers.spacechar^1
7376                   * parsers.attribute)^0

```

```

7377             * parsers.optionalspace
7378             * parsers.rbrace
7379
7380 parsers.raw_attribute = parsers.lbrace
7381             * parsers.optionalspace
7382             * parsers.equal
7383             * C(parsers.attribute_raw)
7384             * parsers.optionalspace
7385             * parsers.rbrace
7386
7387 -- block followed by 0 or more optionally
7388 -- indented blocks with first line indented.
7389 parsers.indented_blocks = function(bl)
7390   return Cs( bl
7391     * ( parsers.blankline^1
7392       * parsers.indent
7393       * -parsers.blankline
7394       * bl)^0
7395     * (parsers.blankline^1 + parsers.eof) )
7396 end

```

### 3.1.5.2 Parsers Used for HTML Entities

```

7397 local function repeat_between(pattern, min, max)
7398   return -pattern^(max + 1) * pattern^min
7399 end
7400
7401 parsers.hexentity = parsers.ampersand * parsers.hash * C(S("Xx"))
7402                   * C(repeat_between(parsers.hexdigit, 1, 6))
7403                   * parsers.semicolon
7404 parsers.decentity = parsers.ampersand * parsers.hash
7405                   * C(repeat_between(parsers.digit, 1, 7))
7406                   * parsers.semicolon
7407 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
7408                   * parsers.semicolon
7409
7410 parsers.html_entities
7411   = parsers.hexentity / entities.hex_entity_with_x_char
7412   + parsers.decentity / entities.dec_entity
7413   + parsers.tagentity / entities.char_entity

```

### 3.1.5.3 Parsers Used for Markdown Lists

```

7414 parsers.bullet = function(bullet_char, interrupting)
7415   local allowed_end
7416   if interrupting then
7417     allowed_end = C(parsers.spacechar^1) * #parsers.linechar
7418   else

```

```

7419     allowed_end = C(parsers.spacechar^1)
7420                 + #(parsers.newline + parsers.eof)
7421   end
7422   return parsers.check_trail
7423         * Ct(C(bullet_char) * Cc(""))
7424         * allowed_end
7425 end
7426
7427 local function tickbox(interior)
7428   return parsers.optionalspace * parsers.lbracket
7429         * interior * parsers.rbracket * parsers.spacechar^1
7430 end
7431
7432 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
7433 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
7434 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
7435

```

### 3.1.5.4 Parsers Used for Markdown Code Spans

```

7436 parsers.openticks = Cg(parsers.backtick^1, "ticks")
7437
7438 local function captures_equal_length(_,i,a,b)
7439   return #a == #b and i
7440 end
7441
7442 parsers.closeticks = Cmt(C(parsers.backtick^1)
7443                         * Cb("ticks"), captures_equal_length)
7444
7445 parsers.intickschar = (parsers.any - S("\n\r`"))
7446                     + V("NoSoftLineBreakEndline")
7447                     + (parsers.backtick^1 - parsers.closeticks)
7448
7449 local function process_inticks(s)
7450   s = s:gsub("\n", " ")
7451   s = s:gsub("^ (.*) $", "%1")
7452   return s
7453 end
7454
7455 parsers.inticks = parsers.openticks
7456                 * C(parsers.space^0)
7457                 * parsers.closeticks
7458                 + parsers.openticks
7459                 * Cs(parsers.intickschar^0) / process_inticks)
7460                 * parsers.closeticks
7461

```

### 3.1.5.5 Parsers Used for HTML

```
7462 -- case-insensitive match (we assume s is lowercase)
7463 -- must be single byte encoding
7464 parsers.keyword_exact = function(s)
7465     local parser = P(0)
7466     for i=1,#s do
7467         local c = s:sub(i,i)
7468         local m = c .. upper(c)
7469         parser = parser * S(m)
7470     end
7471     return parser
7472 end
7473
7474 parsers.special_block_keyword =
7475     parsers.keyword_exact("pre") +
7476     parsers.keyword_exact("script") +
7477     parsers.keyword_exact("style") +
7478     parsers.keyword_exact("textarea")
7479
7480 parsers.block_keyword =
7481     parsers.keyword_exact("address") +
7482     parsers.keyword_exact("article") +
7483     parsers.keyword_exact("aside") +
7484     parsers.keyword_exact("base") +
7485     parsers.keyword_exact("basefont") +
7486     parsers.keyword_exact("blockquote") +
7487     parsers.keyword_exact("body") +
7488     parsers.keyword_exact("caption") +
7489     parsers.keyword_exact("center") +
7490     parsers.keyword_exact("col") +
7491     parsers.keyword_exact("colgroup") +
7492     parsers.keyword_exact("dd") +
7493     parsers.keyword_exact("details") +
7494     parsers.keyword_exact("dialog") +
7495     parsers.keyword_exact("dir") +
7496     parsers.keyword_exact("div") +
7497     parsers.keyword_exact("dl") +
7498     parsers.keyword_exact("dt") +
7499     parsers.keyword_exact("fieldset") +
7500     parsers.keyword_exact("figcaption") +
7501     parsers.keyword_exact("figure") +
7502     parsers.keyword_exact("footer") +
7503     parsers.keyword_exact("form") +
7504     parsers.keyword_exact("frame") +
7505     parsers.keyword_exact("frameset") +
7506     parsers.keyword_exact("h1") +
7507     parsers.keyword_exact("h2") +
```

```

7508     parsers.keyword_exact("h3") +
7509     parsers.keyword_exact("h4") +
7510     parsers.keyword_exact("h5") +
7511     parsers.keyword_exact("h6") +
7512     parsers.keyword_exact("head") +
7513     parsers.keyword_exact("header") +
7514     parsers.keyword_exact("hr") +
7515     parsers.keyword_exact("html") +
7516     parsers.keyword_exact("iframe") +
7517     parsers.keyword_exact("legend") +
7518     parsers.keyword_exact("li") +
7519     parsers.keyword_exact("link") +
7520     parsers.keyword_exact("main") +
7521     parsers.keyword_exact("menu") +
7522     parsers.keyword_exact("menuitem") +
7523     parsers.keyword_exact("nav") +
7524     parsers.keyword_exact("noframes") +
7525     parsers.keyword_exact("ol") +
7526     parsers.keyword_exact("optgroup") +
7527     parsers.keyword_exact("option") +
7528     parsers.keyword_exact("p") +
7529     parsers.keyword_exact("param") +
7530     parsers.keyword_exact("section") +
7531     parsers.keyword_exact("source") +
7532     parsers.keyword_exact("summary") +
7533     parsers.keyword_exact("table") +
7534     parsers.keyword_exact("tbody") +
7535     parsers.keyword_exact("td") +
7536     parsers.keyword_exact("tfoot") +
7537     parsers.keyword_exact("th") +
7538     parsers.keyword_exact("thead") +
7539     parsers.keyword_exact("title") +
7540     parsers.keyword_exact("tr") +
7541     parsers.keyword_exact("track") +
7542     parsers.keyword_exact("ul")
7543
7544 -- end conditions
7545 parsers.html_blankline_end_condition
7546 = parsers.linechar^0
7547 * ( parsers.newline
7548     * (parsers.check_minimal_blank_indent_and_any_trail
7549         * #parsers.blankline
7550         + parsers.check_minimal_indent_and_any_trail)
7551     * parsers.linechar^1)^0
7552 * (parsers.newline^-1 / "")
7553
7554 local function remove_trailing_blank_lines(s)

```



```

7555 return s:gsub("[\n\r]+%s*$", "")
7556 end
7557
7558 parsers.html_until_end = function(end_marker)
7559 return Cs(Cs((parsers.newline
7560     * (parsers.check_minimal_blank_indent_and_any_trail
7561     * #parsers.blankline
7562     + parsers.check_minimal_indent_and_any_trail)
7563     + parsers.linechar - end_marker)^0
7564     * parsers.linechar^0 * parsers.newline^-1)
7565     / remove_trailing_blank_lines)
7566 end
7567
7568 -- attributes
7569 parsers.html_attribute_spacing = parsers.optionalspace
7570     * V("NoSoftLineBreakEndline")
7571     * parsers.optionalspace
7572     + parsers.spacechar^1
7573
7574 parsers.html_attribute_name = ( parsers.letter
7575     + parsers.colon
7576     + parsers.underscore)
7577     * ( parsers.alphanumeric
7578     + parsers.colon
7579     + parsers.underscore
7580     + parsers.period
7581     + parsers.dash)^0
7582
7583 parsers.html_attribute_value = parsers.squote
7584     * (parsers.linechar - parsers.squote)^0
7585     * parsers.squote
7586     + parsers.dquote
7587     * (parsers.linechar - parsers.dquote)^0
7588     * parsers.dquote
7589     + ( parsers.any
7590     - parsers.spacechar
7591     - parsers.newline
7592     - parsers.dquote
7593     - parsers.squote
7594     - parsers.backtick
7595     - parsers.equal
7596     - parsers.less
7597     - parsers.more)^1
7598
7599 parsers.html_inline_attribute_value = parsers.squote
7600     * (V("NoSoftLineBreakEndline")
7601     + parsers.any

```

```

7602         - parsers.blankline^2
7603         - parsers.squote)^0
7604     * parsers.squote
7605     + parsers.dquote
7606     * (V("NoSoftLineBreakEndline")
7607         + parsers.any
7608         - parsers.blankline^2
7609         - parsers.dquote)^0
7610     * parsers.dquote
7611     + (parsers.any
7612         - parsers.spacechar
7613         - parsers.newline
7614         - parsers.dquote
7615         - parsers.squote
7616         - parsers.backtick
7617         - parsers.equal
7618         - parsers.less
7619         - parsers.more)^1
7620
7621 parsers.html_attribute_value_specification
7622 = parsers.optionalspace
7623 * parsers.equal
7624 * parsers.optionalspace
7625 * parsers.html_attribute_value
7626
7627 parsers.html_spnl = parsers.optionalspace
7628                 * (V("NoSoftLineBreakEndline")
7629                 * parsers.optionalspace)^-1
7630
7631 parsers.html_inline_attribute_value_specification
7632 = parsers.html_spnl
7633 * parsers.equal
7634 * parsers.html_spnl
7635 * parsers.html_inline_attribute_value
7636
7637 parsers.html_attribute
7638 = parsers.html_attribute_spacing
7639 * parsers.html_attribute_name
7640 * parsers.html_inline_attribute_value_specification^-1
7641
7642 parsers.html_non_newline_attribute
7643 = parsers.spacechar^1
7644 * parsers.html_attribute_name
7645 * parsers.html_attribute_value_specification^-1
7646
7647 parsers.nested_breaking_blank = parsers.newline
7648                             * parsers.check_minimal_blank_indent

```

```

7649             * parsers.blankline
7650
7651 parsers.html_comment_start = P("<!--")
7652
7653 parsers.html_comment_end = P("-->")
7654
7655 parsers.html_comment
7656     = Cs( parsers.html_comment_start
7657           * parsers.html_until_end(parsers.html_comment_end))
7658
7659 parsers.html_inline_comment = (parsers.html_comment_start / "")
7660                               * -P(">") * -P("->")
7661                               * Cs(( V("NoSoftLineBreakEndline")
7662                                     + parsers.any
7663                                     - parsers.nested_breaking_blank
7664                                     - parsers.html_comment_end)^0)
7665                               * (parsers.html_comment_end / "")
7666
7667 parsers.html_cdatasection_start = P("<![CDATA[")
7668
7669 parsers.html_cdatasection_end = P("]]>")
7670
7671 parsers.html_cdatasection
7672     = Cs( parsers.html_cdatasection_start
7673           * parsers.html_until_end(parsers.html_cdatasection_end))
7674
7675 parsers.html_inline_cdatasection
7676     = parsers.html_cdatasection_start
7677       * Cs(V("NoSoftLineBreakEndline") + parsers.any
7678           - parsers.nested_breaking_blank - parsers.html_cdatasection_end)^0
7679       * parsers.html_cdatasection_end
7680
7681 parsers.html_declaration_start = P("<!") * parsers.letter
7682
7683 parsers.html_declaration_end = P(">")
7684
7685 parsers.html_declaration
7686     = Cs( parsers.html_declaration_start
7687           * parsers.html_until_end(parsers.html_declaration_end))
7688
7689 parsers.html_inline_declaration
7690     = parsers.html_declaration_start
7691       * Cs(V("NoSoftLineBreakEndline") + parsers.any
7692           - parsers.nested_breaking_blank - parsers.html_declaration_end)^0
7693       * parsers.html_declaration_end
7694
7695 parsers.html_instruction_start = P("<?")

```

```

7696
7697 parsers.html_instruction_end = P(">")
7698
7699 parsers.html_instruction
7700     = Cs( parsers.html_instruction_start
7701           * parsers.html_until_end(parsers.html_instruction_end))
7702
7703 parsers.html_inline_instruction = parsers.html_instruction_start
7704                                 * Cs( V("NoSoftLineBreakEndline")
7705                                       + parsers.any
7706                                       - parsers.nested_breaking_blank
7707                                       - parsers.html_instruction_end)^0
7708                                 * parsers.html_instruction_end
7709
7710 parsers.html_blankline = parsers.newline
7711                       * parsers.optionalspace
7712                       * parsers.newline
7713
7714 parsers.html_tag_start = parsers.less
7715
7716 parsers.html_tag_closing_start = parsers.less
7717                               * parsers.slash
7718
7719 parsers.html_tag_end = parsers.html_spnl
7720                     * parsers.more
7721
7722 parsers.html_empty_tag_end = parsers.html_spnl
7723                           * parsers.slash
7724                           * parsers.more
7725
7726 -- opening tags
7727 parsers.html_any_open_inline_tag = parsers.html_tag_start
7728                                 * parsers.keyword
7729                                 * parsers.html_attribute^0
7730                                 * parsers.html_tag_end
7731
7732 parsers.html_any_open_tag = parsers.html_tag_start
7733                          * parsers.keyword
7734                          * parsers.html_non_newline_attribute^0
7735                          * parsers.html_tag_end
7736
7737 parsers.html_open_tag = parsers.html_tag_start
7738                      * parsers.block_keyword
7739                      * parsers.html_attribute^0
7740                      * parsers.html_tag_end
7741
7742 parsers.html_open_special_tag = parsers.html_tag_start

```

```

7743             * parsers.special_block_keyword
7744             * parsers.html_attribute^0
7745             * parsers.html_tag_end
7746
7747 -- incomplete tags
7748 parsers.incomplete_tag_following = parsers.spacechar
7749                                 + parsers.more
7750                                 + parsers.slash * parsers.more
7751                                 + #(parsers.newline + parsers.eof)
7752
7753 parsers.incomplete_special_tag_following = parsers.spacechar
7754                                         + parsers.more
7755                                         + #( parsers.newline
7756                                           + parsers.eof)
7757
7758 parsers.html_incomplete_open_tag = parsers.html_tag_start
7759                                 * parsers.block_keyword
7760                                 * parsers.incomplete_tag_following
7761
7762 parsers.html_incomplete_open_special_tag
7763 = parsers.html_tag_start
7764 * parsers.special_block_keyword
7765 * parsers.incomplete_special_tag_following
7766
7767 parsers.html_incomplete_close_tag = parsers.html_tag_closing_start
7768                                 * parsers.block_keyword
7769                                 * parsers.incomplete_tag_following
7770
7771 parsers.html_incomplete_close_special_tag
7772 = parsers.html_tag_closing_start
7773 * parsers.special_block_keyword
7774 * parsers.incomplete_tag_following
7775
7776 -- closing tags
7777 parsers.html_close_tag = parsers.html_tag_closing_start
7778                       * parsers.block_keyword
7779                       * parsers.html_tag_end
7780
7781 parsers.html_any_close_tag = parsers.html_tag_closing_start
7782                           * parsers.keyword
7783                           * parsers.html_tag_end
7784
7785 parsers.html_close_special_tag = parsers.html_tag_closing_start
7786                               * parsers.special_block_keyword
7787                               * parsers.html_tag_end
7788
7789 -- empty tags

```

```

7790 parsers.html_any_empty_inline_tag = parsers.html_tag_start
7791                                     * parsers.keyword
7792                                     * parsers.html_attribute^0
7793                                     * parsers.html_empty_tag_end
7794
7795 parsers.html_any_empty_tag = parsers.html_tag_start
7796                               * parsers.keyword
7797                               * parsers.html_non_newline_attribute^0
7798                               * parsers.optionalspace
7799                               * parsers.slash
7800                               * parsers.more
7801
7802 parsers.html_empty_tag = parsers.html_tag_start
7803                          * parsers.block_keyword
7804                          * parsers.html_attribute^0
7805                          * parsers.html_empty_tag_end
7806
7807 parsers.html_empty_special_tag = parsers.html_tag_start
7808                                 * parsers.special_block_keyword
7809                                 * parsers.html_attribute^0
7810                                 * parsers.html_empty_tag_end
7811
7812 parsers.html_incomplete_blocks
7813   = parsers.html_incomplete_open_tag
7814     + parsers.html_incomplete_open_special_tag
7815     + parsers.html_incomplete_close_tag
7816
7817 -- parse special html blocks
7818 parsers.html_blankline_ending_special_block_opening
7819   = ( parsers.html_close_special_tag
7820       + parsers.html_empty_special_tag)
7821     * #( parsers.optionalspace
7822          * (parsers.newline + parsers.eof))
7823
7824 parsers.html_blankline_ending_special_block
7825   = parsers.html_blankline_ending_special_block_opening
7826     * parsers.html_blankline_end_condition
7827
7828 parsers.html_special_block_opening
7829   = parsers.html_incomplete_open_special_tag
7830     - parsers.html_empty_special_tag
7831
7832 parsers.html_closing_special_block
7833   = parsers.html_special_block_opening
7834     * parsers.html_until_end(parsers.html_close_special_tag)
7835
7836 parsers.html_special_block

```

```

7837 = parsers.html_blankline_ending_special_block
7838 + parsers.html_closing_special_block
7839
7840 -- parse html blocks
7841 parsers.html_block_opening = parsers.html_incomplete_open_tag
7842                             + parsers.html_incomplete_close_tag
7843
7844 parsers.html_block = parsers.html_block_opening
7845                     * parsers.html_blankline_end_condition
7846
7847 -- parse any html blocks
7848 parsers.html_any_block_opening
7849 = ( parsers.html_any_open_tag
7850     + parsers.html_any_close_tag
7851     + parsers.html_any_empty_tag)
7852 * #(parsers.optionalspace * (parsers.newline + parsers.eof))
7853
7854 parsers.html_any_block = parsers.html_any_block_opening
7855                         * parsers.html_blankline_end_condition
7856
7857 parsers.html_inline_comment_full = parsers.html_comment_start
7858                                  * -P(">") * -P("->")
7859                                  * Cs(( V("NoSoftLineBreakEndline")
7860                                          + parsers.any - P("--")
7861                                          - parsers.nested_breaking_blank
7862                                          - parsers.html_comment_end)^0)
7863                                  * parsers.html_comment_end
7864
7865 parsers.html_inline_tags = parsers.html_inline_comment_full
7866                          + parsers.html_any_empty_inline_tag
7867                          + parsers.html_inline_instruction
7868                          + parsers.html_inline_cdatasection
7869                          + parsers.html_inline_declaration
7870                          + parsers.html_any_open_inline_tag
7871                          + parsers.html_any_close_tag
7872

```

### 3.1.5.6 Parsers Used for Markdown Tags and Links

```

7873 parsers.urlchar = parsers.anyescaped
7874                 - parsers.newline
7875                 - parsers.more
7876
7877 parsers.auto_link_scheme_part = parsers.alphanumeric
7878                               + parsers.plus
7879                               + parsers.period
7880                               + parsers.dash

```

```

7881
7882 parsers.auto_link_scheme = parsers.letter
7883                             * parsers.auto_link_scheme_part
7884                             * parsers.auto_link_scheme_part^-30
7885
7886 parsers.absolute_uri = parsers.auto_link_scheme * parsers.colon
7887                       * ( parsers.any - parsers.spacing
7888                           - parsers.less - parsers.more)^0
7889
7890 parsers.printable_characters = S(" !#$%&'*/=?^_`{|}~-")
7891
7892 parsers.email_address_local_part_char = parsers.alphanumeric
7893                                         + parsers.printable_characters
7894
7895 parsers.email_address_local_part
7896   = parsers.email_address_local_part_char^1
7897
7898 parsers.email_address_dns_label = parsers.alphanumeric
7899                                 * ( parsers.alphanumeric
7900                                   + parsers.dash)^-62
7901                                 * B(parsers.alphanumeric)
7902
7903 parsers.email_address_domain = parsers.email_address_dns_label
7904                               * ( parsers.period
7905                                 * parsers.email_address_dns_label)^0
7906
7907 parsers.email_address = parsers.email_address_local_part
7908                         * parsers.at
7909                         * parsers.email_address_domain
7910
7911 parsers.auto_link_url = parsers.less
7912                       * C(parsers.absolute_uri)
7913                       * parsers.more
7914
7915 parsers.auto_link_email = parsers.less
7916                          * C(parsers.email_address)
7917                          * parsers.more
7918
7919 parsers.auto_link_relative_reference = parsers.less
7920                                       * C(parsers.urlchar^1)
7921                                       * parsers.more
7922
7923 parsers.autolink = parsers.auto_link_url
7924                  + parsers.auto_link_email
7925
7926 -- content in balanced brackets, parentheses, or quotes:
7927 parsers.bracketed = P{ parsers.lbracket

```



```

7928         * (( parsers.backslash / "" * parsers.rbracket
7929           + parsers.any - (parsers.lbracket
7930                           + parsers.rbracket
7931                           + parsers.blankline^2)
7932           ) + V(1))^0
7933         * parsers.rbracket }
7934
7935 parsers.inparens = P{ parsers.lparent
7936                   * ((parsers.anyescaped - (parsers.lparent
7937                                             + parsers.rparent
7938                                             + parsers.blankline^2)
7939                   ) + V(1))^0
7940                   * parsers.rparent }
7941
7942 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
7943                   * ((parsers.anyescaped - (parsers.squote
7944                                             + parsers.blankline^2)
7945                   ) + V(1))^0
7946                   * parsers.squote }
7947
7948 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
7949                   * ((parsers.anyescaped - (parsers.dquote
7950                                             + parsers.blankline^2)
7951                   ) + V(1))^0
7952                   * parsers.dquote }
7953
7954 parsers.link_text = parsers.lbracket
7955                   * Cs((parsers.alphanumeric^1
7956                       + parsers.bracketed
7957                       + parsers.inticks
7958                       + parsers.autolink
7959                       + V("InlineHtml")
7960                       + ( parsers.backslash * parsers.backslash)
7961                       + ( parsers.backslash
7962                           * ( parsers.lbracket
7963                               + parsers.rbracket)
7964                           + V("NoSoftLineBreakSpace")
7965                           + V("NoSoftLineBreakEndline")
7966                           + (parsers.any
7967                               - ( parsers.newline
7968                                   + parsers.lbracket
7969                                   + parsers.rbracket
7970                                   + parsers.blankline^2))))))^0)
7971                   * parsers.rbracket
7972
7973 parsers.link_label_body = -(parsers.sp * parsers.rbracket)
7974                   * #( ( parsers.any

```

```

7975         - parsers.rbracket)^-999
7976     * parsers.rbracket)
7977     * Cs((parsers.alphanumeric^1
7978         + parsers.inticks
7979         + parsers.autolink
7980         + V("InlineHtml")
7981         + ( parsers.backslash * parsers.backslash)
7982         + ( parsers.backslash
7983           * ( parsers.lbracket
7984             + parsers.rbracket)
7985           + V("NoSoftLineBreakSpace")
7986           + V("NoSoftLineBreakEndline")
7987           + (parsers.any
7988             - ( parsers.newline
7989               + parsers.lbracket
7990               + parsers.rbracket
7991               + parsers.blankline^2))))^1)
7992
7993 parsers.link_label = parsers.lbracket
7994                   * parsers.link_label_body
7995                   * parsers.rbracket
7996
7997 parsers.inparens_url = P{ parsers.lparent
7998                       * ((parsers.anyescaped - (parsers.lparent
7999                                                                 + parsers.rparent
8000                                                                 + parsers.spacing)
8001                          ) + V(1))^0
8002                       * parsers.rparent }
8003
8004 -- url for markdown links, allowing nested brackets:
8005 parsers.url         = parsers.less * Cs((parsers.anyescaped
8006                                         - parsers.newline
8007                                         - parsers.less
8008                                         - parsers.more)^0)
8009                                         * parsers.more
8010 + -parsers.less
8011 * Cs((parsers.inparens_url + (parsers.anyescaped
8012                               - parsers.spacing
8013                               - parsers.lparent
8014                               - parsers.rparent))^1)
8015
8016 -- quoted text:
8017 parsers.title_s     = parsers.squote
8018                   * Cs((parsers.html_entities
8019                         + V("NoSoftLineBreakSpace")
8020                         + V("NoSoftLineBreakEndline")
8021                         + ( parsers.anyescaped

```

```

8022         - parsers.newline
8023         - parsers.squote
8024         - parsers.blankline^2))^0)
8025     * parsers.squote
8026
8027 parsers.title_d = parsers.dquote
8028     * Cs((parsers.html_entities
8029         + V("NoSoftLineBreakSpace")
8030         + V("NoSoftLineBreakEndline")
8031         + ( parsers.anyescaped
8032           - parsers.newline
8033           - parsers.dquote
8034           - parsers.blankline^2))^0)
8035     * parsers.dquote
8036
8037 parsers.title_p = parsers.lparent
8038     * Cs((parsers.html_entities
8039         + V("NoSoftLineBreakSpace")
8040         + V("NoSoftLineBreakEndline")
8041         + ( parsers.anyescaped
8042           - parsers.newline
8043           - parsers.lparent
8044           - parsers.rparent
8045           - parsers.blankline^2))^0)
8046     * parsers.rparent
8047
8048 parsers.title
8049     = parsers.title_d + parsers.title_s + parsers.title_p
8050
8051 parsers.optionaltitle
8052     = parsers.spnlc * parsers.title * parsers.spacechar^0 + Cc("")
8053

```

### 3.1.5.7 Helpers for Links and Link Reference Definitions

```

8054 -- parse a reference definition: [foo]: /bar "title"
8055 parsers.define_reference_parser = (parsers.check_trail / "")
8056     * parsers.link_label * parsers.colon
8057     * parsers.spnlc * parsers.url
8058     * ( parsers.spnlc_sep * parsers.title
8059       * parsers.only_blank
8060       + Cc("") * parsers.only_blank)

```

### 3.1.5.8 Inline Elements

```

8061 parsers.Inline = V("Inline")
8062
8063 -- parse many p between starter and ender

```

```

8064 parsers.between = function(p, starter, ender)
8065   local ender2 = B(parsers.nonspacechar) * ender
8066   return ( starter
8067           * #parsers.nonspacechar
8068           * Ct(p * (p - ender2)^0)
8069           * ender2)
8070 end
8071

```

### 3.1.5.9 Block Elements

```

8072 parsers.lineof = function(c)
8073   return ( parsers.check_trail_no_rem
8074           * (P(c) * parsers.optionalspace)^3
8075           * (parsers.newline + parsers.eof))
8076 end
8077
8078 parsers.thematic_break_lines = parsers.lineof(parsers.asterisk)
8079                               + parsers.lineof(parsers.dash)
8080                               + parsers.lineof(parsers.underscore)

```

### 3.1.5.10 Headings

```

8081 -- parse Atx heading start and return level
8082 parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
8083                       * -parsers.hash / length
8084
8085 -- parse setext header ending and return level
8086 parsers.heading_level
8087   = parsers.nonindentspace * parsers.equal^1
8088   * parsers.optionalspace * #parsers.newline * Cc(1)
8089   + parsers.nonindentspace * parsers.dash^1
8090   * parsers.optionalspace * #parsers.newline * Cc(2)
8091
8092 local function strip_atx_end(s)
8093   return s:gsub("%s+##%s*\n$", "")
8094 end
8095
8096 parsers.atx_heading = parsers.check_trail_no_rem
8097                     * Cg(parsers.heading_start, "level")
8098                     * (C( parsers.optionalspace
8099                           * parsers.hash^0
8100                           * parsers.optionalspace
8101                           * parsers.newline)
8102                       + parsers.spacechar^1
8103                       * C(parsers.line))

```

### 3.1.6 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.

The `reader.new` method creates and returns a new T<sub>E</sub>X reader object associated with the Lua interface options (see Section 2.1.3) `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`.

```
8104 M.reader = {}
8105 function M.reader.new(writer, options)
8106   local self = {}
```

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

```
8107   self.writer = writer
8108   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.

```
8109   self.parsers = {}
8110   (function(parsers)
8111     setmetatable(self.parsers, {
8112       __index = function (_, key)
8113         return parsers[key]
8114       end
8115     })
8116   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.

```
8117   local parsers = self.parsers
```

#### 3.1.6.1 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.

```
8118   function self.normalize_tag(tag)
8119     tag = util.rope_to_string(tag)
8120     tag = tag:gsub("[ \n\r\t]+", " ")
8121     tag = tag:gsub("^ ", ""):gsub(" $", "")
8122     tag = uni_algos.case.casefold(tag, true, false)
8123     return tag
```

```
8124 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.

```
8125 local function iterlines(s, f)
8126     local rope = lpeg.match(Ct((parsers.line / f)^1), s)
8127     return util.rope_to_string(rope)
8128 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.

```
8129 if options.preserveTabs then
8130     self.expandtabs = function(s) return s end
8131 else
8132     self.expandtabs = function(s)
8133         if s:find("\t") then
8134             return iterlines(s, util.expand_tabs_in_line)
8135         else
8136             return s
8137         end
8138     end
8139 end
```

### 3.1.6.2 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.

```
8140 self.parser_functions = {}
8141 self.create_parser = function(name, grammar, toplevel)
8142     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.

```
8143         if toplevel and options.stripIndent then
8144             local min_prefix_length, min_prefix = nil, ''
8145             str = iterlines(str, function(line)
8146                 if lpeg.match(parsers.nonemptyline, line) == nil then
8147                     return line
8148                 end
8149                 line = util.expand_tabs_in_line(line)
8150                 local prefix = lpeg.match(C(parsers.optionalspace), line)
```

```

8151         local prefix_length = #prefix
8152         local is_shorter = min_prefix_length == nil
8153         if not is_shorter then
8154             is_shorter = prefix_length < min_prefix_length
8155         end
8156         if is_shorter then
8157             min_prefix_length, min_prefix = prefix_length, prefix
8158         end
8159         return line
8160     end)
8161     str = str:gsub('^' .. min_prefix, '')
8162 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.

```

8163     if toplevel and (options.texComments or options.hybrid) then
8164         str = lpeg.match(Ct(parsers.commented_line^1), str)
8165         str = util.ropo_to_string(str)
8166     end
8167     local res = lpeg.match(grammar(), str)
8168     if res == nil then
8169         return writer.error(format("%s failed on:\n%s",
8170                                   name, str:sub(1,20)))
8171     else
8172         return res
8173     end
8174 end
8175 end
8176
8177 self.create_parser("parse_blocks",
8178                   function()
8179                       return parsers.blocks
8180                   end, true)
8181
8182 self.create_parser("parse_blocks_nested",
8183                   function()
8184                       return parsers.blocks_nested
8185                   end, false)
8186
8187 self.create_parser("parse_inlines",
8188                   function()
8189                       return parsers.inlines
8190                   end, false)
8191
8192 self.create_parser("parse_inlines_no_inline_note",
8193                   function()

```

```

8194         return parsers.inlines_no_inline_note
8195         end, false)
8196
8197     self.create_parser("parse_inlines_no_html",
8198         function()
8199             return parsers.inlines_no_html
8200         end, false)
8201
8202     self.create_parser("parse_inlines_nbsp",
8203         function()
8204             return parsers.inlines_nbsp
8205         end, false)
8206     self.create_parser("parse_inlines_no_link_or_emphasis",
8207         function()
8208             return parsers.inlines_no_link_or_emphasis
8209         end, false)

```

### 3.1.6.3 Parsers Used for Indentation (local)

The following patterns represent basic building blocks of indented content.

```

8210     parsers.minimally_indented_blankline
8211         = parsers.check_minimal_indent * (parsers.blankline / "")
8212
8213     parsers.minimally_indented_block
8214         = parsers.check_minimal_indent * V("Block")
8215
8216     parsers.minimally_indented_block_or_paragraph
8217         = parsers.check_minimal_indent * V("BlockOrParagraph")
8218
8219     parsers.minimally_indented_paragraph
8220         = parsers.check_minimal_indent * V("Paragraph")
8221
8222     parsers.minimally_indented_plain
8223         = parsers.check_minimal_indent * V("Plain")
8224
8225     parsers.minimally_indented_par_or_plain
8226         = parsers.minimally_indented_paragraph
8227         + parsers.minimally_indented_plain
8228
8229     parsers.minimally_indented_par_or_plain_no_blank
8230         = parsers.minimally_indented_par_or_plain
8231         - parsers.minimally_indented_blankline
8232
8233     parsers.minimally_indented_ref
8234         = parsers.check_minimal_indent * V("Reference")
8235
8236     parsers.minimally_indented_blank

```



```

8237     = parsers.check_minimal_indent * V("Blank")
8238
8239 parsers.conditionally_indented_blankline
8240     = parsers.check_minimal_blank_indent * (parsers.blankline / "")
8241
8242 parsers.minimally_indented_ref_or_block
8243     = parsers.minimally_indented_ref
8244     + parsers.minimally_indented_block
8245     - parsers.minimally_indented_blankline
8246
8247 parsers.minimally_indented_ref_or_block_or_par
8248     = parsers.minimally_indented_ref
8249     + parsers.minimally_indented_block_or_paragraph
8250     - parsers.minimally_indented_blankline
8251

```

The following pattern parses the properly indented content that follows the initial container start.

```

8252
8253 function parsers.separator_loop(separated_block, paragraph,
8254                                block_separator, paragraph_separator)
8255     return separated_block
8256         + block_separator
8257         * paragraph
8258         * separated_block
8259         + paragraph_separator
8260         * paragraph
8261 end
8262
8263 function parsers.create_loop_body_pair(separated_block, paragraph,
8264                                       block_separator,
8265                                       paragraph_separator)
8266     return {
8267         block = parsers.separator_loop(separated_block, paragraph,
8268                                       block_separator, block_separator),
8269         par = parsers.separator_loop(separated_block, paragraph,
8270                                     block_separator, paragraph_separator)
8271     }
8272 end
8273
8274 parsers.block_sep_group = function(blank)
8275     return blank^0 * parsers.eof
8276         + ( blank^2 / writer.paragraphsep
8277           + blank^0 / writer.interblocksep
8278           )
8279 end
8280

```

```

8281 parsers.par_sep_group = function(blank)
8282     return blank^0 * parsers.eof
8283         + blank^0 / writer.paragraphsep
8284 end
8285
8286 parsers.sep_group_no_output = function(blank)
8287     return blank^0 * parsers.eof
8288         + blank^0
8289 end
8290
8291 parsers.content_blank = parsers.minimally_indented_blankline
8292
8293 parsers.ref_or_block_separated
8294     = parsers.sep_group_no_output(parsers.content_blank)
8295     * ( parsers.minimally_indented_ref
8296         - parsers.content_blank)
8297     + parsers.block_sep_group(parsers.content_blank)
8298     * ( parsers.minimally_indented_block
8299         - parsers.content_blank)
8300
8301 parsers.loop_body_pair =
8302     parsers.create_loop_body_pair(
8303         parsers.ref_or_block_separated,
8304         parsers.minimally_indented_par_or_plain_no_blank,
8305         parsers.block_sep_group(parsers.content_blank),
8306         parsers.par_sep_group(parsers.content_blank))
8307
8308 parsers.content_loop = ( V("Block")
8309     * parsers.loop_body_pair.block^0
8310     + (V("Paragraph") + V("Plain"))
8311     * parsers.ref_or_block_separated
8312     * parsers.loop_body_pair.block^0
8313     + (V("Paragraph") + V("Plain"))
8314     * parsers.loop_body_pair.par^0
8315     * parsers.content_blank^0
8316
8317 parsers.indented_content = function()
8318     return Ct( (V("Reference") + (parsers.blankline / ""))
8319         * parsers.content_blank^0
8320         * parsers.check_minimal_indent
8321         * parsers.content_loop
8322         + (V("Reference") + (parsers.blankline / ""))
8323         * parsers.content_blank^0
8324         + parsers.content_loop)
8325 end
8326
8327 parsers.add_indent = function(pattern, name, breakable)

```

```

8328     return Cg(Cmt( Cb("indent_info")
8329                 * Ct(pattern)
8330                 * ( #parsers.linechar -- check if starter is blank
8331                   * Cc(false) + Cc(true))
8332                 * Cc(name)
8333                 * Cc(breakable),
8334                 process_starter_indent), "indent_info")
8335 end
8336

```

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

```

8337 if options.hashEnumerators then
8338     parsers.dig = parsers.digit + parsers.hash
8339 else
8340     parsers.dig = parsers.digit
8341 end
8342
8343 parsers.enumerator = function(delimiter_type, interrupting)
8344     local delimiter_range
8345     local allowed_end
8346     if interrupting then
8347         delimiter_range = P("1")
8348         allowed_end = C(parsers.spacechar^1) * #parsers.linechar
8349     else
8350         delimiter_range = parsers.dig * parsers.dig^-8
8351         allowed_end = C(parsers.spacechar^1)
8352                     + #(parsers.newline + parsers.eof)
8353     end
8354
8355     return parsers.check_trail
8356             * Ct(C(delimiter_range) * C(delimiter_type))
8357             * allowed_end
8358 end
8359
8360 parsers.starter = parsers.bullet(parsers.dash)
8361                 + parsers.bullet(parsers.asterisk)
8362                 + parsers.bullet(parsers.plus)
8363                 + parsers.enumerator(parsers.period)
8364                 + parsers.enumerator(parsers.rparent)
8365

```

### 3.1.6.5 Parsers Used for Blockquotes (local)

```

8366 parsers.blockquote_start
8367     = parsers.check_trail
8368     * C(parsers.more)
8369     * C(parsers.spacechar^0)

```

```

8370
8371 parsers.blockquote_body
8372   = parsers.add_indent(parsers.blockquote_start, "bq", true)
8373   * parsers.indented_content()
8374   * remove_indent("bq")
8375
8376 if not options.breakableBlockquotes then
8377   parsers.blockquote_body
8378     = parsers.add_indent(parsers.blockquote_start, "bq", false)
8379     * parsers.indented_content()
8380     * remove_indent("bq")
8381 end

```

### 3.1.6.6 Helpers for Emphasis and Strong Emphasis (local)

Parse the content of a table `content_part` with links, images and emphasis disabled.

```

8382 local function parse_content_part(content_part)
8383   local rope = util.rope_to_string(content_part)
8384   local parsed
8385     = self.parser_functions.parse_inlines_no_link_or_emphasis(rope)
8386   parsed.indent_info = nil
8387   return parsed
8388 end
8389

```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

8390 local collect_emphasis_content =
8391   function(t, opening_index, closing_index)
8392     local content = {}
8393
8394     local content_part = {}
8395     for i = opening_index, closing_index do
8396       local value = t[i]
8397
8398       if value.rendered ~= nil then
8399         content[#content + 1] = parse_content_part(content_part)
8400         content_part = {}
8401         content[#content + 1] = value.rendered
8402         value.rendered = nil
8403       else
8404         if value.type == "delimiter"
8405           and value.element == "emphasis" then
8406           if value.is_active then
8407             content_part[#content_part + 1]
8408               = string.rep(value.character, value.current_count)
8409           end

```

```

8410         else
8411             content_part[#content_part + 1] = value.content
8412         end
8413         value.content = ''
8414         value.is_active = false
8415     end
8416 end
8417
8418 if next(content_part) ~= nil then
8419     content[#content + 1] = parse_content_part(content_part)
8420 end
8421
8422 return content
8423 end
8424

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as emphasis.

```

8425 local function fill_emph(t, opening_index, closing_index)
8426     local content
8427     = collect_emphasis_content(t, opening_index + 1,
8428                               closing_index - 1)
8429     t[opening_index + 1].is_active = true
8430     t[opening_index + 1].rendered = writer.emphasis(content)
8431 end
8432

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as strong emphasis.

```

8433 local function fill_strong(t, opening_index, closing_index)
8434     local content
8435     = collect_emphasis_content(t, opening_index + 1,
8436                               closing_index - 1)
8437     t[opening_index + 1].is_active = true
8438     t[opening_index + 1].rendered = writer.strong(content)
8439 end
8440

```

Check whether the opening delimiter `opening_delimiter` and closing delimiter `closing_delimiter` break rule three together.

```

8441 local function breaks_three_rule(opening_delimiter, closing_delimiter)
8442     return ( opening_delimiter.is_closing
8443             or closing_delimiter.is_opening)
8444     and (( opening_delimiter.original_count
8445            + closing_delimiter.original_count) % 3 == 0)
8446     and ( opening_delimiter.original_count % 3 ~= 0
8447           or closing_delimiter.original_count % 3 ~= 0)
8448 end

```

8449

Look for the first potential emphasis opener in the delimiter table `t` in the range from `bottom_index` to `latest_index` that has the same character `character` as the closing delimiter `closing_delimiter`.

```
8450 local find_emphasis_opener = function(t, bottom_index, latest_index,
8451                                     character, closing_delimiter)
8452     for i = latest_index, bottom_index, -1 do
8453         local value = t[i]
8454         if value.is_active and
8455            value.is_opening and
8456            value.type == "delimiter" and
8457            value.element == "emphasis" and
8458            (value.character == character) and
8459            (value.current_count > 0) then
8460             if not breaks_three_rule(value, closing_delimiter) then
8461                 return i
8462             end
8463         end
8464     end
8465 end
8466
```

Iterate over the delimiters in the delimiter table `t`, producing emphasis or strong emphasis macros.

```
8467 local function process_emphasis(t, opening_index, closing_index)
8468     for i = opening_index, closing_index do
8469         local value = t[i]
8470         if value.type == "delimiter" and value.element == "emphasis" then
8471             local delimiter_length = string.len(value.content)
8472             value.character = string.sub(value.content, 1, 1)
8473             value.current_count = delimiter_length
8474             value.original_count = delimiter_length
8475         end
8476     end
8477
8478     local openers_bottom = {
8479         ['*'] = {
8480             [true] = {opening_index, opening_index, opening_index},
8481             [false] = {opening_index, opening_index, opening_index}
8482         },
8483         ['_'] = {
8484             [true] = {opening_index, opening_index, opening_index},
8485             [false] = {opening_index, opening_index, opening_index}
8486         }
8487     }
8488
8489     local current_position = opening_index
```

```

8490     local max_position = closing_index
8491
8492     while current_position <= max_position do
8493         local value = t[current_position]
8494
8495         if value.type ~= "delimiter" or
8496            value.element ~= "emphasis" or
8497            not value.is_active or
8498            not value.is_closing or
8499            (value.current_count <= 0) then
8500             current_position = current_position + 1
8501             goto continue
8502         end
8503
8504         local character = value.character
8505         local is_opening = value.is_opening
8506         local closing_length_modulo_three = value.original_count % 3
8507
8508         local current_openers_bottom
8509             = openers_bottom[character][is_opening]
8510               [closing_length_modulo_three + 1]
8511
8512         local opener_position
8513             = find_emphasis_opener(t, current_openers_bottom,
8514                                   current_position - 1, character, value)
8515
8516         if (opener_position == nil) then
8517             openers_bottom[character][is_opening]
8518                 [closing_length_modulo_three + 1]
8519                 = current_position
8520             current_position = current_position + 1
8521             goto continue
8522         end
8523
8524         local opening_delimiter = t[opener_position]
8525
8526         local current_opening_count = opening_delimiter.current_count
8527         local current_closing_count = t[current_position].current_count
8528
8529         if (current_opening_count >= 2)
8530            and (current_closing_count >= 2) then
8531             opening_delimiter.current_count = current_opening_count - 2
8532             t[current_position].current_count = current_closing_count - 2
8533             fill_strong(t, opener_position, current_position)
8534         else
8535             opening_delimiter.current_count = current_opening_count - 1
8536             t[current_position].current_count = current_closing_count - 1

```

```

8537         fill_emph(t, opener_position, current_position)
8538     end
8539
8540     ::continue::
8541 end
8542 end
8543
8544 local cont = lpeg.R("\128\191") -- continuation byte
8545

```

Match a UTF-8 character of byte length `n`.

```

8546 local function utf8_by_byte_count(n)
8547     if (n == 1) then
8548         return lpeg.R("\0\127")
8549     end
8550     if (n == 2) then
8551         return lpeg.R("\194\223") * cont
8552     end
8553     if (n == 3) then
8554         return lpeg.R("\224\239") * cont * cont
8555     end
8556     if (n == 4) then
8557         return lpeg.R("\240\244") * cont * cont * cont
8558     end
8559 end

```

Check if there is a character of a type `chartype` between the start position `start_pos` and end position `end_pos` in a string `s` relative to current index `i`.

```

8560 local function check_unicode_type(s, i, start_pos, end_pos, chartype)
8561     local c
8562     local char_length
8563     for pos = start_pos, end_pos, 1 do
8564         if (start_pos < 0) then
8565             char_length = -pos
8566         else
8567             char_length = pos + 1
8568         end
8569
8570         if (chartype == "punctuation") then
8571             if lpeg.match(parsers.punctuation[char_length], s, i+pos) then
8572                 return i
8573             end
8574         else
8575             c = lpeg.match({ C(utf8_by_byte_count(char_length)) }, s, i+pos)
8576             if (c ~= nil) and (unicode.utf8.match(c, chartype)) then
8577                 return i
8578             end
8579         end
8580     end

```



```

8580     end
8581 end
8582
8583 local function check_preceding_unicode_punctuation(s, i)
8584     return check_unicode_type(s, i, -4, -1, "punctuation")
8585 end
8586
8587 local function check_preceding_unicode_whitespace(s, i)
8588     return check_unicode_type(s, i, -4, -1, "%s")
8589 end
8590
8591 local function check_following_unicode_punctuation(s, i)
8592     return check_unicode_type(s, i, 0, 3, "punctuation")
8593 end
8594
8595 local function check_following_unicode_whitespace(s, i)
8596     return check_unicode_type(s, i, 0, 3, "%s")
8597 end
8598
8599 parsers.unicode_preceding_punctuation
8600     = B(parsers.escapable)
8601     + Cmt(parsers.succeed, check_preceding_unicode_punctuation)
8602
8603 parsers.unicode_preceding_whitespace
8604     = Cmt(parsers.succeed, check_preceding_unicode_whitespace)
8605
8606 parsers.unicode_following_punctuation
8607     = #parsers.escapable
8608     + Cmt(parsers.succeed, check_following_unicode_punctuation)
8609
8610 parsers.unicode_following_whitespace
8611     = Cmt(parsers.succeed, check_following_unicode_whitespace)
8612
8613 parsers.delimiter_run = function(character)
8614     return (B(parsers.backslash * character) + -B(character))
8615             * character~1
8616             * -#character
8617 end
8618
8619 parsers.left_flanking_delimiter_run = function(character)
8620     return (B( parsers.any)
8621             * ( parsers.unicode_preceding_punctuation
8622                 + parsers.unicode_preceding_whitespace)
8623             + -B(parsers.any))
8624             * parsers.delimiter_run(character)
8625             * parsers.unicode_following_punctuation
8626             + parsers.delimiter_run(character)

```

```

8627         * -( parsers.unicode_following_punctuation
8628           + parsers.unicode_following_whitespace
8629           + parsers.eof)
8630     end
8631
8632     parsers.right_flanking_delimiter_run = function(character)
8633         return parsers.unicode_preceding_punctuation
8634            * parsers.delimiter_run(character)
8635            * ( parsers.unicode_following_punctuation
8636              + parsers.unicode_following_whitespace
8637              + parsers.eof)
8638            + (B(parsers.any)
8639              * -( parsers.unicode_preceding_punctuation
8640                  + parsers.unicode_preceding_whitespace))
8641            * parsers.delimiter_run(character)
8642     end
8643
8644     if options.underscores then
8645         parsers.emph_start
8646             = parsers.left_flanking_delimiter_run(parsers.asterisk)
8647             + ( -#parsers.right_flanking_delimiter_run(parsers.underscore)
8648               + ( parsers.unicode_preceding_punctuation
8649                 * #parsers.right_flanking_delimiter_run(parsers.underscore)))
8650             * parsers.left_flanking_delimiter_run(parsers.underscore)
8651
8652         parsers.emph_end
8653             = parsers.right_flanking_delimiter_run(parsers.asterisk)
8654             + ( -#parsers.left_flanking_delimiter_run(parsers.underscore)
8655               + #( parsers.left_flanking_delimiter_run(parsers.underscore)
8656                  * parsers.unicode_following_punctuation))
8657             * parsers.right_flanking_delimiter_run(parsers.underscore)
8658     else
8659         parsers.emph_start
8660             = parsers.left_flanking_delimiter_run(parsers.asterisk)
8661
8662         parsers.emph_end
8663             = parsers.right_flanking_delimiter_run(parsers.asterisk)
8664     end
8665
8666     parsers.emph_capturing_open_and_close
8667         = #parsers.emph_start * #parsers.emph_end
8668         * Ct( Cg(Cc("delimiter"), "type")
8669             * Cg(Cc("emphasis"), "element")
8670             * Cg(C(parsers.emph_start), "content")
8671             * Cg(Cc(true), "is_opening")
8672             * Cg(Cc(true), "is_closing"))
8673

```

```

8674 parsers.emph_capturing_open = Ct( Cg(Cc("delimiter"), "type")
8675                               * Cg(Cc("emphasis"), "element")
8676                               * Cg(C(parsers.emph_start), "content")
8677                               * Cg(Cc(true), "is_opening")
8678                               * Cg(Cc(false), "is_closing"))
8679
8680 parsers.emph_capturing_close = Ct( Cg(Cc("delimiter"), "type")
8681                               * Cg(Cc("emphasis"), "element")
8682                               * Cg(C(parsers.emph_end), "content")
8683                               * Cg(Cc(false), "is_opening")
8684                               * Cg(Cc(true), "is_closing"))
8685
8686 parsers.emph_open_or_close = parsers.emph_capturing_open_and_close
8687                               + parsers.emph_capturing_open
8688                               + parsers.emph_capturing_close
8689
8690 parsers.emph_open = parsers.emph_capturing_open_and_close
8691                   + parsers.emph_capturing_open
8692
8693 parsers.emph_close = parsers.emph_capturing_open_and_close
8694                   + parsers.emph_capturing_close
8695

```

### 3.1.6.7 Helpers for Links and Link Reference Definitions (local)

```

8696 -- List of references defined in the document
8697 local references
8698
8699 -- List of note references defined in the document
8700 parsers.rawnotes = {}
8701

```

The `reader->register_link` method registers a link reference, where `tag` is the link label, `url` is the link destination, `title` is the optional link title, and `attributes` are the optional attributes.

```

8702 function self.register_link(_, tag, url, title,
8703                             attributes)
8704   local normalized_tag = self.normalize_tag(tag)
8705   if references[normalized_tag] == nil then
8706     references[normalized_tag] = {
8707       url = url,
8708       title = title,
8709       attributes = attributes
8710     }
8711   end
8712   return ""
8713 end
8714

```

The `reader->lookup_reference` method looks up a reference with link label `tag`.

```
8715 function self.lookup_reference(tag)
8716     return references[self.normalize_tag(tag)]
8717 end
8718
```

The `reader->lookup_note_reference` method looks up a note reference with label `tag`.

```
8719 function self.lookup_note_reference(tag)
8720     return parsers.rawnotes[self.normalize_tag(tag)]
8721 end
8722
8723 parsers.title_s_direct_ref = parsers.squote
8724                             * Cs((parsers.html_entities
8725                                 + ( parsers.anyescaped
8726                                   - parsers.squote
8727                                   - parsers.blankline^2))^0)
8728                             * parsers.squote
8729
8730 parsers.title_d_direct_ref = parsers.dquote
8731                             * Cs((parsers.html_entities
8732                                 + ( parsers.anyescaped
8733                                   - parsers.dquote
8734                                   - parsers.blankline^2))^0)
8735                             * parsers.dquote
8736
8737 parsers.title_p_direct_ref = parsers.lparent
8738                             * Cs((parsers.html_entities
8739                                 + ( parsers.anyescaped
8740                                   - parsers.lparent
8741                                   - parsers.rparent
8742                                   - parsers.blankline^2))^0)
8743                             * parsers.rparent
8744
8745 parsers.title_direct_ref = parsers.title_s_direct_ref
8746                          + parsers.title_d_direct_ref
8747                          + parsers.title_p_direct_ref
8748
8749 parsers.inline_direct_ref_inside = parsers.lparent * parsers.spnl
8750                                   * Cg(parsers.url + Cc(""), "url")
8751                                   * parsers.spnl
8752                                   * Cg( parsers.title_direct_ref
8753                                       + Cc(""), "title")
8754                                   * parsers.spnl * parsers.rparent
8755
8756 parsers.inline_direct_ref = parsers.lparent * parsers.spnlc
8757                             * Cg(parsers.url + Cc(""), "url")
```

```

8758             * parsers.spnlc
8759             * Cg(parsers.title + Cc(""), "title")
8760             * parsers.spnlc * parsers.rparent
8761
8762 parsers.empty_link = parsers.lbracket
8763                 * parsers.rbracket
8764
8765 parsers.inline_link = parsers.link_text
8766                 * parsers.inline_direct_ref
8767
8768 parsers.full_link = parsers.link_text
8769                 * parsers.link_label
8770
8771 parsers.shortcut_link = parsers.link_label
8772                 * -(parsers.empty_link + parsers.link_label)
8773
8774 parsers.collapsed_link = parsers.link_label
8775                 * parsers.empty_link
8776
8777 parsers.image_opening = #(parsers.exclamation * parsers.inline_link)
8778                 * Cg(Cc("inline"), "link_type")
8779                 + #(parsers.exclamation * parsers.full_link)
8780                 * Cg(Cc("full"), "link_type")
8781                 + #( parsers.exclamation
8782                 * parsers.collapsed_link)
8783                 * Cg(Cc("collapsed"), "link_type")
8784                 + #(parsers.exclamation * parsers.shortcut_link)
8785                 * Cg(Cc("shortcut"), "link_type")
8786                 + #(parsers.exclamation * parsers.empty_link)
8787                 * Cg(Cc("empty"), "link_type")
8788
8789 parsers.link_opening = #parsers.inline_link
8790                 * Cg(Cc("inline"), "link_type")
8791                 + #parsers.full_link
8792                 * Cg(Cc("full"), "link_type")
8793                 + #parsers.collapsed_link
8794                 * Cg(Cc("collapsed"), "link_type")
8795                 + #parsers.shortcut_link
8796                 * Cg(Cc("shortcut"), "link_type")
8797                 + #parsers.empty_link
8798                 * Cg(Cc("empty_link"), "link_type")
8799                 + #parsers.link_text
8800                 * Cg(Cc("link_text"), "link_type")
8801
8802 parsers.note_opening = #(parsers.circumflex * parsers.link_text)
8803                 * Cg(Cc("note_inline"), "link_type")
8804

```

```

8805 parsers.raw_note_opening = #( parsers.lbracket
8806                             * parsers.circumflex
8807                             * parsers.link_label_body
8808                             * parsers.rbracket)
8809                             * Cg(Cc("raw_note"), "link_type")
8810
8811 local inline_note_element = Cg(Cc("note"), "element")
8812                             * parsers.note_opening
8813                             * Cg( parsers.circumflex
8814                               * parsers.lbracket, "content")
8815
8816 local image_element = Cg(Cc("image"), "element")
8817                             * parsers.image_opening
8818                             * Cg( parsers.exclamation
8819                               * parsers.lbracket, "content")
8820
8821 local note_element = Cg(Cc("note"), "element")
8822                             * parsers.raw_note_opening
8823                             * Cg( parsers.lbracket
8824                               * parsers.circumflex, "content")
8825
8826 local link_element = Cg(Cc("link"), "element")
8827                             * parsers.link_opening
8828                             * Cg(parsers.lbracket, "content")
8829
8830 local opening_elements = parsers.fail
8831
8832 if options.inlineNotes then
8833   opening_elements = opening_elements + inline_note_element
8834 end
8835
8836 opening_elements = opening_elements + image_element
8837
8838 if options.notes then
8839   opening_elements = opening_elements + note_element
8840 end
8841
8842 opening_elements = opening_elements + link_element
8843
8844 parsers.link_image_opening = Ct( Cg(Cc("delimiter"), "type")
8845                                 * Cg(Cc(true), "is_opening")
8846                                 * Cg(Cc(false), "is_closing")
8847                                 * opening_elements)
8848
8849 parsers.link_image_closing = Ct( Cg(Cc("delimiter"), "type")
8850                                 * Cg(Cc("link"), "element")
8851                                 * Cg(Cc(false), "is_opening")

```

```

8852             * Cg(Cc(true), "is_closing")
8853             * ( Cg(Cc(true), "is_direct")
8854             * Cg( parsers.rbracket
8855                 * #parsers.inline_direct_ref,
8856                 "content")
8857             + Cg(Cc(false), "is_direct")
8858             * Cg(parsers.rbracket, "content")))
8859
8860 parsers.link_image_open_or_close = parsers.link_image_opening
8861                                 + parsers.link_image_closing
8862
8863 if options.html then
8864     parsers.link_emph_precedence = parsers.inticks
8865                                 + parsers.autolink
8866                                 + parsers.html_inline_tags
8867 else
8868     parsers.link_emph_precedence = parsers.inticks
8869                                 + parsers.autolink
8870 end
8871
8872 parsers.link_and_emph_endline = parsers.newline
8873                               * ((parsers.check_minimal_indent
8874                               * -V("EndlineExceptions")
8875                               + parsers.check_optional_indent
8876                               * -V("EndlineExceptions")
8877                               * -parsers.starter) / "")
8878                               * parsers.spacechar^0 / "\n"
8879
8880 parsers.link_and_emph_content
8881   = Ct( Cg(Cc("content"), "type")
8882       * Cg(Cs(( parsers.link_emph_precedence
8883                 + parsers.backslash * parsers.any
8884                 + parsers.link_and_emph_endline
8885                 + (parsers.linechar
8886                   - parsers.blankline^2
8887                   - parsers.link_image_open_or_close
8888                   - parsers.emph_open_or_close))^0), "content"))
8889
8890 parsers.link_and_emph_table
8891   = (parsers.link_image_opening + parsers.emph_open)
8892     * parsers.link_and_emph_content
8893     * ((parsers.link_image_open_or_close + parsers.emph_open_or_close)
8894       * parsers.link_and_emph_content)^1
8895

```

Collect the content between the [opening\\_index](#) and [closing\\_index](#) in the delimiter table `t`.

```

8896 local function collect_link_content(t, opening_index, closing_index)
8897     local content = {}
8898     for i = opening_index, closing_index do
8899         content[#content + 1] = t[i].content
8900     end
8901     return util.rope_to_string(content)
8902 end
8903

```

Look for the closest potential link opener in the delimiter table `t` in the range from `bottom_index` to `latest_index`.

```

8904 local function find_link_opener(t, bottom_index, latest_index)
8905     for i = latest_index, bottom_index, -1 do
8906         local value = t[i]
8907         if value.type == "delimiter" and
8908             value.is_opening and
8909             ( value.element == "link"
8910             or value.element == "image"
8911             or value.element == "note")
8912             and not value.removed then
8913             if value.is_active then
8914                 return i
8915             end
8916             value.removed = true
8917             return nil
8918         end
8919     end
8920 end
8921

```

Find the position of a delimiter that closes a full link after an an index `latest_index` in the delimiter table `t`.

```

8922 local function find_next_link_closing_index(t, latest_index)
8923     for i = latest_index, #t do
8924         local value = t[i]
8925         if value.is_closing and
8926             value.element == "link" and
8927             not value.removed then
8928             return i
8929         end
8930     end
8931 end
8932

```

Disable all preceding opening link delimiters by marking them inactive with the `is_active` property to prevent links within links. Images within links are allowed.

```

8933 local function disable_previous_link_openers(t, opening_index)
8934     if t[opening_index].element == "image" then

```



```

8935     return
8936 end
8937
8938 for i = opening_index, 1, -1 do
8939     local value = t[i]
8940     if value.is_active and
8941         value.type == "delimiter" and
8942         value.is_opening and
8943         value.element == "link" then
8944         value.is_active = false
8945     end
8946 end
8947 end
8948

```

Disable the delimiters between the `opening_index` and `closing_index` in the delimiter table `t` by marking them inactive with the `is_active` property.

```

8949 local function disable_range(t, opening_index, closing_index)
8950     for i = opening_index, closing_index do
8951         local value = t[i]
8952         if value.is_active then
8953             value.is_active = false
8954             if value.type == "delimiter" then
8955                 value.removed = true
8956             end
8957         end
8958     end
8959 end
8960

```

Clear the parsed content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

8961 local delete_parsed_content_in_range =
8962     function(t, opening_index, closing_index)
8963         for i = opening_index, closing_index do
8964             t[i].rendered = nil
8965         end
8966     end
8967

```

Clear the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

8968 local function empty_content_in_range(t, opening_index, closing_index)
8969     for i = opening_index, closing_index do
8970         t[i].content = ''
8971     end
8972 end
8973

```

Join the attributes from the link reference definition `reference_attributes` with the link's own attributes `own_attributes`.

```
8974 local function join_attributes(reference_attributes, own_attributes)
8975   local merged_attributes = {}
8976   for _, attribute in ipairs(reference_attributes or {}) do
8977     table.insert(merged_attributes, attribute)
8978   end
8979   for _, attribute in ipairs(own_attributes or {}) do
8980     table.insert(merged_attributes, attribute)
8981   end
8982   if next(merged_attributes) == nil then
8983     merged_attributes = nil
8984   end
8985   return merged_attributes
8986 end
8987
```

Parse content between two delimiters in the delimiter table `t`. Produce the respective link and image macros.

```
8988 local render_link_or_image =
8989   function(t, opening_index, closing_index, content_end_index,
8990     reference)
8991     process_emphasis(t, opening_index, content_end_index)
8992     local mapped = collect_emphasis_content(t, opening_index + 1,
8993       content_end_index - 1)
8994
8995     local rendered = {}
8996     if (t[opening_index].element == "link") then
8997       rendered = writer.link(mapped, reference.url,
8998         reference.title, reference.attributes)
8999     end
9000
9001     if (t[opening_index].element == "image") then
9002       rendered = writer.image(mapped, reference.url, reference.title,
9003         reference.attributes)
9004     end
9005
9006     if (t[opening_index].element == "note") then
9007       if (t[opening_index].link_type == "note_inline") then
9008         rendered = writer.note(mapped)
9009       end
9010       if (t[opening_index].link_type == "raw_note") then
9011         rendered = writer.note(reference)
9012       end
9013     end
9014
9015     t[opening_index].rendered = rendered

```

```

9016     delete_parsed_content_in_range(t, opening_index + 1,
9017                                     closing_index)
9018     empty_content_in_range(t, opening_index, closing_index)
9019     disable_previous_link_openers(t, opening_index)
9020     disable_range(t, opening_index, closing_index)
9021 end
9022

```

Match the link destination of an inline link at index `closing_index` in table `t` when `match_reference` is true. Additionally, match attributes when the option `linkAttributes` is enabled.

```

9023 local resolve_inline_following_content =
9024     function(t, closing_index, match_reference, match_link_attributes)
9025         local content = ""
9026         for i = closing_index + 1, #t do
9027             content = content .. t[i].content
9028         end
9029
9030         local matching_content = parsers.succeed
9031
9032         if match_reference then
9033             matching_content = matching_content
9034                 * parsers.inline_direct_ref_inside
9035         end
9036
9037         if match_link_attributes then
9038             matching_content = matching_content
9039                 * Cg(Ct(parsers.attributes^-1), "attributes")
9040         end
9041
9042         local matched = lpeg.match(Ct( matching_content
9043                                     * Cg(Cp(), "end_position")), content)
9044
9045         local matched_count = matched.end_position - 1
9046         for i = closing_index + 1, #t do
9047             local value = t[i]
9048
9049             local chars_left = matched_count
9050             matched_count = matched_count - #value.content
9051
9052             if matched_count <= 0 then
9053                 value.content = value.content:sub(chars_left + 1)
9054                 break
9055             end
9056
9057             value.content = ''
9058             value.is_active = false

```

```

9059     end
9060
9061     local attributes = matched.attributes
9062     if attributes == nil or next(attributes) == nil then
9063         attributes = nil
9064     end
9065
9066     return {
9067         url = matched.url or "",
9068         title = matched.title or "",
9069         attributes = attributes
9070     }
9071 end
9072

```

Resolve an inline link `[a](b "c")` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Here, compared to other types of links, no reference definition is needed.

```

9073 local function resolve_inline_link(t, opening_index, closing_index)
9074     local inline_content
9075     = resolve_inline_following_content(t, closing_index, true,
9076                                       t.match_link_attributes)
9077     render_link_or_image(t, opening_index, closing_index,
9078                         closing_index, inline_content)
9079 end
9080

```

Resolve an inline note `^[a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`.

```

9081 local resolve_note_inline_link =
9082 function(t, opening_index, closing_index)
9083     local inline_content
9084     = resolve_inline_following_content(t, closing_index,
9085                                       false, false)
9086     render_link_or_image(t, opening_index, closing_index,
9087                         closing_index, inline_content)
9088 end
9089

```

Resolve a shortcut link `[a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `a` is not found in the references.

```

9090 local function resolve_shortcut_link(t, opening_index, closing_index)
9091     local content
9092     = collect_link_content(t, opening_index + 1, closing_index - 1)
9093     local r = self.lookup_reference(content)
9094
9095     if r then

```

```

9096     local inline_content
9097         = resolve_inline_following_content(t, closing_index, false,
9098                                           t.match_link_attributes)
9099     r.attributes
9100         = join_attributes(r.attributes, inline_content.attributes)
9101     render_link_or_image(t, opening_index, closing_index,
9102                         closing_index, r)
9103 end
9104 end
9105

```

Resolve a note `[^a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `a` is not found in the rawnotes.

```

9106 local function resolve_raw_note_link(t, opening_index, closing_index)
9107     local content
9108         = collect_link_content(t, opening_index + 1, closing_index - 1)
9109     local r = self.lookup_note_reference(content)
9110
9111     if r then
9112         local parsed_ref = self.parser_functions.parse_blocks_nested(r)
9113         render_link_or_image(t, opening_index, closing_index,
9114                             closing_index, parsed_ref)
9115     end
9116 end
9117

```

Resolve a full link `[a][b]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `b` is not found in the references.

```

9118 local function resolve_full_link(t, opening_index, closing_index)
9119     local next_link_closing_index
9120         = find_next_link_closing_index(t, closing_index + 4)
9121     local next_link_content
9122         = collect_link_content(t, closing_index + 3,
9123                               next_link_closing_index - 1)
9124     local r = self.lookup_reference(next_link_content)
9125
9126     if r then
9127         local inline_content
9128             = resolve_inline_following_content(t, next_link_closing_index,
9129                                               false,
9130                                               t.match_link_attributes)
9131         r.attributes
9132             = join_attributes(r.attributes, inline_content.attributes)
9133         render_link_or_image(t, opening_index, next_link_closing_index,
9134                             closing_index, r)
9135     end
9136 end
9137

```

Resolve a collapsed link `[a] []` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `a` is not found in the references.

```

9138 local function resolve_collapsed_link(t, opening_index, closing_index)
9139     local next_link_closing_index
9140         = find_next_link_closing_index(t, closing_index + 4)
9141     local content
9142         = collect_link_content(t, opening_index + 1, closing_index - 1)
9143     local r = self.lookup_reference(content)
9144
9145     if r then
9146         local inline_content
9147             = resolve_inline_following_content(t, closing_index, false,
9148                                               t.match_link_attributes)
9149         r.attributes
9150             = join_attributes(r.attributes, inline_content.attributes)
9151         render_link_or_image(t, opening_index, next_link_closing_index,
9152                             closing_index, r)
9153     end
9154 end
9155

```

Parse a table of link and emphasis delimiters `t`. First, iterate over the link delimiters and produce either link or image macros. Then run `process_emphasis` over the entire delimiter table, resolving emphasis and strong emphasis and parsing any content outside of closed delimiters.

```

9156 local function process_links_and_emphasis(t)
9157     for _,value in ipairs(t) do
9158         value.is_active = true
9159     end
9160
9161     for i,value in ipairs(t) do
9162         if not value.is_closing
9163             or value.type ~= "delimiter"
9164             or not ( value.element == "link"
9165                     or value.element == "image"
9166                     or value.element == "note")
9167             or value.removed then
9168             goto continue
9169         end
9170
9171         local opener_position = find_link_opener(t, 1, i - 1)
9172         if (opener_position == nil) then
9173             goto continue
9174         end
9175
9176         local opening_delimiter = t[opener_position]

```

```

9177     opening_delimiter.removed = true
9178
9179     local link_type = opening_delimiter.link_type
9180
9181     if (link_type == "inline") then
9182         resolve_inline_link(t, opener_position, i)
9183     end
9184     if (link_type == "shortcut") then
9185         resolve_shortcut_link(t, opener_position, i)
9186     end
9187     if (link_type == "full") then
9188         resolve_full_link(t, opener_position, i)
9189     end
9190     if (link_type == "collapsed") then
9191         resolve_collapsed_link(t, opener_position, i)
9192     end
9193     if (link_type == "note_inline") then
9194         resolve_note_inline_link(t, opener_position, i)
9195     end
9196     if (link_type == "raw_note") then
9197         resolve_raw_note_link(t, opener_position, i)
9198     end
9199
9200     ::continue::
9201 end
9202
9203 t[#t].content = t[#t].content:gsub("%s*$", "")
9204
9205 process_emphasis(t, 1, #t)
9206 local final_result = collect_emphasis_content(t, 1, #t)
9207 return final_result
9208 end
9209
9210 function self.defer_link_and_emphasis_processing(delimiter_table)
9211     return writer.defer_call(function()
9212         return process_links_and_emphasis(delimiter_table)
9213     end)
9214 end
9215

```

### 3.1.6.8 Inline Elements (local)

```

9216 parsers.Str      = ( parsers.normalchar
9217                     * (parsers.normalchar + parsers.at)^0)
9218                   / writer.string
9219
9220 parsers.Symbol   = (parsers.backtick^1 + V("SpecialChar"))

```

```

9221             / writer.string
9222
9223 parsers.Ellipsis = P("...") / writer.ellipsis
9224
9225 parsers.Smart    = parsers.Ellipsis
9226
9227 parsers.Code     = parsers.inticks / writer.code
9228
9229 if options.blankBeforeBlockquote then
9230   parsers.bqstart = parsers.fail
9231 else
9232   parsers.bqstart = parsers.blockquote_start
9233 end
9234
9235 if options.blankBeforeHeading then
9236   parsers.headerstart = parsers.fail
9237 else
9238   parsers.headerstart = parsers.atx_heading
9239 end
9240
9241 if options.blankBeforeList then
9242   parsers.interrupting_bullets = parsers.fail
9243   parsers.interrupting_enumerators = parsers.fail
9244 else
9245   parsers.interrupting_bullets
9246     = parsers.bullet(parsers.dash, true)
9247     + parsers.bullet(parsers.asterisk, true)
9248     + parsers.bullet(parsers.plus, true)
9249
9250   parsers.interrupting_enumerators
9251     = parsers.enumerator(parsers.period, true)
9252     + parsers.enumerator(parsers.rparent, true)
9253 end
9254
9255 if options.html then
9256   parsers.html_interrupting
9257     = parsers.check_trail
9258     * ( parsers.html_incomplete_open_tag
9259         + parsers.html_incomplete_close_tag
9260         + parsers.html_incomplete_open_special_tag
9261         + parsers.html_comment_start
9262         + parsers.html_cdatasection_start
9263         + parsers.html_declaration_start
9264         + parsers.html_instruction_start
9265         - parsers.html_close_special_tag
9266         - parsers.html_empty_special_tag)
9267 else

```



```

9268     parsers.html_interrupting = parsers.fail
9269 end
9270
9271 parsers.EndlineExceptions
9272     = parsers.blankline -- paragraph break
9273     + parsers.eof      -- end of document
9274     + parsers.bqstart
9275     + parsers.thematic_break_lines
9276     + parsers.interrupting_bullets
9277     + parsers.interrupting_enumerators
9278     + parsers.headerstart
9279     + parsers.html_interrupting
9280
9281 parsers.NoSoftLineBreakEndlineExceptions = parsers.EndlineExceptions
9282
9283 parsers.endline = parsers.newline
9284     * (parsers.check_minimal_indent
9285     * -V("EndlineExceptions")
9286     + parsers.check_optional_indent
9287     * -V("EndlineExceptions")
9288     * -parsers.starter) / function(_) return end
9289     * parsers.spacechar^0
9290
9291 parsers.Endline = parsers.endline
9292     / writer.soft_line_break
9293
9294 parsers.EndlineNoSub = parsers.endline
9295
9296 parsers.NoSoftLineBreakEndline
9297     = parsers.newline
9298     * (parsers.check_minimal_indent
9299     * -V("NoSoftLineBreakEndlineExceptions")
9300     + parsers.check_optional_indent
9301     * -V("NoSoftLineBreakEndlineExceptions")
9302     * -parsers.starter)
9303     * parsers.spacechar^0
9304     / writer.space
9305
9306 parsers.EndlineBreak = parsers.backslash * parsers.Endline
9307     / writer.hard_line_break
9308
9309 parsers.OptionalIndent
9310     = parsers.spacechar^1 / writer.space
9311
9312 parsers.Space = parsers.spacechar^2 * parsers.Endline
9313     / writer.hard_line_break
9314     + parsers.spacechar^1

```

```

9315         * parsers.Endline~-1
9316         * parsers.eof / self.expandtabs
9317         + parsers.spacechar^1 * parsers.Endline
9318           / writer.soft_line_break
9319         + parsers.spacechar^1
9320         * -parsers.newline / self.expandtabs
9321
9322     parsers.NoSoftLineBreakSpace
9323         = parsers.spacechar^2 * parsers.Endline
9324           / writer.hard_line_break
9325         + parsers.spacechar^1
9326         * parsers.Endline~-1
9327         * parsers.eof / self.expandtabs
9328         + parsers.spacechar^1 * parsers.Endline
9329           / writer.soft_line_break
9330         + parsers.spacechar^1
9331         * -parsers.newline / self.expandtabs
9332
9333     parsers.NonbreakingEndline
9334         = parsers.endline
9335         / writer.nbsp
9336
9337     parsers.NonbreakingSpace
9338         = parsers.spacechar^2 * parsers.endline
9339           / writer.nbsp
9340         + parsers.spacechar^1
9341         * parsers.endline~-1 * parsers.eof / ""
9342         + parsers.spacechar^1 * parsers.endline
9343           * parsers.optionalspace
9344           / writer.nbsp
9345         + parsers.spacechar^1 * parsers.optionalspace
9346           / writer.nbsp
9347

```

The `reader->auto_link_url` method produces an autolink to a URL or a relative reference in the output format, where `url` is the link destination and `attributes` are the optional attributes.

```

9348 function self.auto_link_url(url, attributes)
9349     return writer.link(writer.escape(url),
9350                       url, nil, attributes)
9351 end

```

The `reader->auto_link_email` method produces an autolink to an e-mail in the output format, where `email` is the email address destination and `attributes` are the optional attributes.

```

9352 function self.auto_link_email(email, attributes)
9353     return writer.link(writer.escape(email),

```

```

9354         "mailto:".email,
9355         nil, attributes)
9356     end
9357
9358     parsers.AutoLinkUrl = parsers.auto_link_url
9359         / self.auto_link_url
9360
9361     parsers.AutoLinkEmail
9362         = parsers.auto_link_email
9363         / self.auto_link_email
9364
9365     parsers.AutoLinkRelativeReference
9366         = parsers.auto_link_relative_reference
9367         / self.auto_link_url
9368
9369     parsers.LinkAndEmph = Ct(parsers.link_and_emph_table)
9370         / self.defer_link_and_emphasis_processing
9371
9372     parsers.EscapedChar = parsers.backslash
9373         * C(parsers.escapable) / writer.string
9374
9375     parsers.InlineHtml = Cs(parsers.html_inline_comment)
9376         / writer.inline_html_comment
9377         + Cs(parsers.html_any_empty_inline_tag
9378             + parsers.html_inline_instruction
9379             + parsers.html_inline_cdatasection
9380             + parsers.html_inline_declaration
9381             + parsers.html_any_open_inline_tag
9382             + parsers.html_any_close_tag)
9383         / writer.inline_html_tag
9384
9385     parsers.HtmlEntity = parsers.html_entities / writer.string

```

### 3.1.6.9 Block Elements (local)

```

9386     parsers.DisplayHtml = Cs(parsers.check_trail
9387         * ( parsers.html_comment
9388           + parsers.html_special_block
9389           + parsers.html_block
9390           + parsers.html_any_block
9391           + parsers.html_instruction
9392           + parsers.html_cdatasection
9393           + parsers.html_declaration))
9394         / writer.block_html_element
9395
9396     parsers.indented_non_blank_line = parsers.indentedline
9397         - parsers.blankline

```

```

9398
9399 parsers.Verbatim
9400   = Cs( parsers.check_code_trail
9401       * (parsers.line - parsers.blankline)
9402       * (( parsers.check_minimal_blank_indent_and_full_code_trail
9403         * parsers.blankline)^0
9404         * ( (parsers.check_minimal_indent / "")
9405           * parsers.check_code_trail
9406           * (parsers.line - parsers.blankline))^1)^0)
9407   / self.expandtabs / writer.verbatim
9408
9409 parsers.Blockquote = parsers.blockquote_body
9410                   / writer.blockquote
9411
9412 parsers.ThematicBreak = parsers.thematic_break_lines
9413                       / writer.thematic_break
9414
9415 parsers.Reference     = parsers.define_reference_parser
9416                       / self.register_link
9417
9418 parsers.Paragraph     = parsers.freeze_trail
9419                       * (Ct((parsers.Inline)^1)
9420                       * (parsers.newline + parsers.eof)
9421                       * parsers.unfreeze_trail
9422                       / writer.paragraph)
9423
9424 parsers.Plain         = parsers.nonindentspace * Ct(parsers.Inline^1)
9425                       / writer.plain

```

### 3.1.6.10 Lists (local)

```

9426
9427 if options.taskLists then
9428   parsers.tickbox = ( parsers.ticked_box
9429                     + parsers.halfticked_box
9430                     + parsers.unticked_box
9431                     ) / writer.tickbox
9432 else
9433   parsers.tickbox = parsers.fail
9434 end
9435
9436 parsers.list_blank = parsers.conditionally_indented_blankline
9437
9438 parsers.ref_or_block_list_separated
9439   = parsers.sep_group_no_output(parsers.list_blank)
9440   * parsers.minimally_indented_ref
9441   + parsers.block_sep_group(parsers.list_blank)

```

```

9442     * parsers.minimally_indented_block
9443
9444 parsers.ref_or_block_non_separated
9445     = parsers.minimally_indented_ref
9446     + (parsers.succeed / writer.interblocksep)
9447     * parsers.minimally_indented_block
9448     - parsers.minimally_indented_blankline
9449
9450 parsers.tight_list_loop_body_pair =
9451     parsers.create_loop_body_pair(
9452         parsers.ref_or_block_non_separated,
9453         parsers.minimally_indented_par_or_plain_no_blank,
9454         (parsers.succeed / writer.interblocksep),
9455         (parsers.succeed / writer.paragraphsep))
9456
9457 parsers.loose_list_loop_body_pair =
9458     parsers.create_loop_body_pair(
9459         parsers.ref_or_block_list_separated,
9460         parsers.minimally_indented_par_or_plain,
9461         parsers.block_sep_group(parsers.list_blank),
9462         parsers.par_sep_group(parsers.list_blank))
9463
9464 parsers.tight_list_content_loop
9465     = V("Block")
9466     * parsers.tight_list_loop_body_pair.block^0
9467     + (V("Paragraph") + V("Plain"))
9468     * parsers.ref_or_block_non_separated
9469     * parsers.tight_list_loop_body_pair.block^0
9470     + (V("Paragraph") + V("Plain"))
9471     * parsers.tight_list_loop_body_pair.par^0
9472
9473 parsers.loose_list_content_loop
9474     = V("Block")
9475     * parsers.loose_list_loop_body_pair.block^0
9476     + (V("Paragraph") + V("Plain"))
9477     * parsers.ref_or_block_list_separated
9478     * parsers.loose_list_loop_body_pair.block^0
9479     + (V("Paragraph") + V("Plain"))
9480     * parsers.loose_list_loop_body_pair.par^0
9481
9482 parsers.list_item_tightness_condition
9483     = -( parsers.list_blank^0
9484         * parsers.minimally_indented_ref_or_block_or_par)
9485     * remove_indent("li")
9486     + remove_indent("li")
9487     * parsers.fail
9488

```

```

9489 parsers.indented_content_tight
9490   = Ct( (parsers.blankline / "")
9491         * #parsers.list_blank
9492         * remove_indent("li")
9493         + ( (V("Reference") + (parsers.blankline / ""))
9494             * parsers.check_minimal_indent
9495             * parsers.tight_list_content_loop
9496             + (V("Reference") + (parsers.blankline / ""))
9497             + (parsers.checkbox^-1 / writer.escape)
9498             * parsers.tight_list_content_loop
9499             )
9500         * parsers.list_item_tightness_condition)
9501
9502 parsers.indented_content_loose
9503   = Ct( (parsers.blankline / "")
9504         * #parsers.list_blank
9505         + ( (V("Reference") + (parsers.blankline / ""))
9506             * parsers.check_minimal_indent
9507             * parsers.loose_list_content_loop
9508             + (V("Reference") + (parsers.blankline / ""))
9509             + (parsers.checkbox^-1 / writer.escape)
9510             * parsers.loose_list_content_loop))
9511
9512 parsers.TightListItem = function(starter)
9513   return -parsers.ThematicBreak
9514         * parsers.add_indent(starter, "li")
9515         * parsers.indented_content_tight
9516 end
9517
9518 parsers.LooseListItem = function(starter)
9519   return -parsers.ThematicBreak
9520         * parsers.add_indent(starter, "li")
9521         * parsers.indented_content_loose
9522         * remove_indent("li")
9523 end
9524
9525 parsers.BulletListOfType = function(bullet_type)
9526   local bullet = parsers.bullet(bullet_type)
9527   return ( Ct( parsers.TightListItem(bullet)
9528               * ( (parsers.check_minimal_indent / "")
9529                   * parsers.TightListItem(bullet)
9530                   )^0
9531               )
9532           * Cc(true)
9533           * -( (parsers.list_blank^0 / "")
9534               * parsers.check_minimal_indent
9535               * (bullet - parsers.ThematicBreak)

```

```

9536         )
9537         + Ct( parsers.LooseListItem(bullet)
9538             * ( (parsers.list_blank^0 / "")
9539               * (parsers.check_minimal_indent / "")
9540               * parsers.LooseListItem(bullet)
9541               )^0
9542         )
9543         * Cc(false)
9544     ) / writer.bulletlist
9545 end
9546
9547 parsers.BulletList = parsers.BulletListOfType(parsers.dash)
9548                   + parsers.BulletListOfType(parsers.asterisk)
9549                   + parsers.BulletListOfType(parsers.plus)
9550
9551 local function ordered_list(items,tight,starter)
9552     local startnum = starter[2][1]
9553     if options.startNumber then
9554         startnum = tonumber(startnum) or 1 -- fallback for '#'
9555         if startnum ~= nil then
9556             startnum = math.floor(startnum)
9557         end
9558     else
9559         startnum = nil
9560     end
9561     return writer.orderedlist(items,tight,startnum)
9562 end
9563
9564 parsers.OrderedListOfTypes = function(delimiter_type)
9565     local enumerator = parsers.enumerator(delimiter_type)
9566     return Cg(enumerator, "listtype")
9567           * (Ct( parsers.TightListItem(Cb("listtype"))
9568              * ( (parsers.check_minimal_indent / "")
9569                * parsers.TightListItem(enumerator))^0)
9570           * Cc(true)
9571           * -#((parsers.list_blank^0 / "")
9572              * parsers.check_minimal_indent * enumerator)
9573     + Ct( parsers.LooseListItem(Cb("listtype"))
9574        * ((parsers.list_blank^0 / "")
9575          * (parsers.check_minimal_indent / "")
9576          * parsers.LooseListItem(enumerator))^0)
9577     * Cc(false)
9578     ) * Ct(Cb("listtype")) / ordered_list
9579 end
9580
9581 parsers.OrderedList = parsers.OrderedListOfTypes(parsers.period)
9582                   + parsers.OrderedListOfTypes(parsers.rparent)

```

### 3.1.6.11 Blank (local)

```
9583 parsers.Blank          = parsers.blankline / ""
9584                        + V("Reference")
```

### 3.1.6.12 Headings (local)

```
9585 function parsers.parse_heading_text(s)
9586     local inlines = self.parser_functions.parse_inlines(s)
9587     local flatten_inlines = self.writer.flatten_inlines
9588     self.writer.flatten_inlines = true
9589     local flat_text = self.parser_functions.parse_inlines(s)
9590     flat_text = util.ropetostring(flat_text)
9591     self.writer.flatten_inlines = flatten_inlines
9592     return {flat_text, inlines}
9593 end
9594
9595 -- parse atx header
9596 parsers.AtxHeading = parsers.check_trail_no_rem
9597                    * Cg(parsers.heading_start, "level")
9598                    * ((C( parsers.optionalspace
9599                        * parsers.hash^0
9600                        * parsers.optionalspace
9601                        * parsers.newline)
9602                      + parsers.spacechar^1
9603                      * C(parsers.line))
9604                      / strip_atx_end
9605                      / parsers.parse_heading_text)
9606                    * Cb("level")
9607                    / writer.heading
9608
9609 parsers.heading_line = parsers.linechar^1
9610                    - parsers.thematic_break_lines
9611
9612 parsers.heading_text = parsers.heading_line
9613                    * ( (V("Endline") / "\n")
9614                      * ( parsers.heading_line
9615                        - parsers.heading_level))^0
9616                    * parsers.newline^-1
9617
9618 parsers.SetextHeading = parsers.freeze_trail
9619                    * parsers.check_trail_no_rem
9620                    * #( parsers.heading_text
9621                      * parsers.check_minimal_indent
9622                      * parsers.check_trail
9623                      * parsers.heading_level)
9624                    * Cs(parsers.heading_text)
9625                    / parsers.parse_heading_text
```



```

9626             * parsers.check_minimal_indent_and_trail
9627             * parsers.heading_level
9628             * parsers.newline
9629             * parsers.unfreeze_trail
9630             / writer.heading
9631
9632 parsers.Heading = parsers.AtxHeading + parsers.SetextHeading

```

### 3.1.6.13 Syntax Specification

Define `reader->finalize_grammar` as a function that constructs the PEG grammar of markdown, applies syntax extensions `extensions` and returns a conversion function that takes a markdown string and turns it into a plain T<sub>E</sub>X output.

```

9633 function self.finalize_grammar(extensions)

```

Create a local writable copy of the global read-only `walkable_syntax` hash table. This table can be used by user-defined syntax extensions to insert new PEG patterns into existing rules of the PEG grammar of markdown using the `reader->insert_pattern` method. Furthermore, built-in syntax extensions can use this table to override existing rules using the `reader->update_rule` method.

```

9634     local walkable_syntax = (function(global_walkable_syntax)
9635         local local_walkable_syntax = {}
9636         for lhs, rule in pairs(global_walkable_syntax) do
9637             local_walkable_syntax[lhs] = util.table_copy(rule)
9638         end
9639         return local_walkable_syntax
9640     end)(walkable_syntax)

```

The `reader->insert_pattern` method adds a pattern to `walkable_syntax` [*left-hand side terminal symbol*] before, instead of, or after a right-hand-side terminal symbol.

```

9641     local current_extension_name = nil
9642     self.insert_pattern = function(selector, pattern, pattern_name)
9643         assert(pattern_name == nil or type(pattern_name) == "string")
9644         local _, _, lhs, pos, rhs
9645             = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
9646         assert(lhs ~= nil,
9647             [[Expected selector in form ]]
9648             .. [[LHS (before|after|instead of) RHS", not "]]
9649             .. selector .. [[]])
9650         assert(walkable_syntax[lhs] ~= nil,
9651             [[Rule ]] .. lhs
9652             .. [[ -> ... does not exist in markdown grammar]])
9653         assert(pos == "before" or pos == "after" or pos == "instead of",
9654             [[Expected positional specifier "before", "after", ]]
9655             .. [[or "instead of", not "]]
9656             .. pos .. [[]])

```

```

9657     local rule = walkable_syntax[lhs]
9658     local index = nil
9659     for current_index, current_rhs in ipairs(rule) do
9660         if type(current_rhs) == "string" and current_rhs == rhs then
9661             index = current_index
9662             if pos == "after" then
9663                 index = index + 1
9664             end
9665             break
9666         end
9667     end
9668     assert(index ~= nil,
9669         [[Rule ]] .. lhs .. [[ -> ]] .. rhs
9670         .. [[ does not exist in markdown grammar]])
9671     local accountable_pattern
9672     if current_extension_name then
9673         accountable_pattern
9674         = {pattern, current_extension_name, pattern_name}
9675     else
9676         assert(type(pattern) == "string",
9677             [[reader->insert_pattern() was called outside ]]
9678             .. [[an extension with ]]
9679             .. [[a PEG pattern instead of a rule name]])
9680         accountable_pattern = pattern
9681     end
9682     if pos == "instead of" then
9683         rule[index] = accountable_pattern
9684     else
9685         table.insert(rule, index, accountable_pattern)
9686     end
9687 end

```

Create a local `syntax` hash table that stores those rules of the PEG grammar of markdown that can't be represented as an ordered choice of terminal symbols.

```

9688     local syntax =
9689         { "Blocks",
9690
9691           Blocks = V("InitializeState")
9692                 * V("ExpectedJekyllData")
9693                 * V("Blank")^0

```

Only create interblock separators between pairs of blocks that are not both paragraphs. Between a pair of paragraphs, any number of blank lines will always produce a paragraph separator.

```

9694         * ( V("Block")
9695           * ( V("Blank")^0 * parsers.eof
9696             + ( V("Blank")^2 / writer.paragraphsep
9697               + V("Blank")^0 / writer.interblocksep

```

```

9698         )
9699     )
9700     + ( V("Paragraph") + V("Plain") )
9701     * ( V("Blank")^0 * parsers.eof
9702         + ( V("Blank")^2 / writer.paragraphsep
9703             + V("Blank")^0 / writer.interblocksep
9704         )
9705     )
9706     * V("Block")
9707     * ( V("Blank")^0 * parsers.eof
9708         + ( V("Blank")^2 / writer.paragraphsep
9709             + V("Blank")^0 / writer.interblocksep
9710         )
9711     )
9712     + ( V("Paragraph") + V("Plain") )
9713     * ( V("Blank")^0 * parsers.eof
9714         + V("Blank")^0 / writer.paragraphsep
9715     )
9716 )^0,
9717
9718 ExpectedJekyllData = parsers.succeed,
9719
9720 Blank                = parsers.Blank,
9721 Reference            = parsers.Reference,
9722
9723 Blockquote          = parsers.Blockquote,
9724 Verbatim            = parsers.Verbatim,
9725 ThematicBreak      = parsers.ThematicBreak,
9726 BulletList         = parsers.BulletList,
9727 OrderedList        = parsers.OrderedList,
9728 DisplayHtml        = parsers.DisplayHtml,
9729 Heading            = parsers.Heading,
9730 Paragraph          = parsers.Paragraph,
9731 Plain              = parsers.Plain,
9732
9733 EndlineExceptions  = parsers.EndlineExceptions,
9734 NoSoftLineBreakEndlineExceptions
9735                   = parsers.NoSoftLineBreakEndlineExceptions,
9736
9737 Str                = parsers.Str,
9738 Space              = parsers.Space,
9739 NoSoftLineBreakSpace
9740                   = parsers.NoSoftLineBreakSpace,
9741 OptionalIndent     = parsers.OptionalIndent,
9742 Endline            = parsers.Endline,
9743 EndlineNoSub       = parsers.EndlineNoSub,
9744 NoSoftLineBreakEndline

```

```

9745             = parsers.NoSoftLineBreakEndline,
9746     EndlineBreak      = parsers.EndlineBreak,
9747     LinkAndEmph       = parsers.LinkAndEmph,
9748     Code              = parsers.Code,
9749     AutoLinkUrl       = parsers.AutoLinkUrl,
9750     AutoLinkEmail     = parsers.AutoLinkEmail,
9751     AutoLinkRelativeReference
9752             = parsers.AutoLinkRelativeReference,
9753     InlineHtml        = parsers.InlineHtml,
9754     HtmlEntity        = parsers.HtmlEntity,
9755     EscapedChar       = parsers.EscapedChar,
9756     Smart             = parsers.Smart,
9757     Symbol            = parsers.Symbol,
9758     SpecialChar       = parsers.fail,
9759     InitializeState   = parsers.succeed,
9760 }

```

Define `reader->update_rule` as a function that receives two arguments: a left-hand side terminal symbol and a function that accepts the current PEG pattern in `walkable_syntax[left-hand side terminal symbol]` if defined or `nil` otherwise and returns a PEG pattern that will (re)define `walkable_syntax[left-hand side terminal symbol]`.

```

9761     self.update_rule = function(rule_name, get_pattern)
9762         assert(current_extension_name ~= nil)
9763         assert(syntax[rule_name] ~= nil,
9764             [[Rule ]] .. rule_name
9765             .. [[ -> ... does not exist in markdown grammar]])
9766         local previous_pattern
9767         local extension_name
9768         if walkable_syntax[rule_name] then
9769             local previous_accountable_pattern
9770                 = walkable_syntax[rule_name][1]
9771             previous_pattern = previous_accountable_pattern[1]
9772             extension_name
9773                 = previous_accountable_pattern[2]
9774             .. ", " .. current_extension_name
9775         else
9776             previous_pattern = nil
9777             extension_name = current_extension_name
9778         end
9779         local pattern

```

Instead of a function, a PEG pattern `pattern` may also be supplied with roughly the same effect as supplying the following function, which will define `walkable_syntax[left-hand side terminal symbol]` unless it has been previously defined.

```

function(previous_pattern)
  assert(previous_pattern == nil)
  return pattern
end

```

```

9780     if type(get_pattern) == "function" then
9781         pattern = get_pattern(previous_pattern)
9782     else
9783         assert(previous_pattern == nil,
9784             [[Rule ]] .. rule_name ..
9785             [[ has already been updated by ]] .. extension_name)
9786         pattern = get_pattern
9787     end
9788     local accountable_pattern = { pattern, extension_name, rule_name }
9789     walkable_syntax[rule_name] = { accountable_pattern }
9790 end

```

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 of markdown.

```

9791     local special_characters = {}
9792     self.add_special_character = function(c)
9793         table.insert(special_characters, c)
9794         syntax.SpecialChar = S(table.concat(special_characters, ""))
9795     end
9796
9797     self.add_special_character("*")
9798     self.add_special_character("[")
9799     self.add_special_character("]")
9800     self.add_special_character("<")
9801     self.add_special_character("!")
9802     self.add_special_character("\\")

```

Add method `reader->initialize_named_group` that defines named groups with a default capture value.

```

9803     self.initialize_named_group = function(name, value)
9804         local pattern = Ct("")
9805         if value ~= nil then
9806             pattern = pattern / value
9807         end
9808         syntax.InitializeState = syntax.InitializeState
9809             * Cg(pattern, name)
9810     end

```

Add a named group for indentation.

```

9811     self.initialize_named_group("indent_info")

```

Apply syntax extensions.

```
9812     for _, extension in ipairs(extensions) do
9813         current_extension_name = extension.name
9814         extension.extend_writer(writer)
9815         extension.extend_reader(self)
9816     end
9817     current_extension_name = nil
```

If the `debugExtensions` option is enabled, serialize `walkable_syntax` to a JSON for debugging purposes.

```
9818     if options.debugExtensions then
9819         local sorted_lhs = {}
9820         for lhs, _ in pairs(walkable_syntax) do
9821             table.insert(sorted_lhs, lhs)
9822         end
9823         table.sort(sorted_lhs)
9824
9825         local output_lines = {"{"}
9826         for lhs_index, lhs in ipairs(sorted_lhs) do
9827             local encoded_lhs = util.encode_json_string(lhs)
9828             table.insert(output_lines, [{" "] .. encoded_lhs .. [{" ": []}]
9829             local rule = walkable_syntax[lhs]
9830             for rhs_index, rhs in ipairs(rule) do
9831                 local human_readable_rhs
9832                 if type(rhs) == "string" then
9833                     human_readable_rhs = rhs
9834                 else
9835                     local pattern_name
9836                     if rhs[3] then
9837                         pattern_name = rhs[3]
9838                     else
9839                         pattern_name = "Anonymous Pattern"
9840                     end
9841                     local extension_name = rhs[2]
9842                     human_readable_rhs = pattern_name .. [{" (}]
9843                         .. extension_name .. [{" (}]
9844                 end
9845                 local encoded_rhs
9846                 = util.encode_json_string(human_readable_rhs)
9847                 local output_line = [{" "] .. encoded_rhs
9848                 if rhs_index < #rule then
9849                     output_line = output_line .. ", "
9850                 end
9851                 table.insert(output_lines, output_line)
9852             end
9853             local output_line = [{" "]
9854             if lhs_index < #sorted_lhs then
```

```

9855         output_line = output_line .. ","
9856     end
9857     table.insert(output_lines, output_line)
9858 end
9859 table.insert(output_lines, "}")
9860
9861 local output = table.concat(output_lines, "\n")
9862 local output_filename = options.debugExtensionsFileName
9863 local output_file = assert(io.open(output_filename, "w"),
9864     [[Could not open file ]] .. output_filename
9865     .. [[ for writing]])
9866 assert(output_file:write(output))
9867 assert(output_file:close())
9868 end

```

Materialize `walkable_syntax` and merge it into `syntax` to produce the complete PEG grammar of markdown. Whenever a rule exists in both `walkable_syntax` and `syntax`, the rule from `walkable_syntax` overrides the rule from `syntax`.

```

9869     for lhs, rule in pairs(walkable_syntax) do
9870         syntax[lhs] = parsers.fail
9871         for _, rhs in ipairs(rule) do
9872             local pattern

```

Although the interface of the `reader->insert_pattern` method does not document this (see Section 2.1.2), we allow the `reader->insert_pattern` and `reader->update_rule` methods to insert not just PEG patterns, but also rule names that reference the PEG grammar of Markdown.

```

9873         if type(rhs) == "string" then
9874             pattern = V(rhs)
9875         else
9876             pattern = rhs[1]
9877             if type(pattern) == "string" then
9878                 pattern = V(pattern)
9879             end
9880         end
9881         syntax[lhs] = syntax[lhs] + pattern
9882     end
9883 end

```

Finalize the parser by reacting to options and by producing special parsers for difficult edge cases such as blocks nested in definition lists or inline content nested in link, note, and image labels.

```

9884     if options.underscores then
9885         self.add_special_character("_")
9886     end
9887
9888     if not options.codeSpans then

```

```

9889     syntax.Code = parsers.fail
9890 else
9891     self.add_special_character("`")
9892 end
9893
9894 if not options.html then
9895     syntax.DisplayHtml = parsers.fail
9896     syntax.InlineHtml = parsers.fail
9897     syntax.HtmlEntity = parsers.fail
9898 else
9899     self.add_special_character("&")
9900 end
9901
9902 if options.preserveTabs then
9903     options.stripIndent = false
9904 end
9905
9906 if not options.smartEllipses then
9907     syntax.Smart = parsers.fail
9908 else
9909     self.add_special_character(".")
9910 end
9911
9912 if not options.relativeReferences then
9913     syntax.AutoLinkRelativeReference = parsers.fail
9914 end
9915
9916 if options.contentLevel == "inline" then
9917     syntax[1] = "Inlines"
9918     syntax.Inlines = V("InitializeState")
9919         * parsers.Inline^0
9920         * ( parsers.spacing^0
9921             * parsers.eof / "" )
9922     syntax.Space = parsers.Space + parsers.blankline / writer.space
9923 end
9924
9925 local blocks_nested_t = util.table_copy(syntax)
9926 blocks_nested_t.ExpectedJekyllData = parsers.succeed
9927 parsers.blocks_nested = Ct(blocks_nested_t)
9928
9929 parsers.blocks = Ct(syntax)
9930
9931 local inlines_t = util.table_copy(syntax)
9932 inlines_t[1] = "Inlines"
9933 inlines_t.Inlines = V("InitializeState")
9934     * parsers.Inline^0
9935     * ( parsers.spacing^0

```



```

9936             * parsers.eof / "")
9937 parsers.inlines = Ct(inlines_t)
9938
9939 local inlines_no_inline_note_t = util.table_copy(inlines_t)
9940 inlines_no_inline_note_t.InlineNote = parsers.fail
9941 parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
9942
9943 local inlines_no_html_t = util.table_copy(inlines_t)
9944 inlines_no_html_t.DisplayHtml = parsers.fail
9945 inlines_no_html_t.InlineHtml = parsers.fail
9946 inlines_no_html_t.HtmlEntity = parsers.fail
9947 parsers.inlines_no_html = Ct(inlines_no_html_t)
9948
9949 local inlines_nbsp_t = util.table_copy(inlines_t)
9950 inlines_nbsp_t.Endline = parsers.NonbreakingEndline
9951 inlines_nbsp_t.Space = parsers.NonbreakingSpace
9952 parsers.inlines_nbsp = Ct(inlines_nbsp_t)
9953
9954 local inlines_no_link_or_emphasis_t = util.table_copy(inlines_t)
9955 inlines_no_link_or_emphasis_t.LinkAndEmph = parsers.fail
9956 inlines_no_link_or_emphasis_t.EndlineExceptions
9957     = parsers.EndlineExceptions - parsers.eof
9958 parsers.inlines_no_link_or_emphasis
9959     = Ct(inlines_no_link_or_emphasis_t)

```

Return a function that converts markdown string `input` into a plain T<sub>E</sub>X output and returns it..

```

9960     return function(input)

```

Unicode-normalize the input.

```

9961     if options.unicodeNormalization then
9962         local form = options.unicodeNormalizationForm
9963         if form == "nfc" then
9964             input = uni_algos.normalize.NFC(input)
9965         elseif form == "nfd" then
9966             input = uni_algos.normalize.NFD(input)
9967         elseif form == "nfkc" then
9968             input = uni_algos.normalize.NFKC(input)
9969         elseif form == "nfkd" then
9970             input = uni_algos.normalize.NFKD(input)
9971         else
9972             return writer.error(
9973                 format("Unknown normalization form %s", form))
9974         end
9975     end

```

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.

```

9976     input = input:gsub("\r\n?", "\n")
9977     if input:sub(-1) ~= "\n" then
9978         input = input .. "\n"
9979     end

```

Clear the table of references.

```

9980     references = {}
9981     local document = self.parser_functions.parse_blocks(input)
9982     local output = util.ropetostring(writer.document(document))

```

Remove block element / paragraph separators immediately followed by the output of `writer->undosep`, possibly interleaved by section ends. Then, remove any leftover output of `writer->undosep`.

```

9983     local undosep_start, undosep_end
9984     local potential_secend_start, secend_start
9985     local potential_sep_start, sep_start
9986     while true do
9987         -- find a `writer->undosep`
9988         undosep_start, undosep_end
9989         = output:find(writer.undosep_text, 1, true)
9990         if undosep_start == nil then break end
9991         -- skip any preceding section ends
9992         secend_start = undosep_start
9993         while true do
9994             potential_secend_start = secend_start - #writer.secend_text
9995             if potential_secend_start < 1
9996                 or output:sub(potential_secend_start,
9997                             secend_start - 1) ~= writer.secend_text
9998             then
9999                 break
10000         end
10001         secend_start = potential_secend_start
10002     end
10003     -- find an immediately preceding
10004     -- block element / paragraph separator
10005     sep_start = secend_start
10006     potential_sep_start = sep_start - #writer.interblocksep_text
10007     if potential_sep_start >= 1
10008         and output:sub(potential_sep_start,
10009                     sep_start - 1) == writer.interblocksep_text
10010     then
10011         sep_start = potential_sep_start
10012     else
10013         potential_sep_start = sep_start - #writer.paragraphsep_text
10014         if potential_sep_start >= 1
10015             and output:sub(potential_sep_start,
10016                         sep_start - 1) == writer.paragraphsep_text
10017         then

```

```

10018         sep_start = potential_sep_start
10019     end
10020 end
10021 -- remove `writer->undosep` and immediately preceding
10022 -- block element / paragraph separator
10023 output = output:sub(1, sep_start - 1)
10024     .. output:sub(secend_start, undosep_start - 1)
10025     .. output:sub(undosep_end + 1)
10026 end
10027 return output
10028 end
10029 end
10030 return self
10031 end

```

### 3.1.7 Built-In Syntax Extensions

Create `extensions` hash table that contains built-in 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.

```
10032 M.extensions = {}
```

#### 3.1.7.1 Bracketed Spans

The `extensions.bracketed_spans` function implements the Pandoc bracketed span syntax extension.

```

10033 M.extensions.bracketed_spans = function()
10034     return {
10035         name = "built-in bracketed_spans syntax extension",
10036         extend_writer = function(self)

```

Define `writer->span` as a function that will transform an input bracketed span `s` with attributes `attr` to the output format.

```

10037         function self.span(s, attr)
10038             if self.flatten_inlines then return s end
10039             return {"\\markdownRendererBracketedSpanAttributeContextBegin",
10040                 self.attributes(attr),
10041                 s,
10042                 "\\markdownRendererBracketedSpanAttributeContextEnd{}}"}
10043         end
10044     end, extend_reader = function(self)
10045         local parsers = self.parsers
10046         local writer = self.writer
10047
10048         local span_label = parsers.lbracket

```

```

10049         * (Cs((parsers.alphanumeric^1
10050             + parsers.inticks
10051             + parsers.autolink
10052             + V("InlineHtml")
10053             + ( parsers.backslash * parsers.backslash)
10054             + ( parsers.backslash
10055               * (parsers.lbracket + parsers.rbracket)
10056               + V("Space") + V("Endline")
10057             + (parsers.any
10058               - ( parsers.newline
10059                 + parsers.lbracket
10060                 + parsers.rbracket
10061                 + parsers.blankline^2))))^1)
10062         / self.parser_functions.parse_inlines)
10063         * parsers.rbracket
10064
10065         local Span = span_label
10066             * Ct(parsers.attributes)
10067             / writer.span
10068
10069         self.insert_pattern("Inline before LinkAndEmph",
10070             Span, "Span")
10071     end
10072 }
10073 end

```

### 3.1.7.2 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.

```

10074 M.extensions.citations = function(citation_nbsps)
10075     return {
10076         name = "built-in citations syntax extension",
10077         extend_writer = function(self)

```

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.

```

10078     function self.citations(text_cites, cites)
10079         local buffer = {}
10080         if self.flatten_inlines then
10081             for _,cite in ipairs(cites) do
10082                 if cite.prenote then
10083                     table.insert(buffer, {cite.prenote, " "})
10084                 end
10085                 table.insert(buffer, cite.name)
10086                 if cite.postnote then
10087                     table.insert(buffer, {" ", cite.postnote})
10088                 end
10089             end
10090         else
10091             table.insert(buffer,
10092                 {"\\markdownRenderer",
10093                 text_cites and "TextCite" or "Cite",
10094                 "{", #cites, "}"}))
10095             for _,cite in ipairs(cites) do
10096                 table.insert(buffer,
10097                     {cite.suppress_author and "-" or "+", "{",
10098                     cite.prenote or "", "}{" ,
10099                     cite.postnote or "", "}{" , cite.name, "}"}))
10100             end
10101         end
10102         return buffer
10103     end
10104 end, extend_reader = function(self)
10105     local parsers = self.parsers
10106     local writer = self.writer
10107
10108     local citation_chars
10109         = parsers.alphanumeric
10110         + S("#$%&~+<>~/_")
10111
10112     local citation_name
10113         = Cs(parsers.dash^-1) * parsers.at
10114         * Cs(citation_chars
10115             * ((( citation_chars
10116                 + parsers.internal_punctuation
10117                 - parsers.comma - parsers.semicolon)
10118             * -#(( parsers.internal_punctuation
10119                 - parsers.comma
10120                 - parsers.semicolon)^0

```

```

10121         * -( citation_chars
10122           + parsers.internal_punctuation
10123           - parsers.comma
10124           - parsers.semicolon)))^0
10125         * citation_chars)^-1)
10126
10127     local citation_body_prenote
10128         = Cs((parsers.alphanumeric^1
10129           + parsers.bracketed
10130           + parsers.inticks
10131           + parsers.autolink
10132           + V("InlineHtml")
10133           + V("Space") + V("EndlineNoSub"))
10134           + (parsers.anyescaped
10135             - ( parsers.newline
10136               + parsers.rbracket
10137               + parsers.blankline^2))
10138           - ( parsers.spnl
10139             * parsers.dash^-1
10140             * parsers.at))^1)
10141
10142     local citation_body_postnote
10143         = Cs((parsers.alphanumeric^1
10144           + parsers.bracketed
10145           + parsers.inticks
10146           + parsers.autolink
10147           + V("InlineHtml")
10148           + V("Space") + V("EndlineNoSub"))
10149           + (parsers.anyescaped
10150             - ( parsers.newline
10151               + parsers.rbracket
10152               + parsers.semicolon
10153               + parsers.blankline^2))
10154           - (parsers.spnl * parsers.rbracket))^1)
10155
10156     local citation_body_chunk
10157         = ( citation_body_prenote
10158           * parsers.spnlc_sep
10159           + Cc("")
10160           * parsers.spnlc
10161         )
10162         * citation_name
10163         * ( parsers.internal_punctuation
10164           - parsers.semicolon)^-1
10165         * ( parsers.spnlc / function(_) return end
10166           * citation_body_postnote
10167           + Cc(""))

```

```

10168         * parsers.spnlc
10169     )
10170
10171     local citation_body
10172         = citation_body_chunk
10173         * ( parsers.semicolon
10174           * parsers.spnlc
10175           * citation_body_chunk
10176         )^0
10177
10178     local citation_headless_body_postnote
10179         = Cs((parsers.alphanumeric^1
10180             + parsers.bracketed
10181             + parsers.inticks
10182             + parsers.autolink
10183             + V("InlineHtml")
10184             + V("Space") + V("Endline")
10185             + (parsers.anyescaped
10186               - ( parsers.newline
10187                 + parsers.rbracket
10188                 + parsers.at
10189                 + parsers.semicolon + parsers.blankline^2))
10190             - (parsers.spnl * parsers.rbracket))^0)
10191
10192     local citation_headless_body
10193         = citation_headless_body_postnote
10194         * ( parsers.semicolon
10195           * parsers.spnlc
10196           * citation_body_chunk
10197         )^0
10198
10199     local citations
10200         = function(text_cites, raw_cites)
10201         local function normalize(str)
10202             if str == "" then
10203                 str = nil
10204             else
10205                 str = (citation_nbsps and
10206                     self.parser_functions.parse_inlines_nbsp or
10207                     self.parser_functions.parse_inlines)(str)
10208             end
10209             return str
10210         end
10211
10212         local cites = {}
10213         for i = 1,#raw_cites,4 do
10214             cites[#cites+1] = {

```

```

10215         prenote = normalize(raw_cites[i]),
10216         suppress_author = raw_cites[i+1] == "-",
10217         name = writer.identifier(raw_cites[i+2]),
10218         postnote = normalize(raw_cites[i+3]),
10219     }
10220     end
10221     return writer.citations(text_cites, cites)
10222 end
10223
10224 local TextCitations
10225     = Ct((parsers.spnlc
10226     * Cc("")
10227     * citation_name
10228     * ((parsers.spnlc
10229     * parsers.lbracket
10230     * citation_headless_body
10231     * parsers.rbracket) + Cc("")))~1)
10232 / function(raw_cites)
10233     return citations(true, raw_cites)
10234 end
10235
10236 local ParenthesizedCitations
10237     = Ct((parsers.spnlc
10238     * parsers.lbracket
10239     * citation_body
10240     * parsers.rbracket)^1)
10241 / function(raw_cites)
10242     return citations(false, raw_cites)
10243 end
10244
10245 local Citations = TextCitations + ParenthesizedCitations
10246
10247 self.insert_pattern("Inline before LinkAndEmph",
10248     Citations, "Citations")
10249
10250 self.add_special_character("@")
10251 self.add_special_character("-")
10252 end
10253 }
10254 end

```

### 3.1.7.3 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.

```

10255 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.

```

10256 local languages_json = (function()
10257     local base, prev, curr
10258     for _, pathname in ipairs{kpse.lookup(language_map,
10259                                     {all=true})} do
10260         local file = io.open(pathname, "r")
10261         if not file then goto continue end
10262         local input = assert(file:read("*a"))
10263         assert(file:close())
10264         local json = input:gsub('("[^\n]-"):','[%1]=')
10265         curr = load("_ENV = {}; return "..json")()
10266         if type(curr) == "table" then
10267             if base == nil then
10268                 base = curr
10269             else
10270                 setmetatable(prev, { __index = curr })
10271             end
10272             prev = curr
10273         end
10274         ::continue::
10275     end
10276     return base or {}
10277 end)()
10278
10279 return {
10280     name = "built-in content_blocks syntax extension",
10281     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.

```

10282     function self.contentblock(src,suf,type,tit)
10283         if not self.is_writing then return "" end
10284         src = src..".."..suf
10285         suf = suf:lower()
10286         if type == "onlineimage" then
10287             return {"\\markdownRendererContentBlockOnlineImage{"..suf.."},"",
10288                 {"",self.string(src),"}",
10289                 {"",self.uri(src),"}",
10290                 {"",self.string(tit or ""),"}"}
10291         elseif languages_json[suf] then
10292             return {"\\markdownRendererContentBlockCode{"..suf.."},"",
10293                 {"",self.string(languages_json[suf]),"}",

```

```

10294         {"",self.string(src),""},
10295         {"",self.uri(src),""},
10296         {"",self.string(tit or ""),""}
10297     else
10298         return {"\\markdownRendererContentBlock{"",suf,""},
10299             {"",self.string(src),""},
10300             {"",self.uri(src),""},
10301             {"",self.string(tit or ""),""}
10302     end
10303 end
10304 end, extend_reader = function(self)
10305     local parsers = self.parsers
10306     local writer = self.writer
10307
10308     local contentblock_tail
10309         = parsers.optionaltitle
10310         * (parsers.newline + parsers.eof)
10311
10312     -- case insensitive online image suffix:
10313     local onlineimagesuffix
10314         = (function(...)
10315             local parser = nil
10316             for _, suffix in ipairs({...}) do
10317                 local pattern=nil
10318                 for i=1,#suffix do
10319                     local char=suffix:sub(i,i)
10320                     char = S(char:lower()..char:upper())
10321                     if pattern == nil then
10322                         pattern = char
10323                     else
10324                         pattern = pattern * char
10325                     end
10326                 end
10327                 if parser == nil then
10328                     parser = pattern
10329                 else
10330                     parser = parser + pattern
10331                 end
10332             end
10333             return parser
10334         end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
10335
10336     -- online image url for iA Writer content blocks with
10337     -- mandatory suffix, allowing nested brackets:
10338     local onlineimageurl
10339         = (parsers.less
10340         * Cs((parsers.anyescaped

```

```

10341         - parsers.more
10342         - parsers.spacing
10343         - #(parsers.period
10344           * onlineimagesuffix
10345           * parsers.more
10346           * contentblock_tail))^0)
10347     * parsers.period
10348     * Cs(onlineimagesuffix)
10349     * parsers.more
10350     + (Cs((parsers.inparens
10351       + (parsers.anyescaped
10352         - parsers.spacing
10353         - parsers.rparent
10354         - #(parsers.period
10355           * onlineimagesuffix
10356           * contentblock_tail))))^0)
10357     * parsers.period
10358     * Cs(onlineimagesuffix)
10359   ) * Cc("onlineimage")
10360
10361   -- filename for iA Writer content blocks with mandatory suffix:
10362   local localfilepath
10363     = parsers.slash
10364     * Cs((parsers.anyescaped
10365       - parsers.tab
10366       - parsers.newline
10367       - #(parsers.period
10368         * parsers.alphanumeric^1
10369         * contentblock_tail))^1)
10370     * parsers.period
10371     * Cs(parsers.alphanumeric^1)
10372     * Cc("localfile")
10373
10374   local ContentBlock
10375     = parsers.check_trail_no_rem
10376     * (localfilepath + onlineimageurl)
10377     * contentblock_tail
10378     / writer.contentblock
10379
10380   self.insert_pattern("Block before Blockquote",
10381     ContentBlock, "ContentBlock")
10382 end
10383 }
10384 end

```

### 3.1.7.4 Definition Lists

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

```
10385 M.extensions.definition_lists = function(tight_lists)
10386   return {
10387     name = "built-in definition_lists syntax extension",
10388     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.

```
10389     local function dliitem(term, defs)
10390       local retVal = {"\\markdownRendererDlItem{",term,"}"}
10391       for _, def in ipairs(defs) do
10392         retVal[#retVal+1]
10393           = {"\\markdownRendererDlDefinitionBegin ",def,
10394             "\\markdownRendererDlDefinitionEnd "}
10395       end
10396       retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
10397       return retVal
10398     end
10399
10400     function self.definitionlist(items,tight)
10401       if not self.is_writing then return "" end
10402       local buffer = {}
10403       for _,item in ipairs(items) do
10404         buffer[#buffer + 1] = dliitem(item.term, item.definitions)
10405       end
10406       if tight and tight_lists then
10407         return {"\\markdownRendererDlBeginTight\\n", buffer,
10408             "\\n\\markdownRendererDlEndTight"}
10409       else
10410         return {"\\markdownRendererDlBegin\\n", buffer,
10411             "\\n\\markdownRendererDlEnd"}
10412       end
10413     end
10414     end, extend_reader = function(self)
10415       local parsers = self.parsers
10416       local writer = self.writer
10417
10418       local defstartchar = S("~:")
10419
10420       local defstart
10421         = parsers.check_trail_length(0) * defstartchar
10422         * #parsers.spacing
10423         * (parsers.tab + parsers.space^-3)
```

```

10424     + parsers.check_trail_length(1)
10425     * defstartchar * #parsers.spacing
10426     * (parsers.tab + parsers.space^-2)
10427     + parsers.check_trail_length(2)
10428     * defstartchar * #parsers.spacing
10429     * (parsers.tab + parsers.space^-1)
10430     + parsers.check_trail_length(3)
10431     * defstartchar * #parsers.spacing
10432
10433     local indented_line
10434     = (parsers.check_minimal_indent / "")
10435     * parsers.check_code_trail * parsers.line
10436
10437     local blank
10438     = parsers.check_minimal_blank_indent_and_any_trail
10439     * parsers.optionalspace * parsers.newline
10440
10441     local dlchunk = Cs(parsers.line * (indented_line - blank)^0)
10442
10443     local indented_blocks = function(bl)
10444     return Cs( bl
10445     * (blank^1 * (parsers.check_minimal_indent / ""))
10446     * parsers.check_code_trail * -parsers.blankline * bl)^0
10447     * (blank^1 + parsers.eof))
10448     end
10449
10450     local function definition_list_item(term, defs, _)
10451     return { term = self.parser_functions.parse_inlines(term),
10452     definitions = defs }
10453     end
10454
10455     local DefinitionListItemLoose
10456     = C(parsers.line) * blank^0
10457     * Ct((parsers.check_minimal_indent * (defstart
10458     * indented_blocks(dlchunk)
10459     / self.parser_functions.parse_blocks_nested))^1)
10460     * Cc(false) / definition_list_item
10461
10462     local DefinitionListItemTight
10463     = C(parsers.line)
10464     * Ct((parsers.check_minimal_indent * (defstart * dlchunk
10465     / self.parser_functions.parse_blocks_nested))^1)
10466     * Cc(true) / definition_list_item
10467
10468     local DefinitionList
10469     = ( Ct(DefinitionListItemLoose^1) * Cc(false)
10470     + Ct(DefinitionListItemTight^1)

```

```

10471         * (blank~0
10472         * -DefinitionListItemLoose * Cc(true))
10473         ) / writer.definitionlist
10474
10475         self.insert_pattern("Block after Heading",
10476                             DefinitionList, "DefinitionList")
10477     end
10478 }
10479 end

```

### 3.1.7.5 Fancy Lists

The `extensions.fancy_lists` function implements the Pandoc fancy list syntax extension.

```

10480 M.extensions.fancy_lists = function()
10481   return {
10482     name = "built-in fancy_lists syntax extension",
10483     extend_writer = function(self)
10484       local options = self.options
10485

```

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.

```

10486     function self.fancylist(items,tight,startnum,numstyle,numdelim)
10487       if not self.is_writing then return "" end
10488       local buffer = {}
10489       local num = startnum

```

```

10490     for _,item in ipairs(items) do
10491         if item ~= "" then
10492             buffer[#buffer + 1] = self.fancyitem(item,num)
10493         end
10494         if num ~= nil and item ~= "" then
10495             num = num + 1
10496         end
10497     end
10498     local contents = util.intersperse(buffer,"\n")
10499     if tight and options.tightLists then
10500         return {"\\markdownRendererFancyOlBeginTight{",
10501             numstyle,"}{",numdelim,"}",contents,
10502             "\\n\\markdownRendererFancyOlEndTight "}
10503     else
10504         return {"\\markdownRendererFancyOlBegin{",
10505             numstyle,"}{",numdelim,"}",contents,
10506             "\\n\\markdownRendererFancyOlEnd "}
10507     end
10508 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.

```

10509     function self.fancyitem(s,num)
10510         if num ~= nil then
10511             return {"\\markdownRendererFancyOlItemWithNumber{",num,"}",s,
10512                 "\\markdownRendererFancyOlItemEnd "}
10513         else
10514             return {"\\markdownRendererFancyOlItem ",s,
10515                 "\\markdownRendererFancyOlItemEnd "}
10516         end
10517     end
10518 end, extend_reader = function(self)
10519     local parsers = self.parsers
10520     local options = self.options
10521     local writer = self.writer
10522
10523     local function combine_markers_and_delims(markers, delims)
10524         local markers_table = {}
10525         for _,marker in ipairs(markers) do
10526             local start_marker
10527             local continuation_marker
10528             if type(marker) == "table" then
10529                 start_marker = marker[1]
10530                 continuation_marker = marker[2]
10531             else
10532                 start_marker = marker

```

```

10533         continuation_marker = marker
10534     end
10535     for _,delim in ipairs(delims) do
10536         table.insert(markers_table,
10537             {start_marker, continuation_marker, delim})
10538     end
10539 end
10540 return markers_table
10541 end
10542
10543 local function join_table_with_func(func, markers_table)
10544     local pattern = func(table.unpack(markers_table[1]))
10545     for i = 2, #markers_table do
10546         pattern = pattern + func(table.unpack(markers_table[i]))
10547     end
10548     return pattern
10549 end
10550
10551 local lowercase_letter_marker = R("az")
10552 local uppercase_letter_marker = R("AZ")
10553
10554 local roman_marker = function(chars)
10555     local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])
10556     local l, x, v, i
10557         = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])
10558     return m^-3
10559         * (c*m + c*d + d^-1 * c^-3)
10560         * (x*c + x*l + l^-1 * x^-3)
10561         * (i*x + i*v + v^-1 * i^-3)
10562 end
10563
10564 local lowercase_roman_marker
10565     = roman_marker({"m", "d", "c", "l", "x", "v", "i"})
10566 local uppercase_roman_marker
10567     = roman_marker({"M", "D", "C", "L", "X", "V", "I"})
10568
10569 local lowercase_opening_roman_marker = P("i")
10570 local uppercase_opening_roman_marker = P("I")
10571
10572 local digit_marker = parsers.dig * parsers.dig^-8
10573
10574 local markers = {
10575     {lowercase_opening_roman_marker, lowercase_roman_marker},
10576     {uppercase_opening_roman_marker, uppercase_roman_marker},
10577     lowercase_letter_marker,
10578     uppercase_letter_marker,
10579     lowercase_roman_marker,

```



```

10580     uppercase_roman_marker,
10581     digit_marker
10582 }
10583
10584 local delims = {
10585     parsers.period,
10586     parsers.rparent
10587 }
10588
10589 local markers_table = combine_markers_and_delims(markers, delims)
10590
10591 local function enumerator(start_marker, _,
10592     delimiter_type, interrupting)
10593     local delimiter_range
10594     local allowed_end
10595     if interrupting then
10596         delimiter_range = P("1")
10597         allowed_end = C(parsers.spacechar^1) * #parsers.linechar
10598     else
10599         delimiter_range = start_marker
10600         allowed_end = C(parsers.spacechar^1)
10601             + #(parsers.newline + parsers.eof)
10602     end
10603
10604     return parsers.check_trail
10605         * Ct(C(delimiter_range) * C(delimiter_type))
10606         * allowed_end
10607 end
10608
10609 local starter = join_table_with_func(enumerator, markers_table)
10610
10611 local TightListItem = function(starter)
10612     return parsers.add_indent(starter, "li")
10613         * parsers.indented_content_tight
10614 end
10615
10616 local LooseListItem = function(starter)
10617     return parsers.add_indent(starter, "li")
10618         * parsers.indented_content_loose
10619         * remove_indent("li")
10620 end
10621
10622 local function roman2number(roman)
10623     local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100,
10624         ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
10625     local numeral = 0
10626

```

```

10627     local i = 1
10628     local len = string.len(roman)
10629     while i < len do
10630         local z1, z2 = romans[ string.sub(roman, i, i) ],
10631             romans[ string.sub(roman, i+1, i+1) ]
10632         if z1 < z2 then
10633             numeral = numeral + (z2 - z1)
10634             i = i + 2
10635         else
10636             numeral = numeral + z1
10637             i = i + 1
10638         end
10639     end
10640     if i <= len then
10641         numeral = numeral + romans[ string.sub(roman,i,i) ]
10642     end
10643     return numeral
10644 end
10645
10646 local function sniffstyle(numstr, delimend)
10647     local numdelim
10648     if delimend == ")" then
10649         numdelim = "OneParen"
10650     elseif delimend == "." then
10651         numdelim = "Period"
10652     else
10653         numdelim = "Default"
10654     end
10655
10656     local num
10657     num = numstr:match("^([I])$")
10658     if num then
10659         return roman2number(num), "UpperRoman", numdelim
10660     end
10661     num = numstr:match("^([i])$")
10662     if num then
10663         return roman2number(string.upper(num)), "LowerRoman", numdelim
10664     end
10665     num = numstr:match("^([A-Z])$")
10666     if num then
10667         return string.byte(num) - string.byte("A") + 1,
10668             "UpperAlpha", numdelim
10669     end
10670     num = numstr:match("^([a-z])$")
10671     if num then
10672         return string.byte(num) - string.byte("a") + 1,
10673             "LowerAlpha", numdelim

```

```

10674     end
10675     num = numstr:match("^([IVXLCDM]+)")
10676     if num then
10677         return roman2number(num), "UpperRoman", numdelim
10678     end
10679     num = numstr:match("^([ivxlc dm]+)")
10680     if num then
10681         return roman2number(string.upper(num)), "LowerRoman", numdelim
10682     end
10683     return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
10684 end
10685
10686 local function fancylist(items,tight,start)
10687     local startnum, numstyle, numdelim
10688     = sniffstyle(start[2][1], start[2][2])
10689     return writer.fancylist(items,tight,
10690                             options.startNumber and startnum or 1,
10691                             numstyle or "Decimal",
10692                             numdelim or "Default")
10693 end
10694
10695 local FancyListOfType
10696 = function(start_marker, continuation_marker, delimiter_type)
10697     local enumerator_start
10698     = enumerator(start_marker, continuation_marker,
10699                 delimiter_type)
10700     local enumerator_cont
10701     = enumerator(continuation_marker, continuation_marker,
10702                 delimiter_type)
10703     return Cg(enumerator_start, "listtype")
10704     * (Ct( TightListItem(Cb("listtype"))
10705         * ((parsers.check_minimal_indent / ""))
10706         * TightListItem(enumerator_cont))^0)
10707     * Cc(true)
10708     * -#((parsers.conditionally_indented_blankline^0 / ""))
10709     * parsers.check_minimal_indent * enumerator_cont)
10710     + Ct( LooseListItem(Cb("listtype"))
10711         * ((parsers.conditionally_indented_blankline^0 / ""))
10712         * (parsers.check_minimal_indent / ""))
10713         * LooseListItem(enumerator_cont))^0)
10714     * Cc(false)
10715     ) * Ct(Cb("listtype")) / fancylist
10716 end
10717
10718 local FancyList
10719 = join_table_with_func(FancyListOfType, markers_table)
10720

```

```

10721     local Endline = parsers.newline
10722                 * (parsers.check_minimal_indent
10723                   * -parsers.EndlineExceptions
10724                   + parsers.check_optional_indent
10725                   * -parsers.EndlineExceptions
10726                   * -starter)
10727                 * parsers.spacechar^0
10728                 / writer.soft_line_break
10729
10730     self.update_rule("OrderedList", FancyList)
10731     self.update_rule("Endline", Endline)
10732 end
10733 }
10734 end

```

### 3.1.7.6 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 a blank line between a paragraph and the following fenced code block.

When the `allow_attributes` option is `true`, the syntax extension permits attributes following the infostring. When the `allow_raw_blocks` option is `true`, the syntax extension permits the specification of raw blocks using the Pandoc raw attribute syntax extension.

```

10735 M.extensions.fenced_code = function(blank_before_code_fence,
10736                                     allow_attributes,
10737                                     allow_raw_blocks)
10738   return {
10739     name = "built-in fenced_code syntax extension",
10740     extend_writer = function(self)
10741       local options = self.options
10742

```

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

```

10743     function self.fencedCode(s, i, attr)
10744       if not self.is_writing then return "" end
10745       s = s:gsub("\n$", "")
10746       local buf = {}
10747       if attr ~= nil then
10748         table.insert(buf,
10749                     {"\markdownRendererFencedCodeAttributeContextBegin",
10750                      self.attributes(attr)})
10751       end
10752       local name = util.cache_verbatim(options.cacheDir, s)
10753       table.insert(buf,

```

```

10754         {"\\markdownRendererInputFencedCode{" ,
10755         name,"}{" ,self.string(i),"}{" ,self.infostring(i),"}")
10756     if attr ~= nil then
10757         table.insert(buf,
10758         "\\markdownRendererFencedCodeAttributeContextEnd{")
10759     end
10760     return buf
10761 end
10762

```

Define `writer->rawBlock` as a function that will transform an input raw block `s` with the raw attribute `attr` to the output format.

```

10763     if allow_raw_blocks then
10764         function self.rawBlock(s, attr)
10765             if not self.is_writing then return "" end
10766             s = s:gsub("\\n$", "")
10767             local name = util.cache_verbatim(options.cacheDir, s)
10768             return {"\\markdownRendererInputRawBlock{" ,
10769             name,"}{" , self.string(attr),"}"}
10770         end
10771     end
10772 end, extend_reader = function(self)
10773     local parsers = self.parsers
10774     local writer = self.writer
10775
10776     local function captures_geq_length(_,i,a,b)
10777         return #a >= #b and i
10778     end
10779
10780     local function strip_enclosing_whitespaces(str)
10781         return str:gsub("^%s*(.)%s*$", "%1")
10782     end
10783
10784     local tilde_infostring = Cs(Cs((V("HtmlEntity")
10785         + parsers.anyescaped
10786         - parsers.newline)^0)
10787         / strip_enclosing_whitespaces)
10788
10789     local backtick_infostring
10790     = Cs( Cs((V("HtmlEntity")
10791         + ( -#(parsers.backslash * parsers.backtick)
10792         * parsers.anyescaped)
10793         - parsers.newline
10794         - parsers.backtick)^0)
10795         / strip_enclosing_whitespaces)
10796
10797     local fenceindent

```

```

10798
10799     local function has_trail(indent_table)
10800         return indent_table ~= nil and
10801             indent_table.trail ~= nil and
10802             next(indent_table.trail) ~= nil
10803     end
10804
10805     local function has_indents(indent_table)
10806         return indent_table ~= nil and
10807             indent_table.indents ~= nil and
10808             next(indent_table.indents) ~= nil
10809     end
10810
10811     local function get_last_indent_name(indent_table)
10812         if has_indents(indent_table) then
10813             return indent_table.indents[#indent_table.indents].name
10814         end
10815     end
10816
10817     local count_fenced_start_indent =
10818         function(_, _, indent_table, trail)
10819             local last_indent_name = get_last_indent_name(indent_table)
10820             fenceindent = 0
10821             if last_indent_name ~= "li" then
10822                 fenceindent = #trail
10823             end
10824             return true
10825         end
10826
10827     local fencehead = function(char, infostring)
10828         return Cmt( Cb("indent_info")
10829             * parsers.check_trail, count_fenced_start_indent)
10830             * Cg(char^3, "fencelength")
10831             * parsers.optionalspace
10832             * infostring
10833             * (parsers.newline + parsers.eof)
10834     end
10835
10836     local fencetail = function(char)
10837         return parsers.check_trail_no_rem
10838             * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
10839             * parsers.optionalspace * (parsers.newline + parsers.eof)
10840             + parsers.eof
10841     end
10842
10843     local process_fenced_line =
10844         function(s, i, -- luacheck: ignore s i

```

```

10845         indent_table, line_content, is_blank)
10846     local remainder = ""
10847     if has_trail(indent_table) then
10848         remainder = indent_table.trail.internal_remainder
10849     end
10850
10851     if is_blank
10852         and get_last_indent_name(indent_table) == "li" then
10853         remainder = ""
10854     end
10855
10856     local str = remainder .. line_content
10857     local index = 1
10858     local remaining = fenceindent
10859
10860     while true do
10861         local c = str:sub(index, index)
10862         if c == " " and remaining > 0 then
10863             remaining = remaining - 1
10864             index = index + 1
10865         elseif c == "\t" and remaining > 3 then
10866             remaining = remaining - 4
10867             index = index + 1
10868         else
10869             break
10870         end
10871     end
10872
10873     return true, str:sub(index)
10874 end
10875
10876 local fencedline = function(char)
10877     return Cmt( Cb("indent_info")
10878         * C(parsers.line - fencetail(char))
10879         * Cc(false), process_fenced_line)
10880 end
10881
10882 local blankfencedline
10883     = Cmt( Cb("indent_info")
10884         * C(parsers.blankline)
10885         * Cc(true), process_fenced_line)
10886
10887 local TildeFencedCode
10888     = fencehead(parsers.tilde, tilde_infostring)
10889     * Cs(( parsers.check_minimal_blank_indent / ""))
10890         * blankfencedline
10891         + ( parsers.check_minimal_indent / ""))

```

```

10892         * fencedline(parsers.tilde)^0)
10893     * ( (parsers.check_minimal_indent / "")
10894         * fencetail(parsers.tilde) + parsers.succeed)
10895
10896     local BacktickFencedCode
10897         = fencehead(parsers.backtick, backtick_infostring)
10898         * Cs(( (parsers.check_minimal_blank_indent / "")
10899             * blankfencedline
10900             + (parsers.check_minimal_indent / "")
10901             * fencedline(parsers.backtick))^0)
10902         * ( (parsers.check_minimal_indent / "")
10903             * fencetail(parsers.backtick) + parsers.succeed)
10904
10905     local infostring_with_attributes
10906         = Ct(C((parsers.linechar
10907             - ( parsers.optionalspace
10908                 * parsers.attributes))^0)
10909             * parsers.optionalspace
10910             * Ct(parsers.attributes))
10911
10912     local FencedCode
10913         = ((TildeFencedCode + BacktickFencedCode)
10914         / function(infostring, code)
10915             local expanded_code = self.expandtabs(code)
10916
10917             if allow_raw_blocks then
10918                 local raw_attr = lpeg.match(parsers.raw_attribute,
10919                                         infostring)
10920
10921                 if raw_attr then
10922                     return writer.rawBlock(expanded_code, raw_attr)
10923                 end
10924             end
10925
10926             local attr = nil
10927             if allow_attributes then
10928                 local match = lpeg.match(infostring_with_attributes,
10929                                         infostring)
10930
10931                 if match then
10932                     infostring, attr = table.unpack(match)
10933                 end
10934             end
10935             return writer.fencedCode(expanded_code, infostring, attr)
10936         end)
10937
10938     self.insert_pattern("Block after Verbatim",
10939                       FencedCode, "FencedCode")

```



```

10939     local fencestart
10940     if blank_before_code_fence then
10941         fencestart = parsers.fail
10942     else
10943         fencestart = fencehead(parsers.backtick, backtick_infostring)
10944             + fencehead(parsers.tilde, tilde_infostring)
10945     end
10946
10947     self.update_rule("EndlineExceptions", function(previous_pattern)
10948         if previous_pattern == nil then
10949             previous_pattern = parsers.EndlineExceptions
10950         end
10951         return previous_pattern + fencestart
10952     end)
10953
10954     self.add_special_character("`")
10955     self.add_special_character("~")
10956 end
10957 }
10958 end

```

### 3.1.7.7 Fenced Divs

The `extensions.fenced_divs` function implements the Pandoc fenced div syntax extension. When the `blank_before_div_fence` parameter is `true`, the syntax extension requires a blank line between a paragraph and the following fenced code block.

```

10959 M.extensions.fenced_divs = function(blank_before_div_fence)
10960     return {
10961         name = "built-in fenced_divs syntax extension",
10962         extend_writer = function(self)

```

Define `writer->div_begin` as a function that will transform the beginning of an input fenced div with with attributes `attributes` to the output format.

```

10963         function self.div_begin(attributes)
10964             local start_output
10965             = {"\\markdownRendererFencedDivAttributeContextBegin\n",
10966                 self.attributes(attributes)}
10967             local end_output
10968             = {"\\markdownRendererFencedDivAttributeContextEnd{}}"}
10969             return self.push_attributes(
10970                 "div", attributes, start_output, end_output)
10971         end

```

Define `writer->div_end` as a function that will produce the end of a fenced div in the output format.

```

10972         function self.div_end()

```

```

10973     return self.pop_attributes("div")
10974     end
10975 end, extend_reader = function(self)
10976     local parsers = self.parsers
10977     local writer = self.writer

```

Define basic patterns for matching the opening and the closing tag of a div.

```

10978     local fenced_div_infostring
10979         = C((parsers.linechar
10980             - ( parsers.spacechar^1
10981               * parsers.colon^1))^1)
10982
10983     local fenced_div_begin = parsers.nonindentspace
10984         * parsers.colon^3
10985         * parsers.optionalspace
10986         * fenced_div_infostring
10987         * ( parsers.spacechar^1
10988           * parsers.colon^1)^0
10989         * parsers.optionalspace
10990         * (parsers.newline + parsers.eof)
10991
10992     local fenced_div_end = parsers.nonindentspace
10993         * parsers.colon^3
10994         * parsers.optionalspace
10995         * (parsers.newline + parsers.eof)

```

Initialize a named group named `fenced_div_level` for tracking how deep we are nested in divs and the named group `fenced_div_num_opening_indents` for tracking the indent of the starting div fence. The former named group is immutable and should roll back properly when we fail to match a fenced div. The latter is mutable and may contain items from unsuccessful matches on top. However, we always know how many items at the head of the latter we can trust by consulting the former.

```

10996     self.initialize_named_group("fenced_div_level", "0")
10997     self.initialize_named_group("fenced_div_num_opening_indents")
10998
10999     local function increment_div_level()
11000         local push_indent_table =
11001             function(s, i, indent_table, -- luacheck: ignore s i
11002                 fenced_div_num_opening_indents, fenced_div_level)
11003                 fenced_div_level = tonumber(fenced_div_level) + 1
11004                 local num_opening_indents = 0
11005                 if indent_table.indents ~= nil then
11006                     num_opening_indents = #indent_table.indents
11007                 end
11008                 fenced_div_num_opening_indents[fenced_div_level]
11009                     = num_opening_indents
11010                 return true, fenced_div_num_opening_indents

```

```

11011         end
11012
11013     local increment_level =
11014         function(s, i, fenced_div_level) -- luacheck: ignore s i
11015             fenced_div_level = tonumber(fenced_div_level) + 1
11016             return true, tostring(fenced_div_level)
11017         end
11018
11019     return Cg( Cmt( Cb("indent_info")
11020                 * Cb("fenced_div_num_opening_indents")
11021                 * Cb("fenced_div_level"), push_indent_table)
11022             , "fenced_div_num_opening_indents")
11023         * Cg( Cmt( Cb("fenced_div_level"), increment_level)
11024             , "fenced_div_level")
11025     end
11026
11027     local function decrement_div_level()
11028         local pop_indent_table =
11029             function(s, i, -- luacheck: ignore s i
11030                 fenced_div_indent_table, fenced_div_level)
11031                 fenced_div_level = tonumber(fenced_div_level)
11032                 fenced_div_indent_table[fenced_div_level] = nil
11033                 return true, tostring(fenced_div_level - 1)
11034             end
11035
11036     return Cg( Cmt( Cb("fenced_div_num_opening_indents")
11037                 * Cb("fenced_div_level"), pop_indent_table)
11038             , "fenced_div_level")
11039     end
11040
11041
11042     local non_fenced_div_block
11043         = parsers.check_minimal_indent * V("Block")
11044         - parsers.check_minimal_indent_and_trail * fenced_div_end
11045
11046     local non_fenced_div_paragraph
11047         = parsers.check_minimal_indent * V("Paragraph")
11048         - parsers.check_minimal_indent_and_trail * fenced_div_end
11049
11050     local blank = parsers.minimally_indented_blank
11051
11052     local block_separated = parsers.block_sep_group(blank)
11053         * non_fenced_div_block
11054
11055     local loop_body_pair
11056         = parsers.create_loop_body_pair(block_separated,
11057                                         non_fenced_div_paragraph,

```

```

11058             parsers.block_sep_group(blank),
11059             parsers.par_sep_group(blank))
11060
11061     local content_loop = ( non_fenced_div_block
11062                          * loop_body_pair.block^0
11063                          + non_fenced_div_paragraph
11064                          * block_separated
11065                          * loop_body_pair.block^0
11066                          + non_fenced_div_paragraph
11067                          * loop_body_pair.par^0)
11068     * blank^0
11069
11070     local FencedDiv = fenced_div_begin
11071     / function (infostring)
11072     local attr
11073     = lpeg.match(Ct(parsers.attributes),
11074                infostring)
11075     if attr == nil then
11076     attr = {"." .. infostring}
11077     end
11078     return attr
11079     end
11080     / writer.div_begin
11081     * increment_div_level()
11082     * parsers.skipblanklines
11083     * Ct(content_loop)
11084     * parsers.minimally_indented_blank^0
11085     * parsers.check_minimal_indent_and_trail
11086     * fenced_div_end
11087     * decrement_div_level()
11088     * (Cc("") / writer.div_end)
11089
11090     self.insert_pattern("Block after Verbatim",
11091                       FencedDiv, "FencedDiv")
11092
11093     self.add_special_character(":")
11094

```

If the `blank_before_div_fence` parameter is `false`, we will have the closing div at the beginning of a line break the current paragraph if we are currently nested in a div and the indentation matches the opening div fence.

```

11095     local function is_inside_div()
11096     local check_div_level =
11097     function(s, i, fenced_div_level) -- luacheck: ignore s i
11098     fenced_div_level = tonumber(fenced_div_level)
11099     return fenced_div_level > 0
11100     end

```

```

11101
11102     return Cmt(Cb("fenced_div_level"), check_div_level)
11103 end
11104
11105 local function check_indent()
11106     local compare_indent =
11107         function(s, i, indent_table, -- luacheck: ignore s i
11108             fenced_div_num_opening_indents, fenced_div_level)
11109             fenced_div_level = tonumber(fenced_div_level)
11110             local num_current_indents
11111                 = ( indent_table.current_line_indents ~= nil and
11112                   #indent_table.current_line_indents) or 0
11113             local num_opening_indents
11114                 = fenced_div_num_opening_indents[fenced_div_level]
11115             return num_current_indents == num_opening_indents
11116         end
11117
11118     return Cmt( Cb("indent_info")
11119               * Cb("fenced_div_num_opening_indents")
11120               * Cb("fenced_div_level"), compare_indent)
11121 end
11122
11123 local fencestart = is_inside_div()
11124                 * fenced_div_end
11125                 * check_indent()
11126
11127 if not blank_before_div_fence then
11128     self.update_rule("EndlineExceptions", function(previous_pattern)
11129         if previous_pattern == nil then
11130             previous_pattern = parsers.EndlineExceptions
11131         end
11132         return previous_pattern + fencestart
11133     end)
11134 end
11135 end
11136 }
11137 end

```

### 3.1.7.8 Header Attributes

The `extensions.header_attributes` function implements the Pandoc header attribute syntax extension.

```

11138 M.extensions.header_attributes = function()
11139     return {
11140         name = "built-in header_attributes syntax extension",
11141         extend_writer = function()
11142         end, extend_reader = function(self)

```

```

11143     local parsers = self.parsers
11144     local writer = self.writer
11145
11146     local function strip_atx_end(s)
11147         return s:gsub("%s+##%s*$", "")
11148     end
11149
11150     local AtxHeading = Cg(parsers.heading_start, "level")
11151         * parsers.optionalspace
11152         * (C(((parsers.linechar
11153             - (parsers.attributes
11154                 * parsers.optionalspace
11155                 * parsers.newline))
11156             * (parsers.linechar
11157                 - parsers.lbrace)^0)^1)
11158             / strip_atx_end
11159             / parsers.parse_heading_text)
11160         * Cg(Ct(parsers.newline
11161             + (parsers.attributes
11162                 * parsers.optionalspace
11163                 * parsers.newline)), "attributes")
11164         * Cb("level")
11165         * Cb("attributes")
11166         / writer.heading
11167
11168     local function strip_trailing_spaces(s)
11169         return s:gsub("%s*$", "")
11170     end
11171
11172     local heading_line = (parsers.linechar
11173         - (parsers.attributes
11174             * parsers.optionalspace
11175             * parsers.newline))^1
11176         - parsers.thematic_break_lines
11177
11178     local heading_text
11179         = heading_line
11180         * ( (V("Endline") / "\n")
11181             * (heading_line - parsers.heading_level))^0
11182         * parsers.newline^-1
11183
11184     local SetextHeading
11185         = parsers.freeze_trail * parsers.check_trail_no_rem
11186         * #(heading_text
11187             * (parsers.attributes
11188                 * parsers.optionalspace
11189                 * parsers.newline)^-1

```

```

11190         * parsers.check_minimal_indent
11191         * parsers.check_trail
11192         * parsers.heading_level)
11193     * Cs(heading_text) / strip_trailing_spaces
11194     / parsers.parse_heading_text
11195     * Cg(Ct((parsers.attributes
11196         * parsers.optionalspace
11197         * parsers.newline)^-1), "attributes")
11198     * parsers.check_minimal_indent_and_trail * parsers.heading_level
11199     * Cb("attributes")
11200     * parsers.newline
11201     * parsers.unfreeze_trail
11202     / writer.heading
11203
11204     local Heading = AtxHeading + SetextHeading
11205     self.update_rule("Heading", Heading)
11206 end
11207 }
11208 end

```

### 3.1.7.9 Inline Code Attributes

The `extensions.inline_code_attributes` function implements the Pandoc inline code attribute syntax extension.

```

11209 M.extensions.inline_code_attributes = function()
11210   return {
11211     name = "built-in inline_code_attributes syntax extension",
11212     extend_writer = function()
11213     end, extend_reader = function(self)
11214       local writer = self.writer
11215
11216       local CodeWithAttributes = parsers.inticks
11217         * Ct(parsers.attributes)
11218         / writer.code
11219
11220       self.insert_pattern("Inline before Code",
11221         CodeWithAttributes,
11222         "CodeWithAttributes")
11223     end
11224   }
11225 end

```

### 3.1.7.10 Line Blocks

The `extensions.line_blocks` function implements the Pandoc line block syntax extension.

```

11226 M.extensions.line_blocks = function()

```

```

11227 return {
11228     name = "built-in line_blocks syntax extension",
11229     extend_writer = function(self)

```

Define `writer->lineblock` as a function that will transform a line block consisted of `lines` to the output format, with all but the last newline rendered as a line break.

```

11230     function self.lineblock(lines)
11231         if not self.is_writing then return "" end
11232         local buffer = {}
11233         for i = 1, #lines - 1 do
11234             buffer[#buffer + 1] = { lines[i], self.hard_line_break }
11235         end
11236         buffer[#buffer + 1] = lines[#lines]
11237
11238         return {"\\markdownRendererLineBlockBegin\n"
11239             ,buffer,
11240             "\n\\markdownRendererLineBlockEnd "}
11241     end
11242 end, extend_reader = function(self)
11243     local parsers = self.parsers
11244     local writer = self.writer
11245
11246     local LineBlock
11247     = Ct((Cs(( parsers.pipe * parsers.space) / ""
11248         * ((parsers.space)/entities.char_entity("nbsp"))^0
11249         * parsers.linechar^0 * (parsers.newline/""))
11250         * (-parsers.pipe
11251         * (parsers.space^1/" ")
11252         * parsers.linechar^1
11253         * (parsers.newline/"")
11254         )^0
11255         * (parsers.blankline/"")^0)
11256         / self.parser_functions.parse_inlines)^1)
11257     / writer.lineblock
11258
11259     self.insert_pattern("Block after Blockquote",
11260         LineBlock, "LineBlock")
11261 end
11262 }
11263 end

```

### 3.1.7.11 Marked text

The `extensions.mark` function implements the Pandoc mark syntax extension.

```

11264 M.extensions.mark = function()
11265     return {
11266         name = "built-in mark syntax extension",
11267         extend_writer = function(self)

```



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

```
11268     function self.mark(s)
11269         if self.flatten_inlines then return s end
11270         return {"\\markdownRendererMark{" , s, "}"}
11271     end
11272 end, extend_reader = function(self)
11273     local parsers = self.parsers
11274     local writer = self.writer
11275
11276     local doubleequals = P("==")
11277
11278     local Mark
11279         = parsers.between(V("Inline"), doubleequals, doubleequals)
11280         / function (inlines) return writer.mark(inlines) end
11281
11282     self.add_special_character("=")
11283     self.insert_pattern("Inline before LinkAndEmph",
11284                         Mark, "Mark")
11285 end
11286 }
11287 end
```

### 3.1.7.12 Link Attributes

The `extensions.link_attributes` function implements the Pandoc link attribute syntax extension.

```
11288 M.extensions.link_attributes = function()
11289     return {
11290         name = "built-in link_attributes syntax extension",
11291         extend_writer = function()
11292         end, extend_reader = function(self)
11293             local parsers = self.parsers
11294             local options = self.options
11295
```

The following patterns define link reference definitions with attributes.

```
11296     local define_reference_parser
11297         = (parsers.check_trail / "")
11298         * parsers.link_label
11299         * parsers.colon
11300         * parsers.spnlc * parsers.url
11301         * ( parsers.spnlc_sep * parsers.title
11302           * (parsers.spnlc * Ct(parsers.attributes))
11303           * parsers.only_blank
11304           + parsers.spnlc_sep * parsers.title * parsers.only_blank
11305           + Cc("") * (parsers.spnlc * Ct(parsers.attributes))
```

```

11306         * parsers.only_blank
11307         + Cc("") * parsers.only_blank)
11308
11309     local ReferenceWithAttributes = define_reference_parser
11310         / self.register_link
11311
11312     self.update_rule("Reference", ReferenceWithAttributes)
11313

```

The following patterns define direct and indirect links with attributes.

```

11314
11315     local LinkWithAttributesAndEmph
11316         = Ct(parsers.link_and_emph_table * Cg(Cc(true),
11317             "match_link_attributes"))
11318         / self.defer_link_and_emphasis_processing
11319
11320     self.update_rule("LinkAndEmph", LinkWithAttributesAndEmph)
11321

```

The following patterns define autolinks with attributes.

```

11322     local AutoLinkUrlWithAttributes
11323         = parsers.auto_link_url
11324         * Ct(parsers.attributes)
11325         / self.auto_link_url
11326
11327     self.insert_pattern("Inline before AutoLinkUrl",
11328         AutoLinkUrlWithAttributes,
11329         "AutoLinkUrlWithAttributes")
11330
11331     local AutoLinkEmailWithAttributes
11332         = parsers.auto_link_email
11333         * Ct(parsers.attributes)
11334         / self.auto_link_email
11335
11336     self.insert_pattern("Inline before AutoLinkEmail",
11337         AutoLinkEmailWithAttributes,
11338         "AutoLinkEmailWithAttributes")
11339
11340     if options.relativeReferences then
11341
11342         local AutoLinkRelativeReferenceWithAttributes
11343             = parsers.auto_link_relative_reference
11344             * Ct(parsers.attributes)
11345             / self.auto_link_url
11346
11347         self.insert_pattern(
11348             "Inline before AutoLinkRelativeReference",
11349             AutoLinkRelativeReferenceWithAttributes,

```

```

11350         "AutoLinkRelativeReferenceWithAttributes")
11351
11352     end
11353
11354 end
11355 }
11356 end

```

### 3.1.7.13 Notes

The `extensions.notes` function implements the Pandoc note and inline note syntax extensions. When the `note` parameter is `true`, the Pandoc note syntax extension will be enabled. When the `inline_notes` parameter is `true`, the Pandoc inline note syntax extension will be enabled.

```

11357 M.extensions.notes = function(notes, inline_notes)
11358   assert(notes or inline_notes)
11359   return {
11360     name = "built-in notes syntax extension",
11361     extend_writer = function(self)

```

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

```

11362     function self.note(s)
11363       if self.flatten_inlines then return "" end
11364       return {"\\markdownRendererNote{",s,""}
11365     end
11366   end, extend_reader = function(self)
11367     local parsers = self.parsers
11368     local writer = self.writer
11369
11370     local rawnotes = parsers.rawnotes
11371
11372     if inline_notes then
11373       local InlineNote
11374       = parsers.circumflex
11375       * ( parsers.link_label
11376         / self.parser_functions.parse_inlines_no_inline_note)
11377       / writer.note
11378
11379       self.insert_pattern("Inline after LinkAndEmph",
11380                           InlineNote, "InlineNote")
11381     end
11382     if notes then
11383       local function strip_first_char(s)
11384         return s:sub(2)
11385       end
11386

```

```

11387     local RawNoteRef
11388         = #(parsers.lbracket * parsers.circumflex)
11389         * parsers.link_label / strip_first_char
11390
11391     -- like indirect_link
11392     local function lookup_note(ref)
11393         return writer.defer_call(function()
11394             local found = rawnotes[self.normalize_tag(ref)]
11395             if found then
11396                 return writer.note(
11397                     self.parser_functions.parse_blocks_nested(found))
11398             else
11399                 return {"[",
11400                     self.parser_functions.parse_inlines("^" .. ref), ""]}
11401             end
11402         end)
11403     end
11404
11405     local function register_note(ref,rawnote)
11406         local normalized_tag = self.normalize_tag(ref)
11407         if rawnotes[normalized_tag] == nil then
11408             rawnotes[normalized_tag] = rawnote
11409         end
11410         return ""
11411     end
11412
11413     local NoteRef = RawNoteRef / lookup_note
11414
11415     local optionally_indented_line
11416         = parsers.check_optional_indent_and_any_trail * parsers.line
11417
11418     local blank
11419         = parsers.check_optional_blank_indent_and_any_trail
11420         * parsers.optionalspace * parsers.newline
11421
11422     local chunk
11423         = Cs(parsers.line
11424             * (optionally_indented_line - blank)^0)
11425
11426     local indented_blocks = function(bl)
11427         return Cs( bl
11428             * ( blank^1 * (parsers.check_optional_indent / "")
11429             * parsers.check_code_trail
11430             * -parsers.blankline * bl)^0)
11431     end
11432
11433     local NoteBlock

```

```

11434         = parsers.check_trail_no_rem
11435         * RawNoteRef * parsers.colon
11436         * parsers.spnlc * indented_blocks(chunk)
11437         / register_note
11438
11439         local Reference = NoteBlock + parsers.Reference
11440
11441         self.update_rule("Reference", Reference)
11442         self.insert_pattern("Inline before LinkAndEmph",
11443                             NoteRef, "NoteRef")
11444     end
11445
11446     self.add_special_character("^")
11447 end
11448 }
11449 end

```

### 3.1.7.14 Pipe Tables

The `extensions.pipe_table` function implements the PHP Markdown table syntax extension (also known as pipe tables in Pandoc). When the `table_captions` parameter is `true`, the function also implements the Pandoc table caption syntax extension for table captions. When the `table_attributes` parameter is also `true`, the function also allows attributes to be attached to the (possibly empty) table captions.

```

11450 M.extensions.pipe_tables = function(table_captions, table_attributes)
11451
11452     local function make_pipe_table_rectangular(rows)
11453         local num_columns = #rows[2]
11454         local rectangular_rows = {}
11455         for i = 1, #rows do
11456             local row = rows[i]
11457             local rectangular_row = {}
11458             for j = 1, num_columns do
11459                 rectangular_row[j] = row[j] or ""
11460             end
11461             table.insert(rectangular_rows, rectangular_row)
11462         end
11463         return rectangular_rows
11464     end
11465
11466     local function pipe_table_row(allow_empty_first_column
11467                                   , nonempty_column
11468                                   , column_separator
11469                                   , column)
11470         local row_beginning
11471         if allow_empty_first_column then

```

```

11472     row_beginning = -- empty first column
11473                 #(parsers.spacechar^4
11474                 * column_separator)
11475                 * parsers.optionalspace
11476                 * column
11477                 * parsers.optionalspace
11478     -- non-empty first column
11479     + parsers.nonindentspace
11480     * nonempty_column^-1
11481     * parsers.optionalspace
11482 else
11483     row_beginning = parsers.nonindentspace
11484                 * nonempty_column^-1
11485                 * parsers.optionalspace
11486 end
11487
11488 return Ct(row_beginning
11489         * (-- single column with no leading pipes
11490         #(column_separator
11491         * parsers.optionalspace
11492         * parsers.newline)
11493         * column_separator
11494         * parsers.optionalspace
11495         -- single column with leading pipes or
11496         -- more than a single column
11497         + (column_separator
11498         * parsers.optionalspace
11499         * column
11500         * parsers.optionalspace)^1
11501         * (column_separator
11502         * parsers.optionalspace)^-1))
11503 end
11504
11505 return {
11506     name = "built-in pipe_tables syntax extension",
11507     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.

```

11508     function self.table(rows, caption, attributes)
11509         if not self.is_writing then return "" end
11510         local buffer = {}
11511         if attributes ~= nil then
11512             table.insert(buffer,
11513                 "\\markdownRendererTableAttributeContextBegin\n")
11514             table.insert(buffer, self.attributes(attributes))

```

```

11515     end
11516     table.insert(buffer,
11517         {"\\markdownRendererTable{" ,
11518         caption or "", "}" , #rows - 1, "}" ,
11519         #rows[1], "}")
11520     local temp = rows[2] -- put alignments on the first row
11521     rows[2] = rows[1]
11522     rows[1] = temp
11523     for i, row in ipairs(rows) do
11524         table.insert(buffer, "{")
11525         for _, column in ipairs(row) do
11526             if i > 1 then -- do not use braces for alignments
11527                 table.insert(buffer, "{")
11528             end
11529             table.insert(buffer, column)
11530             if i > 1 then
11531                 table.insert(buffer, "}")
11532             end
11533         end
11534         table.insert(buffer, "}")
11535     end
11536     if attributes ~= nil then
11537         table.insert(buffer,
11538             "\\markdownRendererTableAttributeContextEnd{}")
11539     end
11540     return buffer
11541 end
11542 end, extend_reader = function(self)
11543     local parsers = self.parsers
11544     local writer = self.writer
11545
11546     local table_hline_separator = parsers.pipe + parsers.plus
11547
11548     local table_hline_column = (parsers.dash
11549         - #(parsers.dash
11550             * (parsers.spacechar
11551                 + table_hline_separator
11552                 + parsers.newline)))^1
11553         * (parsers.colon * Cc("r")
11554             + parsers.dash * Cc("d"))
11555         + parsers.colon
11556         * (parsers.dash
11557             - #(parsers.dash
11558                 * (parsers.spacechar
11559                     + table_hline_separator
11560                     + parsers.newline)))^1
11561         * (parsers.colon * Cc("c")

```

```

11562             + parsers.dash * Cc("l"))
11563
11564     local table_hline = pipe_table_row(false
11565                                     , table_hline_column
11566                                     , table_hline_separator
11567                                     , table_hline_column)
11568
11569     local table_caption_beginning
11570     = ( parsers.check_minimal_blank_indent_and_any_trail_no_rem
11571       * parsers.optionalspace * parsers.newline)^0
11572     * parsers.check_minimal_indent_and_trail
11573     * (P("Table")^-1 * parsers.colon)
11574     * parsers.optionalspace
11575
11576     local function strip_trailing_spaces(s)
11577     return s:gsub("%s*$","")
11578     end
11579
11580     local table_row
11581     = pipe_table_row(true
11582                     , (C((parsers.linechar - parsers.pipe)^1)
11583                       / strip_trailing_spaces
11584                       / self.parser_functions.parse_inlines)
11585                     , parsers.pipe
11586                     , (C((parsers.linechar - parsers.pipe)^0)
11587                       / strip_trailing_spaces
11588                       / self.parser_functions.parse_inlines))
11589
11590     local table_caption
11591     if table_captions then
11592       table_caption = #table_caption_beginning
11593                     * table_caption_beginning
11594       if table_attributes then
11595         table_caption = table_caption
11596                       * (C(((( parsers.linechar
11597                             - (parsers.attributes
11598                               * parsers.optionalspace
11599                               * parsers.newline
11600                               * -( parsers.optionalspace
11601                                 * parsers.linechar))))
11602                             + ( parsers.newline
11603                               * #( parsers.optionalspace
11604                                 * parsers.linechar)
11605                               * C(parsers.optionalspace)
11606                               / writer.space))
11607                               * (parsers.linechar
11608                                 - parsers.lbrace)^0)^1)

```



```

11609         / self.parser_functions.parse_inlines)
11610     * (parsers.newline
11611     + ( Ct(parsers.attributes)
11612     * parsers.optionalspace
11613     * parsers.newline))
11614     else
11615         table_caption = table_caption
11616         * C(( parsers.linechar
11617         + ( parsers.newline
11618         * #( parsers.optionalspace
11619         * parsers.linechar)
11620         * C(parsers.optionalspace)
11621         / writer.space))^1)
11622         / self.parser_functions.parse_inlines
11623         * parsers.newline
11624     end
11625     else
11626         table_caption = parsers.fail
11627     end
11628
11629     local PipeTable
11630     = Ct( table_row * parsers.newline
11631     * (parsers.check_minimal_indent_and_trail / {})
11632     * table_hline * parsers.newline
11633     * ( (parsers.check_minimal_indent / {})
11634     * table_row * parsers.newline)^0)
11635     / make_pipe_table_rectangular
11636     * table_caption^-1
11637     / writer.table
11638
11639     self.insert_pattern("Block after Blockquote",
11640         PipeTable, "PipeTable")
11641     end
11642 }
11643 end

```

### 3.1.7.15 Raw Attributes

The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```

11644 M.extensions.raw_inline = function()
11645     return {
11646         name = "built-in raw_inline syntax extension",
11647         extend_writer = function(self)
11648             local options = self.options
11649

```

Define `writer->rawInline` as a function that will transform an input inline raw span `s` with the raw attribute `attr` to the output format.

```
11650     function self.rawInline(s, attr)
11651         if not self.is_writing then return "" end
11652         if self.flatten_inlines then return s end
11653         local name = util.cache_verbatim(options.cacheDir, s)
11654         return {"\\markdownRendererInputRawInline{" ,
11655             name,"}{" , self.string(attr),"}" }
11656     end
11657 end, extend_reader = function(self)
11658     local writer = self.writer
11659
11660     local RawInline = parsers.inticks
11661         * parsers.raw_attribute
11662         / writer.rawInline
11663
11664     self.insert_pattern("Inline before Code",
11665         RawInline, "RawInline")
11666 end
11667 }
11668 end
```

### 3.1.7.16 Strike-Through

The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```
11669 M.extensions.strike_through = function()
11670     return {
11671         name = "built-in strike_through syntax extension",
11672         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.

```
11673     function self.strike_through(s)
11674         if self.flatten_inlines then return s end
11675         return {"\\markdownRendererStrikeThrough{" ,s,"}" }
11676     end
11677 end, extend_reader = function(self)
11678     local parsers = self.parsers
11679     local writer = self.writer
11680
11681     local StrikeThrough = (
11682         parsers.between(parsers.Inline, parsers.doubletildes,
11683             parsers.doubletildes)
11684     ) / writer.strike_through
11685
11686     self.insert_pattern("Inline after LinkAndEmph",
```

```

11687             StrikeThrough, "StrikeThrough")
11688
11689         self.add_special_character("~")
11690     end
11691 }
11692 end

```

### 3.1.7.17 Subscripts

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

```

11693 M.extensions.subscripts = function()
11694   return {
11695     name = "built-in subscripts syntax extension",
11696     extend_writer = function(self)

```

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

```

11697         function self.subscript(s)
11698           if self.flatten_inlines then return s end
11699           return {"\\markdownRendererSubscript{" ,s,"}"}
11700         end
11701     end, extend_reader = function(self)
11702       local parsers = self.parsers
11703       local writer = self.writer
11704
11705       local Subscript = (
11706         parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
11707       ) / writer.subscript
11708
11709       self.insert_pattern("Inline after LinkAndEmph",
11710         Subscript, "Subscript")
11711
11712       self.add_special_character("~")
11713     end
11714   }
11715 end

```

### 3.1.7.18 Superscripts

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

```

11716 M.extensions.superscripts = function()
11717   return {
11718     name = "built-in superscripts syntax extension",
11719     extend_writer = function(self)

```

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

```
11720     function self.superscript(s)
11721         if self.flatten_inlines then return s end
11722         return {"\\markdownRendererSuperscript{" ,s,"}"}
11723     end
11724 end, extend_reader = function(self)
11725     local parsers = self.parsers
11726     local writer = self.writer
11727
11728     local Superscript = (
11729         parsers.between(parsers.Str, parsers.circumflex,
11730             parsers.circumflex)
11731     ) / writer.superscript
11732
11733     self.insert_pattern("Inline after LinkAndEmph",
11734         Superscript, "Superscript")
11735
11736     self.add_special_character("^")
11737 end
11738 }
11739 end
```

### 3.1.7.19 T<sub>E</sub>X Math

The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```
11740 M.extensions.tex_math = function(tex_math_dollars,
11741     tex_math_single_backslash,
11742     tex_math_double_backslash)
11743     return {
11744         name = "built-in tex_math syntax extension",
11745         extend_writer = function(self)
```

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

```
11746     function self.display_math(s)
11747         if self.flatten_inlines then return s end
11748         return {"\\markdownRendererDisplayMath{" ,self.math(s),"}"}
11749     end
```

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

```
11750     function self.inline_math(s)
11751         if self.flatten_inlines then return s end
11752         return {"\\markdownRendererInlineMath{" ,self.math(s),"}"}
11753     end
```

```

11754     end, extend_reader = function(self)
11755         local parsers = self.parsers
11756         local writer = self.writer
11757
11758         local function between(p, starter, ender)
11759             return (starter * Cs(p * (p - ender)^0) * ender)
11760         end
11761
11762         local function strip_preceding_whitespaces(str)
11763             return str:gsub("^%s*(.)$", "%1")
11764         end
11765
11766         local allowed_before_closing
11767             = B( parsers.backslash * parsers.any
11768                 + parsers.any * (parsers.any - parsers.backslash))
11769
11770         local allowed_before_closing_no_space
11771             = B( parsers.backslash * parsers.any
11772                 + parsers.any * (parsers.nonspacechar - parsers.backslash))
11773

```

The following patterns implement the Pandoc dollar math syntax extension.

```

11774     local dollar_math_content
11775         = (parsers.newline * (parsers.check_optional_indent / "")
11776           + parsers.backslash^-1
11777           * parsers.linechar)
11778         - parsers.blankline^2
11779         - parsers.dollar
11780
11781     local inline_math_opening_dollars = parsers.dollar
11782                                         * #(parsers.nonspacechar)
11783
11784     local inline_math_closing_dollars
11785         = allowed_before_closing_no_space
11786         * parsers.dollar
11787         * -#(parsers.digit)
11788
11789     local inline_math_dollars = between(Cs( dollar_math_content),
11790                                         inline_math_opening_dollars,
11791                                         inline_math_closing_dollars)
11792
11793     local display_math_opening_dollars = parsers.dollar
11794                                         * parsers.dollar
11795
11796     local display_math_closing_dollars = parsers.dollar
11797                                         * parsers.dollar
11798
11799     local display_math_dollars = between(Cs( dollar_math_content),

```

```

11800             display_math_opening_dollars,
11801             display_math_closing_dollars)

```

The following patterns implement the Pandoc single and double backslash math syntax extensions.

```

11802     local backslash_math_content
11803     = (parsers.newline * (parsers.check_optional_indent / ""))
11804     + parsers.linechar)
11805     - parsers.blankline^2

```

The following patterns implement the Pandoc double backslash math syntax extension.

```

11806     local inline_math_opening_double = parsers.backslash
11807     * parsers.backslash
11808     * parsers.lparent
11809
11810     local inline_math_closing_double = allowed_before_closing
11811     * parsers.spacechar^0
11812     * parsers.backslash
11813     * parsers.backslash
11814     * parsers.rparent
11815
11816     local inline_math_double = between(Cs( backslash_math_content),
11817     inline_math_opening_double,
11818     inline_math_closing_double)
11819     / strip_preceding_whitespaces
11820
11821     local display_math_opening_double = parsers.backslash
11822     * parsers.backslash
11823     * parsers.lbracket
11824
11825     local display_math_closing_double = allowed_before_closing
11826     * parsers.spacechar^0
11827     * parsers.backslash
11828     * parsers.backslash
11829     * parsers.rbracket
11830
11831     local display_math_double = between(Cs( backslash_math_content),
11832     display_math_opening_double,
11833     display_math_closing_double)
11834     / strip_preceding_whitespaces

```

The following patterns implement the Pandoc single backslash math syntax extension.

```

11835     local inline_math_opening_single = parsers.backslash
11836     * parsers.lparent
11837
11838     local inline_math_closing_single = allowed_before_closing
11839     * parsers.spacechar^0

```

```

11840             * parsers.backslash
11841             * parsers.rparent
11842
11843     local inline_math_single = between(Cs( backslash_math_content),
11844                                     inline_math_opening_single,
11845                                     inline_math_closing_single)
11846                                     / strip_preceding_whitespaces
11847
11848     local display_math_opening_single = parsers.backslash
11849                                     * parsers.lbracket
11850
11851     local display_math_closing_single = allowed_before_closing
11852                                     * parsers.spacechar^0
11853                                     * parsers.backslash
11854                                     * parsers.rbracket
11855
11856     local display_math_single = between(Cs( backslash_math_content),
11857                                     display_math_opening_single,
11858                                     display_math_closing_single)
11859                                     / strip_preceding_whitespaces
11860
11861     local display_math = parsers.fail
11862
11863     local inline_math = parsers.fail
11864
11865     if tex_math_dollars then
11866         display_math = display_math + display_math_dollars
11867         inline_math = inline_math + inline_math_dollars
11868     end
11869
11870     if tex_math_double_backslash then
11871         display_math = display_math + display_math_double
11872         inline_math = inline_math + inline_math_double
11873     end
11874
11875     if tex_math_single_backslash then
11876         display_math = display_math + display_math_single
11877         inline_math = inline_math + inline_math_single
11878     end
11879
11880     local TexMath = display_math / writer.display_math
11881                   + inline_math / writer.inline_math
11882
11883     self.insert_pattern("Inline after LinkAndEmph",
11884                       TexMath, "TexMath")
11885
11886     if tex_math_dollars then

```

```

11887         self.add_special_character("$")
11888     end
11889
11890     if tex_math_single_backslash or tex_math_double_backslash then
11891         self.add_special_character("\\")
11892         self.add_special_character("[")
11893         self.add_special_character("]")
11894         self.add_special_character("(")
11895         self.add_special_character("(")
11896     end
11897 end
11898 }
11899 end

```

### 3.1.7.20 YAML Metadata

The `extensions.jekyll_data` function implements the Pandoc YAML metadata block syntax extension. 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. When both `expect_jekyll_data` and `ensure_jekyll_data` parameters are `true`, then a a markdown document must begin directly with YAML metadata and must contain nothing but YAML metadata.

```

11900 M.extensions.jekyll_data = function(expect_jekyll_data,
11901                                     ensure_jekyll_data)
11902   return {
11903     name = "built-in jekyll_data syntax extension",
11904     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` for the typographic output format used by the `\markdownRendererJekyllDataTypographicString` macro.

```

11905     function self.jekyllData(d, t, p)
11906       if not self.is_writing then return "" end
11907
11908       local buf = {}
11909
11910       local keys = {}
11911       for k, _ in pairs(d) do
11912         table.insert(keys, k)
11913       end

```

For reproducibility, sort the keys. For mixed string-and-numeric keys, sort numeric keys before string keys.

```

11914       table.sort(keys, function(first, second)

```



```

11915         if type(first) ~= type(second) then
11916             return type(first) < type(second)
11917         else
11918             return first < second
11919         end
11920     end)
11921
11922     if not p then
11923         table.insert(buf, "\\markdownRendererJekyllDataBegin")
11924     end
11925
11926     local is_sequence = false
11927     if #d > 0 and #d == #keys then
11928         for i=1, #d do
11929             if d[i] == nil then
11930                 goto not_a_sequence
11931             end
11932         end
11933         is_sequence = true
11934     end
11935     ::not_a_sequence::
11936
11937     if is_sequence then
11938         table.insert(buf,
11939             "\\markdownRendererJekyllDataSequenceBegin{")
11940         table.insert(buf, self.identifier(p or "null"))
11941         table.insert(buf, "}{"")
11942         table.insert(buf, #keys)
11943         table.insert(buf, "}")
11944     else
11945         table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
11946         table.insert(buf, self.identifier(p or "null"))
11947         table.insert(buf, "}{"")
11948         table.insert(buf, #keys)
11949         table.insert(buf, "}")
11950     end
11951
11952     for _, k in ipairs(keys) do
11953         local v = d[k]
11954         local typ = type(v)
11955         k = tostring(k or "null")
11956         if typ == "table" and next(v) ~= nil then
11957             table.insert(
11958                 buf,
11959                 self.jekyllData(v, t, k)
11960             )
11961         else

```

```

11962         k = self.identifier(k)
11963         v = tostring(v)
11964         if typ == "boolean" then
11965             table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
11966             table.insert(buf, k)
11967             table.insert(buf, "}{")
11968             table.insert(buf, v)
11969             table.insert(buf, "}")
11970         elseif typ == "number" then
11971             table.insert(buf, "\\markdownRendererJekyllDataNumber{")
11972             table.insert(buf, k)
11973             table.insert(buf, "}{")
11974             table.insert(buf, v)
11975             table.insert(buf, "}")
11976         elseif typ == "string" then
11977             table.insert(buf,
11978                 "\\markdownRendererJekyllDataProgrammaticString{")
11979             table.insert(buf, k)
11980             table.insert(buf, "}{")
11981             table.insert(buf, self.identifier(v))
11982             table.insert(buf, "}")
11983             table.insert(buf,
11984                 "\\markdownRendererJekyllDataTypographicString{")
11985             table.insert(buf, k)
11986             table.insert(buf, "}{")
11987             table.insert(buf, t(v))
11988             table.insert(buf, "}")
11989         elseif typ == "table" then
11990             table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
11991             table.insert(buf, k)
11992             table.insert(buf, "}")
11993         else
11994             local error = self.error(format(
11995                 "Unexpected type %s for value of "
11996                 .. "YAML key %s", typ, k))
11997             table.insert(buf, error)
11998         end
11999     end
12000 end
12001
12002 if is_sequence then
12003     table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
12004 else
12005     table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
12006 end
12007
12008 if not p then

```

```

12009         table.insert(buf, "\\markdownRendererJekyllDataEnd")
12010     end
12011
12012     return buf
12013 end
12014 end, extend_reader = function(self)
12015     local parsers = self.parsers
12016     local writer = self.writer
12017
12018     local JekyllData
12019     = Cmt( C((parsers.line - P("----") - P("..."))^0)
12020         , function(s, i, text) -- luacheck: ignore s i
12021             local data
12022             local ran_ok, _ = pcall(function()
12023                 -- TODO: Use `require("tinyyaml")` in TeX Live 2023
12024                 local tinyyaml = require("markdown-tinyyaml")
12025                 data = tinyyaml.parse(text, {timestamps=false})
12026             end)
12027             if ran_ok and data ~= nil then
12028                 return true, writer.jekyllData(data, function(s)
12029                     return self.parser_functions.parse_blocks_nested(s)
12030                 end, nil)
12031             else
12032                 return false
12033             end
12034         end
12035     )
12036
12037     local UnexpectedJekyllData
12038     = P("----")
12039     * parsers.blankline / 0
12040     -- if followed by blank, it's thematic break
12041     * #(-parsers.blankline)
12042     * JekyllData
12043     * (P("----") + P("..."))
12044
12045     local ExpectedJekyllData
12046     = ( P("----")
12047         * parsers.blankline / 0
12048         -- if followed by blank, it's thematic break
12049         * #(-parsers.blankline)
12050         )^-1
12051     * JekyllData
12052     * (P("----") + P("..."))^-1
12053
12054     if ensure_jekyll_data then
12055         ExpectedJekyllData = ExpectedJekyllData

```

```

12056             * parsers.eof
12057     else
12058         ExpectedJekyllData = ( ExpectedJekyllData
12059             * (V("Blank")^0 / writer.interblocksep)
12060             )^-1
12061     end
12062
12063     self.insert_pattern("Block before Blockquote",
12064         UnexpectedJekyllData, "UnexpectedJekyllData")
12065     if expect_jekyll_data then
12066         self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
12067     end
12068 end
12069 }
12070 end

```

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

The `new` function of file `markdown.lua` loads file `markdown-parser.lua` and calls its own function `new` unless option `eagerCache` or `finalizeCache` has been enabled and a cached conversion output exists, in which case it is returned without loading file `markdown-parser.lua`.

```

12071 local function warn(s)
12072     io.stderr:write("Warning: " .. s .. "\n")
12073 end
12074
12075 function M.new(options)

```

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

```

12076     options = options or {}
12077     setmetatable(options, { __index = function (_, key)
12078         return defaultOptions[key] end })

```

Return a conversion function that tries to produce a cached conversion output exists. If no cached conversion output exists, we load the file `markdown-parser.lua` and use it to convert the input.

```

12079     local parser_convert = nil
12080     return function(input)
12081         local function convert(input)
12082             if parser_convert == nil then

```

Lazy-load `markdown-parser.lua` and check that it originates from the same version of the Markdown package.

```

12083         local parser = require("markdown-parser")
12084         if metadata.version ~= parser.metadata.version then
12085             warn("markdown.lua " .. metadata.version .. " used with " ..
12086                 "markdown-parser.lua " .. parser.metadata.version .. ".")

```

```

12087         end
12088         parser_convert = parser.new(options)
12089     end
12090     return parser_convert(input)
12091 end

```

If we cache markdown documents, produce the cache file and transform its filename to plain TeX output.

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 2.1.3).

```

12092     local output
12093     if options.eagerCache or options.finalizeCache then
12094         local salt = util.salt(options)
12095         local name = util.cache(options.cacheDir, input, salt, convert,
12096                               ".md.tex")
12097         output = [[\input{}} .. name .. [{}\relax]]

```

Otherwise, return the result of the conversion directly.

```

12098     else
12099         output = convert(input)
12100     end

```

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

```

12101     if options.finalizeCache then
12102         local file, mode
12103         if options.frozenCacheCounter > 0 then
12104             mode = "a"
12105         else
12106             mode = "w"
12107         end
12108         file = assert(io.open(options.frozenCacheFileName, mode),
12109                      [[Could not open file "]] .. options.frozenCacheFileName
12110                      .. ["] for writing]])
12111         assert(file:write(
12112             [[\expandafter\global\expandafter\def\csname ]]
12113             .. [[markdownFrozenCache]] .. options.frozenCacheCounter
12114             .. [[\endcsname{}} .. output .. [{}]] .. "\n"))
12115         assert(file:close())
12116     end
12117     return output
12118 end
12119 end

```

The `new` function from file `markdown-parser.lua` returns a conversion function that takes a markdown string and turns it into a plain TeX output. See Section 2.1.1.

```
12120 function M.new(options)
```

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

```
12121 options = options or {}
12122 setmetatable(options, { __index = function (_, key)
12123     return defaultOptions[key] end })
```

If the singleton cache contains a conversion function for the same `options`, reuse it.

```
12124 if options.singletonCache and singletonCache.convert then
12125     for k, v in pairs(defaultOptions) do
12126         if type(v) == "table" then
12127             for i = 1, math.max(#singletonCache.options[k], #options[k]) do
12128                 if singletonCache.options[k][i] ~= options[k][i] then
12129                     goto miss
12130                 end
12131             end
12132         end
12133     end
```

The `cacheDir` option is disregarded.

```
12132     elseif k ~= "cacheDir"
12133         and singletonCache.options[k] ~= options[k] then
12134         goto miss
12135     end
12136 end
12137 return singletonCache.convert
12138 end
12139 ::miss::
```

Apply built-in syntax extensions based on `options`.

```
12140 local extensions = {}
12141
12142 if options.bracketedSpans then
12143     local bracketed_spans_extension = M.extensions.bracketed_spans()
12144     table.insert(extensions, bracketed_spans_extension)
12145 end
12146
12147 if options.contentBlocks then
12148     local content_blocks_extension = M.extensions.content_blocks(
12149         options.contentBlocksLanguageMap)
12150     table.insert(extensions, content_blocks_extension)
12151 end
12152
12153 if options.definitionLists then
12154     local definition_lists_extension = M.extensions.definition_lists(
12155         options.tightLists)
```

```

12156     table.insert(extensions, definition_lists_extension)
12157 end
12158
12159 if options.fencedCode then
12160     local fenced_code_extension = M.extensions.fenced_code(
12161         options.blankBeforeCodeFence,
12162         options.fencedCodeAttributes,
12163         options.rawAttribute)
12164     table.insert(extensions, fenced_code_extension)
12165 end
12166
12167 if options.fencedDivs then
12168     local fenced_div_extension = M.extensions.fenced_divs(
12169         options.blankBeforeDivFence)
12170     table.insert(extensions, fenced_div_extension)
12171 end
12172
12173 if options.headerAttributes then
12174     local header_attributes_extension = M.extensions.header_attributes()
12175     table.insert(extensions, header_attributes_extension)
12176 end
12177
12178 if options.inlineCodeAttributes then
12179     local inline_code_attributes_extension =
12180         M.extensions.inline_code_attributes()
12181     table.insert(extensions, inline_code_attributes_extension)
12182 end
12183
12184 if options.jekyllData then
12185     local jekyll_data_extension = M.extensions.jekyll_data(
12186         options.expectJekyllData, options.ensureJekyllData)
12187     table.insert(extensions, jekyll_data_extension)
12188 end
12189
12190 if options.linkAttributes then
12191     local link_attributes_extension =
12192         M.extensions.link_attributes()
12193     table.insert(extensions, link_attributes_extension)
12194 end
12195
12196 if options.lineBlocks then
12197     local line_block_extension = M.extensions.line_blocks()
12198     table.insert(extensions, line_block_extension)
12199 end
12200
12201 if options.mark then
12202     local mark_extension = M.extensions.mark()

```

```

12203     table.insert(extensions, mark_extension)
12204 end
12205
12206 if options.pipeTables then
12207     local pipe_tables_extension = M.extensions.pipe_tables(
12208         options.tableCaptions, options.tableAttributes)
12209     table.insert(extensions, pipe_tables_extension)
12210 end
12211
12212 if options.rawAttribute then
12213     local raw_inline_extension = M.extensions.raw_inline()
12214     table.insert(extensions, raw_inline_extension)
12215 end
12216
12217 if options.strikeThrough then
12218     local strike_through_extension = M.extensions.strike_through()
12219     table.insert(extensions, strike_through_extension)
12220 end
12221
12222 if options.subscripts then
12223     local subscript_extension = M.extensions.subscripts()
12224     table.insert(extensions, subscript_extension)
12225 end
12226
12227 if options.superscripts then
12228     local superscript_extension = M.extensions.superscripts()
12229     table.insert(extensions, superscript_extension)
12230 end
12231
12232 if options.texMathDollars or
12233     options.texMathSingleBackslash or
12234     options.texMathDoubleBackslash then
12235     local tex_math_extension = M.extensions.tex_math(
12236         options.texMathDollars,
12237         options.texMathSingleBackslash,
12238         options.texMathDoubleBackslash)
12239     table.insert(extensions, tex_math_extension)
12240 end
12241
12242 if options.notes or options.inlineNotes then
12243     local notes_extension = M.extensions.notes(
12244         options.notes, options.inlineNotes)
12245     table.insert(extensions, notes_extension)
12246 end
12247
12248 if options.citations then
12249     local citations_extension

```



```

12250     = M.extensions.citations(options.citationNbsps)
12251     table.insert(extensions, citations_extension)
12252 end
12253
12254 if options.fancyLists then
12255     local fancy_lists_extension = M.extensions.fancy_lists()
12256     table.insert(extensions, fancy_lists_extension)
12257 end

```

Apply user-defined syntax extensions based on `options.extensions`.

```

12258 for _, user_extension_filename in ipairs(options.extensions) do
12259     local user_extension = (function(filename)

```

First, load and compile the contents of the user-defined syntax extension.

```

12260     local pathname = assert(kpse.find_file(filename),
12261         [[Could not locate user-defined syntax extension "]]
12262         .. filename)
12263     local input_file = assert(io.open(pathname, "r"),
12264         [[Could not open user-defined syntax extension "]]
12265         .. pathname .. [{" for reading}]]
12266     local input = assert(input_file:read("*a"))
12267     assert(input_file:close())
12268     local user_extension, err = load([[
12269         local sandbox = {}
12270         setmetatable(sandbox, {__index = _G})
12271         _ENV = sandbox
12272     ]] .. input)()
12273     assert(user_extension,
12274         [[Failed to compile user-defined syntax extension "]]
12275         .. pathname .. [{": }]] .. (err or [{"}]))

```

Then, validate the user-defined syntax extension.

```

12276     assert(user_extension.api_version ~= nil,
12277         [[User-defined syntax extension "]] .. pathname
12278         .. [{" does not specify mandatory field "api_version"}]])
12279     assert(type(user_extension.api_version) == "number",
12280         [[User-defined syntax extension "]] .. pathname
12281         .. [{" specifies field "api_version" of type "]]
12282         .. type(user_extension.api_version)
12283         .. [{" but "number" was expected}]]
12284     assert(user_extension.api_version > 0
12285         and user_extension.api_version
12286         <= metadata.user_extension_api_version,
12287         [[User-defined syntax extension "]] .. pathname
12288         .. [{" uses syntax extension API version "]]
12289         .. user_extension.api_version .. [{" but markdown.lua }]]
12290     .. metadata.version .. [{" uses API version }]]
12291     .. metadata.user_extension_api_version

```

```

12292     .. [[, which is incompatible]])
12293
12294     assert(user_extension.grammar_version ~= nil,
12295           [[User-defined syntax extension "]] .. pathname
12296           .. [[ " does not specify mandatory field "grammar_version"]])
12297     assert(type(user_extension.grammar_version) == "number",
12298           [[User-defined syntax extension "]] .. pathname
12299           .. [[ " specifies field "grammar_version" of type "]]
12300           .. type(user_extension.grammar_version)
12301           .. [[ " but "number" was expected]])
12302     assert(user_extension.grammar_version == metadata.grammar_version,
12303           [[User-defined syntax extension "]] .. pathname
12304           .. [[ " uses grammar version "]]
12305           .. user_extension.grammar_version
12306           .. [[ but markdown.lua ]] .. metadata.version
12307           .. [[ uses grammar version ]] .. metadata.grammar_version
12308           .. [[, which is incompatible]])
12309
12310     assert(user_extension.finalize_grammar ~= nil,
12311           [[User-defined syntax extension "]] .. pathname
12312           .. [[ " does not specify mandatory "finalize_grammar" field]])
12313     assert(type(user_extension.finalize_grammar) == "function",
12314           [[User-defined syntax extension "]] .. pathname
12315           .. [[ " specifies field "finalize_grammar" of type "]]
12316           .. type(user_extension.finalize_grammar)
12317           .. [[ " but "function" was expected]])

```

Finally, cast the user-defined syntax extension to the internal format of user extensions used by the Markdown package (see Section 3.1.7.)

```

12318     local extension = {
12319       name = [[user-defined "]] .. pathname .. [[ " syntax extension]],
12320       extend_reader = user_extension.finalize_grammar,
12321       extend_writer = function() end,
12322     }
12323     return extension
12324   end)(user_extension_filename)
12325   table.insert(extensions, user_extension)
12326 end

```

Produce a conversion function from markdown to plain  $\text{\TeX}$ .

```

12327   local writer = M.writer.new(options)
12328   local reader = M.reader.new(writer, options)
12329   local convert = reader.finalize_grammar(extensions)

```

Force garbage collection to reclaim memory for temporary objects created in `writer.new`, `reader.new`, and `reader->finalize_grammar`.

```

12330   collectgarbage("collect")

```

Update the singleton cache.

```

12331 if options.singletonCache then
12332   local singletonCacheOptions = {}
12333   for k, v in pairs(options) do
12334     singletonCacheOptions[k] = v
12335   end
12336   setmetatable(singletonCacheOptions,
12337     { __index = function (_, key)
12338       return defaultOptions[key] end })
12339   singletonCache.options = singletonCacheOptions
12340   singletonCache.convert = convert
12341 end

```

Return the conversion function from markdown to plain T<sub>E</sub>X.

```

12342 return convert
12343 end
12344 return M

```

### 3.1.9 Command-Line Implementation

The command-line implementation provides the actual conversion routine for the command-line interface described in Section 2.1.7.

```

12345
12346 local input
12347 if input_filename then
12348   local input_file = assert(io.open(input_filename, "r"),
12349     [[Could not open file ]] .. input_filename .. [[ for reading]])
12350   input = assert(input_file:read("*a"))
12351   assert(input_file:close())
12352 else
12353   input = assert(io.read("*a"))
12354 end
12355

```

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

```

12356 local lfs = require("lfs")
12357 if options.cacheDir and not lfs.isdir(options.cacheDir) then
12358   assert(lfs.mkdir(options["cacheDir"]))
12359 end

```

If Kpathsea has not been loaded before or if LuaT<sub>E</sub>X has not yet been initialized, configure Kpathsea on top of loading it.

```

12360 local kpse
12361 (function()
12362   local should_initialize = package.loaded.kpse == nil
12363     or tex.initialize ~= nil
12364   kpse = require("kpse")
12365   if should_initialize then

```

```

12366     kpse.set_program_name("luatex")
12367   end
12368 end)()
12369 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.

```

12370 if metadata.version ~= md.metadata.version then
12371   warn("markdown-cli.lua " .. metadata.version .. " used with " ..
12372     "markdown.lua " .. md.metadata.version .. ".")
12373 end
12374 local convert = md.new(options)
12375 local output = convert(input)
12376
12377 if output_filename then
12378   local output_file = assert(io.open(output_filename, "w"),
12379     [[Could not open file ]] .. output_filename .. [[ for writing]])
12380   assert(output_file:write(output))
12381   assert(output_file:close())
12382 else
12383   assert(io.write(output))
12384 end

```

Remove the `options.cacheDir` directory if it is empty.

```

12385 if options.cacheDir then
12386   lfs.rmdir(options["cacheDir"])
12387 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

```

12388 \ExplSyntaxOn
12389 \cs_if_free:NT
12390   \markdownInfo
12391   {
12392     \cs_new:Npn
12393       \markdownInfo #1
12394       {
12395         \msg_info:nne
12396           { markdown }
12397           { generic-message }

```

```

12398         { #1 }
12399     }
12400 }
12401 \cs_if_free:NT
12402   \markdownWarning
12403   {
12404     \cs_new:Npn
12405       \markdownWarning #1
12406       {
12407         \msg_warning:nne
12408           { markdown }
12409           { generic-message }
12410           { #1 }
12411       }
12412   }
12413 \cs_if_free:NT
12414   \markdownError
12415   {
12416     \cs_new:Npn
12417       \markdownError #1 #2
12418       {
12419         \msg_error:nnee
12420           { markdown }
12421           { generic-message-with-help-text }
12422           { #1 }
12423           { #2 }
12424       }
12425   }
12426 \msg_new:nnn
12427   { markdown }
12428   { generic-message }
12429   { #1 }
12430 \msg_new:nnnn
12431   { markdown }
12432   { generic-message-with-help-text }
12433   { #1 }
12434   { #2 }
12435 \cs_generate_variant:Nn
12436   \msg_info:nnn
12437   { nne }
12438 \cs_generate_variant:Nn
12439   \msg_warning:nnn
12440   { nne }
12441 \cs_generate_variant:Nn
12442   \msg_error:nnnn
12443   { nnee }
12444 \ExplSyntaxOff

```

### 3.2.2 Themes

This section implements the theme-loading mechanism and the built-in themes provided with the Markdown package. Furthermore, this section also implements the built-in plain T<sub>E</sub>X themes provided with the Markdown package.

```
12445 \ExplSyntaxOn
12446 \prop_new:N \g_@@_plain_tex_loaded_themes_linenos_prop
12447 \cs_new:Nn
12448   \@@_plain_tex_load_theme:nn
12449   {
12450     \prop_get:NnNTF
12451       \g_@@_plain_tex_loaded_themes_linenos_prop
12452       { #1 }
12453       \l_tmpa_tl
12454       {
12455         \msg_warning:nnnV
12456           { markdown }
12457           { repeatedly-loaded-plain-tex-theme }
12458           { #1 }
12459           \l_tmpa_tl
12460       }
12461     {
12462       \msg_info:nnn
12463         { markdown }
12464         { loading-plain-tex-theme }
12465         { #1 }
12466       \prop_gput:Nnx
12467         \g_@@_plain_tex_loaded_themes_linenos_prop
12468         { #1 }
12469         { \tex_the:D \tex_inputlineno:D }
12470       \file_input:n
12471         { markdown theme #2 }
12472     }
12473   }
12474 \msg_new:nnn
12475   { markdown }
12476   { loading-plain-tex-theme }
12477   { Loading~plain~TeX~Markdown~theme~#1 }
12478 \msg_new:nnn
12479   { markdown }
12480   { repeatedly-loaded-plain-tex-theme }
12481   {
12482     Plain~TeX~Markdown~theme~#1~was~previously~
12483     loaded~on~line~#2,~not~loading~it~again
12484   }
12485 \cs_generate_variant:Nn
12486   \prop_gput:Nnn
```

```

12487 { Nnx }
12488 \cs_gset_eq:NN
12489 \@@_load_theme:nn
12490 \@@_plain_tex_load_theme:nn
12491 \cs_generate_variant:Nn
12492 \@@_load_theme:nn
12493 { nV }

```

Developers can use the `\markdownLoadPlainTeXTheme` macro to load a corresponding plain TeX theme from within themes for higher-level TeX formats such as L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt.

```

12494 \cs_new:Npn
12495 \markdownLoadPlainTeXTheme
12496 {

```

First, we extract the name of the current theme from the `\g_@@_current_theme_tl` macro.

```

12497 \tl_set:NV
12498 \l_tmpa_tl
12499 \g_@@_current_theme_tl
12500 \tl_reverse:N
12501 \l_tmpa_tl
12502 \tl_set:Ne
12503 \l_tmpb_tl
12504 {
12505 \tl_tail:V
12506 \l_tmpa_tl
12507 }
12508 \tl_reverse:N
12509 \l_tmpb_tl

```

Next, we munge the theme name.

```

12510 \str_set:NV
12511 \l_tmpa_str
12512 \l_tmpb_tl
12513 \str_replace_all:Nnn
12514 \l_tmpa_str
12515 { / }
12516 { _ }

```

Finally, we load the plain TeX theme.

```

12517 \@@_plain_tex_load_theme:VV
12518 \l_tmpb_tl
12519 \l_tmpa_str
12520 }
12521 \cs_generate_variant:Nn
12522 \tl_set:Nn
12523 { Ne }

```

```

12524 \cs_generate_variant:Nn
12525   \@@_plain_tex_load_theme:nn
12526   { VV }
12527 \ExplSyntaxOff

```

The `witiko/tilde` theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```

12528 \markdownSetup {
12529   rendererPrototypes = {
12530     tilde = {~},
12531   },
12532 }

```

The `witiko/markdown/defaults` plain  $\TeX$  theme provides default definitions for token renderer prototypes. See Section 3.2.3 for the actual definitions.

### 3.2.3 Token Renderer Prototypes

The following definitions should be considered placeholder.

```

12533 \def\markdownRendererInterblockSeparatorPrototype{\par}%
12534 \def\markdownRendererParagraphSeparatorPrototype{%
12535   \markdownRendererInterblockSeparator}%
12536 \def\markdownRendererHardLineBreakPrototype{\hfil\break}%
12537 \def\markdownRendererSoftLineBreakPrototype{ }%
12538 \let\markdownRendererEllipsisPrototype\dots
12539 \def\markdownRendererNbspPrototype{~}%
12540 \def\markdownRendererLeftBracePrototype{\char`\{}%
12541 \def\markdownRendererRightBracePrototype{\char`\}%
12542 \def\markdownRendererDollarSignPrototype{\char`\$}%
12543 \def\markdownRendererPercentSignPrototype{\char`\}%
12544 \def\markdownRendererAmpersandPrototype{\&%
12545 \def\markdownRendererUnderscorePrototype{\char`\_%
12546 \def\markdownRendererHashPrototype{\char`\#}%
12547 \def\markdownRendererCircumflexPrototype{\char`\^}%
12548 \def\markdownRendererBackslashPrototype{\char`\}%
12549 \def\markdownRendererTildePrototype{\char`\~}%
12550 \def\markdownRendererPipePrototype{|}%
12551 \def\markdownRendererCodeSpanPrototype#1{\tt#1}%
12552 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
12553 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
12554   \markdownInput{#3}}%
12555 \def\markdownRendererContentBlockOnlineImagePrototype{%
12556   \markdownRendererImage}%
12557 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
12558   \markdownRendererInputFencedCode{#3}{#2}{#2}}%
12559 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
12560 \def\markdownRendererUlBeginPrototype{}%
12561 \def\markdownRendererUlBeginTightPrototype{}%

```



```

12562 \def\markdownRendererUListItemPrototype{}%
12563 \def\markdownRendererUListItemEndPrototype{}%
12564 \def\markdownRendererUListEndPrototype{}%
12565 \def\markdownRendererUListEndTightPrototype{}%
12566 \def\markdownRendererOListBeginPrototype{}%
12567 \def\markdownRendererOListBeginTightPrototype{}%
12568 \def\markdownRendererFancyOListBeginPrototype#1#2{%
12569   \markdownRendererOListBegin}%
12570 \def\markdownRendererFancyOListBeginTightPrototype#1#2{%
12571   \markdownRendererOListBeginTight}%
12572 \def\markdownRendererOListItemPrototype{}%
12573 \def\markdownRendererOListItemWithNumberPrototype#1{}%
12574 \def\markdownRendererOListItemEndPrototype{}%
12575 \def\markdownRendererFancyOListItemPrototype{\markdownRendererOListItem}%
12576 \def\markdownRendererFancyOListItemWithNumberPrototype{%
12577   \markdownRendererOListItemWithNumber}%
12578 \def\markdownRendererFancyOListItemEndPrototype{}%
12579 \def\markdownRendererOListEndPrototype{}%
12580 \def\markdownRendererOListEndTightPrototype{}%
12581 \def\markdownRendererFancyOListEndPrototype{\markdownRendererOListEnd}%
12582 \def\markdownRendererFancyOListEndTightPrototype{%
12583   \markdownRendererOListEndTight}%
12584 \def\markdownRendererDListBeginPrototype{}%
12585 \def\markdownRendererDListBeginTightPrototype{}%
12586 \def\markdownRendererDListItemPrototype#1{#1}%
12587 \def\markdownRendererDListItemEndPrototype{}%
12588 \def\markdownRendererDListDefinitionBeginPrototype{}%
12589 \def\markdownRendererDListDefinitionEndPrototype{\par}%
12590 \def\markdownRendererDListEndPrototype{}%
12591 \def\markdownRendererDListEndTightPrototype{}%
12592 \def\markdownRendererEmphasisPrototype#1{\it#1}%
12593 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
12594 \def\markdownRendererBlockQuoteBeginPrototype{\begingroup\it}%
12595 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
12596 \def\markdownRendererLineBlockBeginPrototype{\begingroup\parindent=0pt}%
12597 \def\markdownRendererLineBlockEndPrototype{\endgroup}%
12598 \def\markdownRendererInputVerbatimPrototype#1{%
12599   \par{\tt\input#1\relax}\par}%
12600 \def\markdownRendererInputFencedCodePrototype#1#2#3{%
12601   \markdownRendererInputVerbatim{#1}}%
12602 \def\markdownRendererHeadingOnePrototype#1{#1}%
12603 \def\markdownRendererHeadingTwoPrototype#1{#1}%
12604 \def\markdownRendererHeadingThreePrototype#1{#1}%
12605 \def\markdownRendererHeadingFourPrototype#1{#1}%
12606 \def\markdownRendererHeadingFivePrototype#1{#1}%
12607 \def\markdownRendererHeadingSixPrototype#1{#1}%
12608 \def\markdownRendererThematicBreakPrototype{}%

```

```

12609 \def\markdownRendererNotePrototype#1{#1}%
12610 \def\markdownRendererCitePrototype#1{}%
12611 \def\markdownRendererTextCitePrototype#1{}%
12612 \def\markdownRendererTickedBoxPrototype{[X]}%
12613 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
12614 \def\markdownRendererUntickedBoxPrototype{[ ]}%
12615 \def\markdownRendererStrikeThroughPrototype#1{#1}%
12616 \def\markdownRendererSuperscriptPrototype#1{#1}%
12617 \def\markdownRendererSubscriptPrototype#1{#1}%
12618 \def\markdownRendererDisplayMathPrototype#1{##1$}$%
12619 \def\markdownRendererInlineMathPrototype#1{##1$}%
12620 \ExplSyntaxOn
12621 \cs_gset:Npn
12622   \markdownRendererHeaderAttributeContextBeginPrototype
12623   {
12624     \group_begin:
12625     \color_group_begin:
12626   }
12627 \cs_gset:Npn
12628   \markdownRendererHeaderAttributeContextEndPrototype
12629   {
12630     \color_group_end:
12631     \group_end:
12632   }
12633 \cs_gset_eq:NN
12634   \markdownRendererBracketedSpanAttributeContextBeginPrototype
12635   \markdownRendererHeaderAttributeContextBeginPrototype
12636 \cs_gset_eq:NN
12637   \markdownRendererBracketedSpanAttributeContextEndPrototype
12638   \markdownRendererHeaderAttributeContextEndPrototype
12639 \cs_gset_eq:NN
12640   \markdownRendererFencedDivAttributeContextBeginPrototype
12641   \markdownRendererHeaderAttributeContextBeginPrototype
12642 \cs_gset_eq:NN
12643   \markdownRendererFencedDivAttributeContextEndPrototype
12644   \markdownRendererHeaderAttributeContextEndPrototype
12645 \cs_gset_eq:NN
12646   \markdownRendererFencedCodeAttributeContextBeginPrototype
12647   \markdownRendererHeaderAttributeContextBeginPrototype
12648 \cs_gset_eq:NN
12649   \markdownRendererFencedCodeAttributeContextEndPrototype
12650   \markdownRendererHeaderAttributeContextEndPrototype
12651 \cs_gset:Npn
12652   \markdownRendererReplacementCharacterPrototype
12653   { \codepoint_str_generate:n { fffd } }
12654 \ExplSyntaxOff
12655 \def\markdownRendererSectionBeginPrototype{}%

```

```

12656 \def\markdownRendererSectionEndPrototype{%
12657 \let\markdownRendererWarningPrototype\markdownWarning
12658 \let\markdownRendererErrorPrototype\markdownError

```

### 3.2.3.1 Raw Attributes

In the raw block and inline raw span renderer prototypes, execute the content with TeX when the raw attribute is `tex`, display the content as markdown when the raw attribute is `md`, and ignore the content otherwise.

```

12659 \ExplSyntaxOn
12660 \cs_new:Nn
12661   \@@_plain_tex_default_input_raw_inline:nn
12662   {
12663     \str_case:nn
12664       { #2 }
12665       {
12666         { md } { \markdownInput{#1} }
12667         { tex } { \markdownEscape{#1} \unskip }
12668       }
12669   }
12670 \cs_new:Nn
12671   \@@_plain_tex_default_input_raw_block:nn
12672   {
12673     \str_case:nn
12674       { #2 }
12675       {
12676         { md } { \markdownInput{#1} }
12677         { tex } { \markdownEscape{#1} }
12678       }
12679   }
12680 \cs_gset:Npn
12681   \markdownRendererInputRawInlinePrototype#1#2
12682   {
12683     \@@_plain_tex_default_input_raw_inline:nn
12684       { #1 }
12685       { #2 }
12686   }
12687 \cs_gset:Npn
12688   \markdownRendererInputRawBlockPrototype#1#2
12689   {
12690     \@@_plain_tex_default_input_raw_block:nn
12691       { #1 }
12692       { #2 }
12693   }
12694 \ExplSyntaxOff

```

### 3.2.3.2 YAML Metadata Renderer Prototypes

To keep track of the current type of structure we inhabit when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_datatypes_seq` stack. At every step of the traversal, the stack will contain one of the following constants at any position  $p$ :

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

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

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

```
12695 \ExplSyntaxOn
12696 \seq_new:N \g_@@_jekyll_data_datatypes_seq
12697 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
12698 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
12699 \tl_const:Nn \c_@@_jekyll_data_scalar_tl { scalar }
```

To keep track of our current place when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_wildcard_absolute_address_seq` stack of keys using the `\markdown_jekyll_data_push_address_segment:n` macro.

```
12700 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
12701 \cs_new:Nn \markdown_jekyll_data_push_address_segment:n
12702 {
12703   \seq_if_empty:NF
12704     \g_@@_jekyll_data_datatypes_seq
12705     {
12706       \seq_get_right:NN
12707         \g_@@_jekyll_data_datatypes_seq
12708         \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.

```
12709   \str_if_eq:NNTF
12710     \l_tmpa_tl
12711     \c_@@_jekyll_data_sequence_tl
12712     {
12713       \seq_put_right:Nn
12714         \g_@@_jekyll_data_wildcard_absolute_address_seq
12715         { * }
12716     }
```

```

12717     {
12718         \seq_put_right:Nn
12719         \g_@@_jekyll_data_wildcard_absolute_address_seq
12720         { #1 }
12721     }
12722 }
12723 }

```

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.

```

12724 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
12725 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
12726 \cs_new:Nn \markdown_jekyll_data_concatenate_address:NN
12727 {
12728     \seq_pop_left:NN #1 \l_tmpa_tl
12729     \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
12730     \seq_put_left:NV #1 \l_tmpa_tl
12731 }
12732 \cs_new:Nn \markdown_jekyll_data_update_address_tls:
12733 {
12734     \markdown_jekyll_data_concatenate_address:NN
12735     \g_@@_jekyll_data_wildcard_absolute_address_seq

```

```

12736     \g_@@_jekyll_data_wildcard_absolute_address_tl
12737     \seq_get_right:NN
12738     \g_@@_jekyll_data_wildcard_absolute_address_seq
12739     \g_@@_jekyll_data_wildcard_relative_address_tl
12740 }

```

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.

```

12741 \cs_new:Nn \markdown_jekyll_data_push:nN
12742 {
12743     \markdown_jekyll_data_push_address_segment:n
12744     { #1 }
12745     \seq_put_right:NV
12746     \g_@@_jekyll_data_datatypes_seq
12747     #2
12748     \markdown_jekyll_data_update_address_tls:
12749 }
12750 \cs_new:Nn \markdown_jekyll_data_pop:
12751 {
12752     \seq_pop_right:NN
12753     \g_@@_jekyll_data_wildcard_absolute_address_seq
12754     \l_tmpa_tl
12755     \seq_pop_right:NN
12756     \g_@@_jekyll_data_datatypes_seq
12757     \l_tmpa_tl
12758     \markdown_jekyll_data_update_address_tls:
12759 }

```

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.

```

12760 \cs_new:Nn \markdown_jekyll_data_set_keyval:nn
12761 {
12762     \keys_set_known:nn
12763     { markdown/jekyllData }
12764     { { #1 } = { #2 } }
12765 }
12766 \cs_generate_variant:Nn
12767     \markdown_jekyll_data_set_keyval:nn
12768     { Vn }
12769 \cs_new:Nn \markdown_jekyll_data_set_keyvals:nn
12770 {
12771     \markdown_jekyll_data_push:nN
12772     { #1 }
12773     \c_@@_jekyll_data_scalar_tl
12774     \markdown_jekyll_data_set_keyval:Vn
12775     \g_@@_jekyll_data_wildcard_absolute_address_tl
12776     { #2 }

```

```

12777     \markdown_jekyll_data_set_keyval:Vn
12778     \g_@@_jekyll_data_wildcard_relative_address_tl
12779     { #2 }
12780     \markdown_jekyll_data_pop:
12781 }

```

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

```

12782 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
12783     \markdown_jekyll_data_push:nN
12784     { #1 }
12785     \c_@@_jekyll_data_sequence_tl
12786 }
12787 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
12788     \markdown_jekyll_data_push:nN
12789     { #1 }
12790     \c_@@_jekyll_data_mapping_tl
12791 }
12792 \def\markdownRendererJekyllDataSequenceEndPrototype{
12793     \markdown_jekyll_data_pop:
12794 }
12795 \def\markdownRendererJekyllDataMappingEndPrototype{
12796     \markdown_jekyll_data_pop:
12797 }
12798 \def\markdownRendererJekyllDataBooleanPrototype#1#2{
12799     \markdown_jekyll_data_set_keyvals:nn
12800     { #1 }
12801     { #2 }
12802 }
12803 \def\markdownRendererJekyllDataEmptyPrototype#1{}
12804 \def\markdownRendererJekyllDataNumberPrototype#1#2{
12805     \markdown_jekyll_data_set_keyvals:nn
12806     { #1 }
12807     { #2 }
12808 }

```

We will process all string scalar values assuming that they may contain markdown markup and are intended for typesetting.

```

12809 \def\markdownRendererJekyllDataProgrammaticStringPrototype#1#2{}
12810 \def\markdownRendererJekyllDataTypographicStringPrototype#1#2{
12811     \markdown_jekyll_data_set_keyvals:nn
12812     { #1 }
12813     { #2 }
12814 }
12815 \ExplSyntaxOff

```

If plain  $\TeX$  is the top layer, we load the [witiko/markdown/defaults](#) plain  $\TeX$  theme with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```

12816 \ExplSyntaxOn
12817 \str_if_eq:VVT
12818   \c_@@_top_layer_tl
12819   \c_@@_option_layer_plain_tex_tl
12820   {
12821     \ExplSyntaxOff
12822     \@@_if_option:nF
12823       { noDefaults }
12824       {
12825         \@@_setup:n
12826           {theme = witiko/markdown/defaults}
12827       }
12828     \ExplSyntaxOn
12829   }
12830 \ExplSyntaxOff

```

### 3.2.4 Lua Snippets

After the `\markdownPrepareLuaOptions` macro has been fully expanded, the `\markdownLuaOptions` macro will expand to a Lua table that contains the plain  $\TeX$  options (see Section 2.2.2) in a format recognized by Lua (see Section 2.1.3).

```

12831 \ExplSyntaxOn
12832 \tl_new:N \g_@@_formatted_lua_options_tl
12833 \cs_new:Nn \@@_format_lua_options:
12834   {
12835     \tl_gclear:N
12836       \g_@@_formatted_lua_options_tl
12837     \seq_map_function:NN
12838       \g_@@_lua_options_seq
12839       \@@_format_lua_option:n
12840   }
12841 \cs_new:Nn \@@_format_lua_option:n
12842   {
12843     \@@_typecheck_option:n
12844       { #1 }
12845     \@@_get_option_type:nN
12846       { #1 }
12847     \l_tmpa_tl
12848     \bool_case_true:nF
12849       {
12850         {
12851           \str_if_eq_p:VV
12852             \l_tmpa_tl

```



```

12853         \c_@@_option_type_boolean_tl ||
12854     \str_if_eq_p:VV
12855         \l_tmpa_tl
12856         \c_@@_option_type_number_tl ||
12857     \str_if_eq_p:VV
12858         \l_tmpa_tl
12859         \c_@@_option_type_counter_tl
12860     }
12861     {
12862         \@@_get_option_value:nN
12863         { #1 }
12864         \l_tmpa_tl
12865         \tl_gput_right:Nx
12866         \g_@@_formatted_lua_options_tl
12867         { #1~::~ \l_tmpa_tl ,~ }
12868     }
12869     {
12870         \str_if_eq_p:VV
12871         \l_tmpa_tl
12872         \c_@@_option_type_clist_tl
12873     }
12874     {
12875         \@@_get_option_value:nN
12876         { #1 }
12877         \l_tmpa_tl
12878         \tl_gput_right:Nx
12879         \g_@@_formatted_lua_options_tl
12880         { #1~::~\c_left_brace_str }
12881         \clist_map_inline:Vn
12882         \l_tmpa_tl
12883         {
12884             \tl_gput_right:Nx
12885             \g_@@_formatted_lua_options_tl
12886             { "##1" ,~ }
12887         }
12888         \tl_gput_right:Nx
12889         \g_@@_formatted_lua_options_tl
12890         { \c_right_brace_str ,~ }
12891     }
12892 }
12893 {
12894     \@@_get_option_value:nN
12895     { #1 }
12896     \l_tmpa_tl
12897     \tl_gput_right:Nx
12898     \g_@@_formatted_lua_options_tl
12899     { #1~::~ " \l_tmpa_tl " ,~ }

```

```

12900     }
12901   }
12902   \cs_generate_variant:Nn
12903     \clist_map_inline:nn
12904     { Vn }
12905   \let\markdownPrepareLuaOptions=\@@_format_lua_options:
12906   \def\markdownLuaOptions#{ \g_@@_formatted_lua_options_tl }
12907   \ExplSyntaxOff

```

The `\markdownPrepare` macro contains the Lua code that is executed prior to any conversion from markdown to plain TeX. It exposes the `convert` function for the use by any further Lua code.

```

12908   \def\markdownPrepare{%
First, ensure that the cacheDir directory exists.
12909     local lfs = require("lfs")
12910     local cacheDir = "\markdownOptionCacheDir"
12911     if not lfs.isdir(cacheDir) then
12912       assert(lfs.mkdir(cacheDir))
12913     end

```

Next, load the `markdown` module and create a converter function using the plain TeX options, which were serialized to a Lua table via the `\markdownLuaOptions` macro.

```

12914     local md = require("markdown")
12915     local convert = md.new(\markdownLuaOptions)
12916   }%

```

The `\markdownCleanup` macro contains the Lua code that is executed after any conversion from markdown to plain TeX.

```

12917   \def\markdownCleanup{%
Remove the options.cacheDir directory if it is empty.
12918     lfs.rmdir(cacheDir)
12919   }%

```

### 3.2.5 Buffering Block-Level Markdown Input

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.

```

12920   \csname newread\endcsname\markdownInputFileStream
12921   \csname newwrite\endcsname\markdownOutputFileStream

```

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

```

12922   \begingroup
12923     \catcode\^^I=12%
12924     \gdef\markdownReadAndConvertTab{^^I}%
12925   \endgroup

```

The `\markdownReadAndConvert` macro is largely a rewrite of the L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> `\filecontents` macro to plain T<sub>E</sub>X.

```
12926 \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 `stripPercentSigns` is enabled.

```
12927 \catcode\^^M=13%
12928 \catcode\^^I=13%
12929 \catcode|=0%
12930 \catcode\|=12%
12931 |catcode@=14%
12932 |catcode|=12@
12933 |gdef|markdownReadAndConvert#1#2{@
12934 |begingroup@
```

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

```
12935 |markdownIfOption{frozenCache}{-}{@
12936 |immediate|openout|markdownOutputFileStream@
12937 |markdownOptionInputTempFileName|relax@
12938 |markdownInfo{@
12939 |Buffering block-level markdown input into the temporary @
12940 |input file "|markdownOptionInputTempFileName" and scanning @
12941 |for the closing token sequence "#1"}@
12942 }@
```

Locally change the category of the special plain T<sub>E</sub>X characters to *other* in order to prevent unwanted interpretation of the input. Change also the category of the space character, so that we can retrieve it unaltered.

```
12943 |def|do##1{|catcode`##1=12}|dospecials@
12944 |catcode`|=12@
12945 |markdownMakeOther@
```

The `\markdownReadAndConvertStripPercentSigns` macro will process the individual lines of output, stripping away leading percent signs (`%`) when `stripPercentSigns` 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.

```
12946 |def|markdownReadAndConvertStripPercentSign##1{@
12947 |markdownIfOption{stripPercentSigns}{-}{@
12948 |if##1%@
12949 |expandafter|expandafter|expandafter@
12950 |markdownReadAndConvertProcessLine@
12951 |else@
12952 |expandafter|expandafter|expandafter@
```

```

12953         |markdownReadAndConvertProcessLine@
12954         |expandafter|expandafter|expandafter##1@
12955     |fi@
12956   }{@
12957     |expandafter@
12958     |markdownReadAndConvertProcessLine@
12959     |expandafter##1@
12960   }@
12961 }@

```

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.

```

12962   |def|markdownReadAndConvertProcessLine##1##2##3|relax{@

```

If we are not reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, store the line in the `inputTempFileName` 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.

```

12963     |ifx|relax##3|relax@
12964     |markdownIfOption{frozenCache}{}{@
12965     |immediate|write|markdownOutputFileStream{##1}@
12966     }@
12967     |else@

```

When the ending token sequence appears in the line, make the next newline character close the `inputTempFileName` file, return the character categories back to the former state, convert the `inputTempFileName` file from markdown to plain T<sub>E</sub>X, `\input` the result of the conversion, and expand the ending control sequence.

```

12968     |def^^M{@
12969     |markdownInfo{The ending token sequence was found}@
12970     |markdownIfOption{frozenCache}{}{@
12971     |immediate|closeout|markdownOutputFileStream@
12972     }@
12973     |endgroup@
12974     |markdownInput{@
12975     |markdownOptionOutputDir@
12976     /|markdownOptionInputTempFileName@
12977     }@
12978     #2}@
12979   |fi@

```

Repeat with the next line.

```

12980     ^^M}@

```

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

```

12981   |catcode`|^I=13@

```

```
12982 |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.

```
12983 |catcode`\^^M=13@
12984 |def^^M##1^^M{@
12985 |def^^M###1^^M{@
12986 |markdownReadAndConvertStripPercentSign####1#1|relax}@
12987 ^^M}@
12988 ^^M}@
```

Reset the character categories back to the former state.

```
12989 |endgroup
```

Use the `lt3luabridge` library to define the `\markdownLuaExecute` macro, which takes in a Lua scripts and expands to the standard output produced by its execution.

```
12990 \ExplSyntaxOn
12991 \cs_new:Npn
12992 \markdownLuaExecute
12993 #1
12994 {
12995 \int_compare:nNtT
12996 { \g_luabridge_method_int }
12997 =
12998 { \c_luabridge_method_shell_int }
12999 {
13000 \sys_if_shell_unrestricted:F
13001 {
13002 \sys_if_shell:TF
13003 {
13004 \msg_error:nn
13005 { markdown }
13006 { restricted-shell-access }
13007 }
13008 {
13009 \msg_error:nn
13010 { markdown }
13011 { disabled-shell-access }
13012 }
13013 }
13014 }
13015 \str_gset:NV
13016 \g_luabridge_output_dirname_str
13017 \markdownOptionOutputDir
13018 \luabridge_now:e
13019 { #1 }
13020 }
```

```

13021 \cs_generate_variant:Nn
13022   \msg_new:nnnn
13023   { nnnV }
13024 \tl_set:Nn
13025   \l_tmpa_tl
13026   {
13027     You~may~need~to~run~TeX~with~the~---shell-escape~or~the~
13028     --enable-write18~flag,~or~write~shell_escape=t~in~the~
13029     texmf.cnf~file.
13030   }
13031 \msg_new:nnnV
13032   { markdown }
13033   { restricted-shell-access }
13034   { Shell~escape~is~restricted }
13035   \l_tmpa_tl
13036 \msg_new:nnnV
13037   { markdown }
13038   { disabled-shell-access }
13039   { Shell~escape~is~disabled }
13040   \l_tmpa_tl
13041 \ExplSyntaxOff

```

### 3.2.6 Buffering Inline Markdown Input

This section describes the implementation of the macro `\markinline`.

```

13042 \ExplSyntaxOn
13043 \tl_new:N
13044   \g_@@_after_markinline_tl
13045 \tl_gset:Nn
13046   \g_@@_after_markinline_tl
13047   { \unskip }
13048 \cs_new:Npn
13049   \markinline
13050   {

```

Locally change the category of the special plain  $\TeX$  characters to *other* in order to prevent unwanted interpretation of the input markdown text as  $\TeX$  code.

```

13051   \group_begin:
13052   \cctab_select:N
13053     \c_other_cctab

```

Unless we are reading markdown documents from the frozen cache, open the file `inputTempFileName` for writing.

```

13054   \@@_if_option:nF
13055     { frozenCache }
13056     {
13057       \immediate

```

```

13058         \openout
13059         \markdownOutputFileStream
13060         \markdownOptionInputTempFileName
13061         \relax
13062     \msg_info:nne
13063     { markdown }
13064     { buffering-markinline }
13065     { \markdownOptionInputTempFileName }
13066 }

```

Peek ahead and extract the inline markdown text.

```

13067     \peek_regex_replace_once:nnF
13068     { { (.*) } }
13069     {

```

Unless we are reading markdown documents from the frozen cache, store the text in the file `inputTempFileName` and close it.

```

13070         \c { @@_if_option:nF }
13071         \cB { frozenCache \cE }
13072         \cB {
13073             \c { immediate }
13074             \c { write }
13075             \c { markdownOutputFileStream }
13076             \cB { \1 \cE }
13077             \c { immediate }
13078             \c { closeout }
13079             \c { markdownOutputFileStream }
13080         \cE }

```

Reset the category codes and `\input` the result of the conversion.

```

13081     \c { group_end: }
13082     \c { group_begin: }
13083     \c { @@_setup:n }
13084     \cB { contentLevel = inline \cE }
13085     \c { markdownInput }
13086     \cB {
13087         \c { markdownOptionOutputDir } /
13088         \c { markdownOptionInputTempFileName }
13089     \cE }
13090     \c { group_end: }
13091     \c { tl_use:N }
13092     \c { g_@@_after_markinline_tl }
13093 }
13094 {
13095     \msg_error:nn
13096     { markdown }
13097     { markinline-peek-failure }
13098 }
\group_end:

```

```

13099     \tl_use:N
13100     \g_@@_after_markinline_tl
13101   }
13102 }
13103 \msg_new:nnn
13104 { markdown }
13105 { buffering-markinline }
13106 { Buffering~inline~markdown~input~into~
13107   the~temporary~input~file~"#1". }
13108 \msg_new:nnnn
13109 { markdown }
13110 { markinline-peek-failure }
13111 { Use-of~\iow_char:N \\ markinline~doesn't~match~its~definition }
13112 { The~macro~should~be~followed~by~inline~
13113   markdown~text~in~curly~braces }
13114 \ExplSyntaxOff

```

### 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 T<sub>E</sub>X.

```

13115 \ExplSyntaxOn
13116 \cs_new:Npn
13117   \markdownInput
13118   #1
13119   {
13120     \@@_if_option:nTF
13121     { frozenCache }
13122     {
13123       \markdownInputRaw
13124       { #1 }
13125     }
13126     {

```

If the file does not exist in the current directory, we will search for it in the directories specified in `\l_file_search_path_seq`. On L<sup>A</sup>T<sub>E</sub>X, this also includes the directories specified in `\input@path`.

```

13127     \tl_set:Nx
13128     \l_tmpa_tl
13129     { #1 }
13130     \file_get_full_name:VNTF
13131     \l_tmpa_tl
13132     \l_tmpb_tl
13133     {
13134       \exp_args:NV
13135       \markdownInputRaw

```



```

13136         \l_tmpb_tl
13137     }
13138     {
13139         \msg_error:nnV
13140         { markdown }
13141         { markdown-file-does-not-exist }
13142         \l_tmpa_tl
13143     }
13144 }
13145 }
13146 \msg_new:nnn
13147 { markdown }
13148 { markdown-file-does-not-exist }
13149 {
13150     Markdown~file~#1~does~not~exist
13151 }
13152 \ExplSyntaxOff
13153 \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. Furthermore, use the ampersand symbol to specify parameters.

```

13154 \catcode`\|=0%
13155 \catcode`\|=12%
13156 \catcode`\&=6%
13157 \gdef|markdownInputRaw#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.

```

13158 |begingroup
13159 |catcode`|=12

```

Furthermore, also change the category code of the hash sign (#) to other, so that it's safe to tokenize the plain TeX output without mistaking hash signs with TeX's parameter numbers.

```

13160 |catcode`|#=12

```

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

```

13161 |markdownIfOption{frozenCache}{%
13162 |ifnum|markdownOptionFrozenCacheCounter=0|relax
13163 |markdownInfo{Reading frozen cache from
13164 "|markdownOptionFrozenCacheFileName"%
13165 |input|markdownOptionFrozenCacheFileName|relax
13166 |fi
13167 |markdownInfo{Including markdown document number
13168 "|the|markdownOptionFrozenCacheCounter" from frozen cache}%

```

```

13169     |csname markdownFrozenCache%
13170     |the|markdownOptionFrozenCacheCounter|endcsname
13171     |global|advance|markdownOptionFrozenCacheCounter by 1|relax
13172     }{%
13173     |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 `LATEX Mk` to track changes to the markdown document.

```

13174     |openin|markdownInputFileStream&1
13175     |closein|markdownInputFileStream
13176     |markdownPrepareLuaOptions
13177     |markdownLuaExecute{%
13178     |markdownPrepare
13179     local file = assert(io.open("&1", "r"),
13180     [[Could not open file "&1" for reading]])
13181     local input = assert(file:read("*a"))
13182     assert(file:close())
13183     print(convert(input))
13184     |markdownCleanup}%

```

If we are finalizing the frozen cache, increment `frozenCacheCounter`.

```

13185     |markdownIfOption{finalizeCache}{%
13186     |global|advance|markdownOptionFrozenCacheCounter by 1|relax}{}%
13187     }%
13188     |endgroup
13189     }%
13190 |endgroup

```

The `\markdownEscape` macro resets the category codes of the percent sign and the hash sign back to comment and parameter, respectively, before using the `\input` built-in of `TEX` to execute a `TEX` document in the middle of a markdown document fragment.

```

13191 \gdef\markdownEscape#1{%
13192 \catcode`\%=14\relax
13193 \catcode`\#=6\relax
13194 \input #1\relax
13195 \catcode`\%=12\relax
13196 \catcode`\#=12\relax
13197 }%

```

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

The L<sup>A</sup>T<sub>E</sub>X implementation 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 [12, Section 9]. As a consequence, we can directly reuse the existing plain T<sub>E</sub>X implementation.

```

13198 \def\markdownVersionSpace{ }%

```

```

13199 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
13200 \markdownVersion\markdownVersionSpace markdown renderer]%

```

### 3.3.1 Typesetting Markdown

The `\markinlinePlainTeX` macro is used to store the original plain  $\TeX$  implementation of the `\markinline` macro. The `\markinline` macro is then redefined to accept an optional argument with options recognized by the  $\LaTeX$  interface (see Section 2.3.2).

```

13201 \ExplSyntaxOn
13202 \cs_gset_eq:NN
13203 \markinlinePlainTeX
13204 \markinline
13205 \cs_gset:Npn
13206 \markinline
13207 {
13208 \peek_regex_replace_once:nn
13209 { ( \[ (.*) \] ) ? }
13210 {

```

Apply the options locally.

```

13211 \c { group_begin: }
13212 \c { @@_setup:n }
13213 \cB { \2 \cE }
13214 \c { tl_put_right:Nn }
13215 \c { g_@@_after_markinline_tl }
13216 \cB { \c { group_end: } \cE }
13217 \c { markinlinePlainTeX }
13218 }
13219 }
13220 \ExplSyntaxOff

```

The `\markdownInputPlainTeX` macro is used to store the original plain  $\TeX$  implementation of the `\markdownInput` macro. The `\markdownInput` macro is then redefined to accept an optional argument with options recognized by the  $\LaTeX$  interface (see Section 2.3.2).

```

13221 \let\markdownInputPlainTeX\markdownInput
13222 \renewcommand\markdownInput [2] [] {%
13223 \begingroup
13224 \markdownSetup{#1}%
13225 \markdownInputPlainTeX{#2}%
13226 \endgroup}%

```

The `markdown`, and `markdown*`  $\LaTeX$  environments are implemented using the `\markdownReadAndConvert` macro.

```

13227 \ExplSyntaxOn
13228 \renewenvironment

```

```
13229 { markdown }
13230 {
```

In our implementation of the `markdown` L<sup>A</sup>T<sub>E</sub>X environment, we want to distinguish between the following two cases:

|   |                               |
|---|-------------------------------|
| <code>\begin{markdown} [smartEllipses]</code> | <code>\begin{markdown}</code> |
| <code>% This is an optional argument ^</code> | <code>[smartEllipses]</code>  |
| <code>% ...</code>                            | <code>% ^ This is link</code> |
| <code>\end{markdown}</code>                   | <code>\end{markdown}</code>   |

Therefore, we cannot use the built-in L<sup>A</sup>T<sub>E</sub>X support for environments with optional arguments or packages such as `xparse`. Instead, we must read the optional argument manually and prevent reading past the end of a line.

To prevent reading past the end of a line when looking for the optional argument of the `markdown` L<sup>A</sup>T<sub>E</sub>X environment and accidentally tokenizing markdown text, we change the category code of carriage return (`\r`, ASCII character 13 in decimal) from 5 (end of line).

While any category code other than 5 (end of line) would work, we switch to the category 13 (active), which is also used by the `\markdownReadAndConvert` macro. This is necessary if we read until the end of a line, because then the carriage return character will be produced by T<sub>E</sub>X via the `\endlinechar` plain T<sub>E</sub>X macro and it needs to have the correct category code, so that `\markdownReadAndConvert` processes it correctly.

```
13231 \group_begin:
13232 \char_set_catcode_active:n { 13 }
```

To prevent doubling the hash signs (`#`, ASCII code 35 in decimal), we switch its category from 6 (parameter) to 12 (letter).

```
13233 \char_set_catcode_letter:n { 35 }
```

After we have matched the opening `[` that begins the optional argument, we accept carriage returns as well.

```
13234 \peek_regex_replace_once:nnF
13235 { \ *\[ \r*([^\r]*)\][^\r]* }
13236 {
```

After we have matched the optional argument, we switch back the category code of carriage returns and hash signs and we retokenize the content. This will cause single new lines to produce a space token and multiple new lines to produce `\par` tokens. Furthermore, this will cause hash signs followed by a number to be recognized as parameter numbers, which is necessary when we use the optional argument to redefine token renderers and token renderer prototypes.

```
13237 \c { group_end: }
13238 \c { t1_set_rescan:Nnn } \c { l_tmpa_t1 } { } { \1 }
```

Then, we pass the retokenized content to the `\markdownSetup` macro.

```
13239     \c { @@_setup:V } \c { l_tmpa_tl }
```

Finally, regardless of whether or not we have matched the optional argument, we let the `\markdownReadAndConvert` macro process the rest of the  $\LaTeX$  environment.

We also make provision for using the `\markdown` command as a part of a different  $\LaTeX$  environment as follows:

```
\newenvironment{foo}%
    {code before \markdown[some, options]}%
    {\markdownEnd code after}
```

```
13240     \c { exp_args:NV }
13241     \c { markdownReadAndConvert@ }
13242     \c { @currenvir }
13243 }
13244 {
13245     \group_end:
13246     \exp_args:NV
13247     \markdownReadAndConvert@
13248     \@currenvir
13249 }
13250 }
13251 { \markdownEnd }
13252 \renewenvironment
13253 { markdown* }
13254 [ 1 ]
13255 {
13256     \msg_warning:nnn
13257     { markdown }
13258     { latex-markdown-star-deprecated }
13259     { #1 }
13260     \@_setup:n
13261     { #1 }
13262     \markdownReadAndConvert@
13263     { markdown* }
13264 }
13265 { \markdownEnd }
13266 \msg_new:nnn
13267 { markdown }
13268 { latex-markdown-star-deprecated }
13269 {
13270     The~markdown*~LaTeX~environment~has~been~deprecated~and~will~
13271     be~removed~in~the~next~major~version~of~the~Markdown~package.
13272 }
13273 \cs_generate_variant:Nn
```

```

13274 \@@_setup:n
13275 { V }
13276 \ExplSyntaxOff
13277 \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*.

```

13278 \catcode`\|=0\catcode`\<=1\catcode`\>=2%
13279 \catcode`\|=12\catcode`\{=12\catcode`\}=12%
13280 |gdef|markdownReadAndConvert@#1<%
13281 |markdownReadAndConvert<\end{#1}>%
13282 <|end<#1>>%
13283 |endgroup

```

### 3.3.2 Options

The supplied package options are processed using the `markdownSetup` macro.

```

13284 \DeclareOption*{%
13285 \expandafter\markdownSetup\expandafter{\CurrentOption}}%
13286 \ProcessOptions\relax

```

### 3.3.3 Themes

This section overrides the plain  $\TeX$  implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in  $\LaTeX$  themes provided with the Markdown package.

```

13287 \ExplSyntaxOn
13288 \cs_gset:Nn
13289 \@@_load_theme:nn
13290 {

```

If the Markdown package has already been loaded, determine whether a file named `markdowntheme<munged theme name>.sty` exists and whether we are still in the preamble.

```

13291 \ifmarkdownLaTeXLoaded
13292 \ifx\@onlypreamble\@notprerr

```

If both conditions are true does, end with an error, since we cannot load  $\LaTeX$  themes after the preamble. Otherwise, try loading a plain  $\TeX$  theme instead.

```

13293 \file_if_exist:nTF
13294 { markdown theme #2.sty }
13295 {
13296 \msg_error:nnn
13297 { markdown }

```

```

13298         { latex-theme-after-preamble }
13299         { #1 }
13300     }
13301     {
13302         \@_plain_tex_load_theme:nn
13303         { #1 }
13304         { #2 }
13305     }
13306     \else

```

If the Markdown package has already been loaded but we are still in the preamble, load a L<sup>A</sup>T<sub>E</sub>X theme if it exists or load a plain T<sub>E</sub>X theme otherwise.

```

13307     \file_if_exist:nTF
13308     { markdown theme #2.sty }
13309     {
13310         \msg_info:nnn
13311         { markdown }
13312         { loading-latex-theme }
13313         { #1 }
13314         \RequirePackage
13315         { markdown theme #2 }
13316     }
13317     {
13318         \@_plain_tex_load_theme:nn
13319         { #1 }
13320         { #2 }
13321     }
13322     \fi
13323     \else

```

If the Markdown package has not yet been loaded, postpone the loading until the Markdown package has finished loading.

```

13324     \msg_info:nnn
13325     { markdown }
13326     { theme-loading-postponed }
13327     { #1 }
13328     \AtEndOfPackage
13329     {
13330         \@_load_theme:nn
13331         { #1 }
13332         { #2 }
13333     }
13334     \fi
13335 }
13336 \msg_new:nnn
13337 { markdown }
13338 { theme-loading-postponed }
13339 {

```

```

13340     Postponing~loading~Markdown~theme~#1~until~
13341     Markdown~package~has~finished~loading
13342   }
13343   \msg_new:nnn
13344     { markdown }
13345     { loading-latex-theme }
13346     { Loading~LaTeX~Markdown~theme~#1 }
13347   \cs_generate_variant:Nn
13348     \msg_new:nnnn
13349     { nnVV }
13350   \tl_set:Nn
13351     \l_tmpa_tl
13352     { Cannot~load~LaTeX~Markdown~theme~#1~after~ }
13353   \tl_put_right:NV
13354     \l_tmpa_tl
13355     \c_backslash_str
13356   \tl_put_right:Nn
13357     \l_tmpa_tl
13358     { begin{document} }
13359   \tl_set:Nn
13360     \l_tmpb_tl
13361     { Load~Markdown~theme~#1~before~ }
13362   \tl_put_right:NV
13363     \l_tmpb_tl
13364     \c_backslash_str
13365   \tl_put_right:Nn
13366     \l_tmpb_tl
13367     { begin{document} }
13368   \msg_new:nnVV
13369     { markdown }
13370     { latex-theme-after-preamble }
13371     \l_tmpa_tl
13372     \l_tmpb_tl
13373   \ExplSyntaxOff

```

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

```
13374 \markdownSetup{fencedCode}%
```

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

```
13375 \RequirePackage{ifthen,grffile}
```

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

```
13376 \let\markdown@witiko@dot@oldRendererInputFencedCodePrototype
13377   \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 `frozenCache` plain T<sub>E</sub>X option is disabled and the code block has not been previously typeset:



```

13378 \renewcommand\markdownRendererInputFencedCodePrototype[3]{%
13379   \def\next##1 ##2\relax{%
13380     \ifthenelse{\equal{##1}{dot}}{%
13381       \markdownIfOption{frozenCache}{}{%
13382         \immediate\write18{%
13383           if ! test -e #1.pdf.source || ! diff #1 #1.pdf.source;
13384           then
13385             dot -Tpdf -o #1.pdf #1;
13386             cp #1 #1.pdf.source;
13387           fi}}%

```

We include the typeset image using the image token renderer:

```

13388   \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:

```

13389   }{%
13390     \markdown@witiko@dot@oldRendererInputFencedCodePrototype
13391     {#1}{#2}{#3}%
13392   }%
13393 }%
13394 \next#2 \relax}%

```

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

```

13395 \let\markdown@witiko@graphicx@http@oldRendererImagePrototype
13396   \markdownRendererImagePrototype

```

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

```

13397 \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.

```

13398 \newcount\markdown@witiko@graphicx@http@counter
13399 \markdown@witiko@graphicx@http@counter=0
13400 \newcommand\markdown@witiko@graphicx@http@filename{%
13401   \markdownOptionCacheDir/witiko_graphicx_http%
13402   .\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.

```

13403 \newcommand\markdown@witiko@graphicx@http@download[2]{%
13404   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:

```
13405 \begingroup
13406 \catcode`\%=12
13407 \catcode`\^^A=14
```

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

```
13408 \global\def\markdownRendererImagePrototype#1#2#3#4{^^A
13409   \begingroup
13410     \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 `frozenCache` plain T<sub>E</sub>X option is disabled:

```
13411     \markdownIfOption{frozenCache}{}{^^A
13412       \immediate\write18{^^A
13413         mkdir -p "\markdownOptionCacheDir";
13414         if printf '%s' "#3" | grep -q -E '^https?:';
13415         then
```

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

```
13416         OUTPUT_PREFIX="\markdownOptionCacheDir";
13417         OUTPUT_BODY="$(printf '%s' '#3' | md5sum | cut -d' ' -f1)";
13418         OUTPUT_SUFFIX="$(printf '%s' '#3' | sed 's/.*[.]/')";
13419         OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";
```

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

```
13420         if ! [ -e "$OUTPUT" ];
13421         then
13422           \markdown@witiko@graphicx@http@download{'#3'}{"$OUTPUT"};
13423           printf '%s' "$OUTPUT" > "\filename";
13424         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:

```
13425         else
13426           printf '%s' '#3' > "\filename";
13427         fi}}^^A
```

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

```
13428     \CatchFileDef{\filename}{\filename}{\endlinechar=-1}^^A
13429     \markdown@witiko@graphicx@http@oldRendererImagePrototype^^A
13430     {#1}{#2}{\filename}{#4}^^A
13431   \endgroup
13432   \global\advance\markdown@witiko@graphicx@http@counter by 1\relax}^^A
13433 \endgroup
```

The `witiko/markdown/defaults` L<sup>A</sup>T<sub>E</sub>X theme provides default definitions for token renderer prototypes. First, the L<sup>A</sup>T<sub>E</sub>X theme loads the plain T<sub>E</sub>X theme with the default definitions for plain T<sub>E</sub>X:

```
13434 \markdownLoadPlainTeXTheme
```

Next, the L<sup>A</sup>T<sub>E</sub>X theme overrides some of the plain T<sub>E</sub>X definitions. See Section 3.3.4 for the actual definitions.

### 3.3.4 Token Renderer Prototypes

The following configuration should be considered placeholder. If the option `plain` has been enabled (see Section 2.2.2.3), none of the definitions will take effect.

```
13435 \markdownIfOption{plain}{\iffalse}{\iftrue}
```

If either the `tightLists` or the `fancyLists` Lua option is enabled and the current document class is not beamer, then load the paralist package.

```
13436 \@ifclassloaded{beamer}{}{%
13437   \markdownIfOption{tightLists}{\RequirePackage{paralist}}{}%
13438   \markdownIfOption{fancyLists}{\RequirePackage{paralist}}{}%
13439 }
```

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.

```
13440 \ExplSyntaxOn
13441 \@ifpackageloaded{paralist}{
13442   \tl_new:N
13443     \l_@@_latex_fancy_list_item_label_number_style_tl
13444   \tl_new:N
13445     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
13446   \cs_new:Nn
13447     \@@_latex_fancy_list_item_label_number:nn
13448     {
13449       \str_case:nn
13450         { #1 }
13451         {
13452           { Decimal } { #2 }
13453           { LowerRoman } { \int_to_roman:n { #2 } }
13454           { UpperRoman } { \int_to_Roman:n { #2 } }
13455           { LowerAlpha } { \int_to_alph:n { #2 } }
13456           { UpperAlpha } { \int_to_Alph:n { #2 } }
13457         }
13458     }
13459   \cs_new:Nn
13460     \@@_latex_fancy_list_item_label_delimiter:n
13461     {
13462       \str_case:nn
```

```

13463     { #1 }
13464     {
13465         { Default } { . }
13466         { OneParen } { ) }
13467         { Period } { . }
13468     }
13469 }
13470 \cs_new:Nn
13471   \@@_latex_fancy_list_item_label:nnn
13472   {
13473     \@@_latex_fancy_list_item_label_number:nn
13474     { #1 }
13475     { #3 }
13476     \@@_latex_fancy_list_item_label_delimiter:n
13477     { #2 }
13478   }
13479 \cs_new:Nn
13480   \@@_latex_paralist_style:nn
13481   {
13482     \str_case:nn
13483     { #1 }
13484     {
13485       { Decimal } { 1 }
13486       { LowerRoman } { i }
13487       { UpperRoman } { I }
13488       { LowerAlpha } { a }
13489       { UpperAlpha } { A }
13490     }
13491     \@@_latex_fancy_list_item_label_delimiter:n
13492     { #2 }
13493   }
13494 \markdownSetup{rendererPrototypes={

```

Make tight bullet lists a little less compact by adding extra vertical space above and below them.

```

13495   ulBeginTight = {%
13496     \group_begin:
13497     \pltopsep=\topsep
13498     \plpartopsep=\partopsep
13499     \begin{compactitem}
13500   },
13501   ulEndTight = {
13502     \end{compactitem}
13503     \group_end:
13504   },
13505   fancyOlBegin = {
13506     \group_begin:

```

```

13507     \tl_set:Nn
13508         \l_@@_latex_fancy_list_item_label_number_style_tl
13509         { #1 }
13510     \tl_set:Nn
13511         \l_@@_latex_fancy_list_item_label_delimiter_style_tl
13512         { #2 }
13513     \@@_if_option:nTF
13514         { startNumber }
13515         {
13516             \tl_set:Nn
13517                 \l_tmpa_tl
13518                 { \begin{enumerate} }
13519         }
13520         {
13521             \tl_set:Nn
13522                 \l_tmpa_tl
13523                 { \begin{enumerate}[ ] }
13524             \tl_put_right:Nx
13525                 \l_tmpa_tl
13526                 { \@@_latex_paralist_style:nm { #1 } { #2 } }
13527             \tl_put_right:Nn
13528                 \l_tmpa_tl
13529                 { ] }
13530         }
13531     \tl_use:N
13532         \l_tmpa_tl
13533 },
13534 fancyO1End = {
13535     \end{enumerate}
13536     \group_end:
13537 },

```

Make tight ordered lists a little less compact by adding extra vertical space above and below them.

```

13538     olBeginTight = {%
13539         \group_begin:
13540         \plpartopsep=\partopsep
13541         \pltopsep=\topsep
13542         \begin{compactenum}
13543     },
13544     olEndTight = {
13545         \end{compactenum}
13546         \group_end:
13547     },
13548     fancyO1BeginTight = {
13549         \group_begin:
13550         \tl_set:Nn

```

```

13551     \l_@@_latex_fancy_list_item_label_number_style_tl
13552     { #1 }
13553   \tl_set:Nn
13554     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
13555     { #2 }
13556   \@@_if_option:nTF
13557     { startNumber }
13558     {
13559     \tl_set:Nn
13560       \l_tmpa_tl
13561       { \begin{compactenum} }
13562     }
13563     {
13564     \tl_set:Nn
13565       \l_tmpa_tl
13566       { \begin{compactenum}[ ] }
13567     \tl_put_right:Nx
13568       \l_tmpa_tl
13569       { \@@_latex_paralist_style:nm { #1 } { #2 } }
13570     \tl_put_right:Nn
13571       \l_tmpa_tl
13572       { ] }
13573     }
13574   \tl_put_left:Nn
13575     \l_tmpa_tl
13576     {
13577     \plpartopsep=\partopsep
13578     \pltopsep=\topsep
13579     }
13580   \tl_use:N
13581     \l_tmpa_tl
13582 },
13583 fancyO1EndTight = {
13584   \end{compactenum}
13585   \group_end:
13586 },
13587 fancyO1ItemWithNumber = {
13588   \item
13589   [
13590     \@@_latex_fancy_list_item_label:VVn
13591     \l_@@_latex_fancy_list_item_label_number_style_tl
13592     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
13593     { #1 }
13594   ]
13595 },

```

Make tight definition lists a little less compact by adding extra vertical space above and below them.

```

13596     dlBeginTight = {
13597         \group_begin:
13598         \plpartopsep=\partopsep
13599         \pltopsep=\topsep
13600         \begin{compactdesc}
13601     },
13602     dlEndTight = {
13603         \end{compactdesc}
13604         \group_end:
13605     }}
13606 \cs_generate_variant:Nn
13607 \@@_latex_fancy_list_item_label:nnn
13608 { VVn }
13609 }{
13610 \markdownSetup{rendererPrototypes={
13611     ulBeginTight = {\markdownRendererUlBegin},
13612     ulEndTight = {\markdownRendererUlEnd},
13613     fancyOlBegin = {\markdownRendererOlBegin},
13614     fancyOlEnd = {\markdownRendererOlEnd},
13615     olBeginTight = {\markdownRendererOlBegin},
13616     olEndTight = {\markdownRendererOlEnd},
13617     fancyOlBeginTight = {\markdownRendererOlBegin},
13618     fancyOlEndTight = {\markdownRendererOlEnd},
13619     dlBeginTight = {\markdownRendererDlBegin},
13620     dlEndTight = {\markdownRendererDlEnd}}}
13621 }
13622 \ExplSyntaxOff
13623 \RequirePackage{amsmath}

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

13624 \@ifpackageloaded{unicode-math}{
13625     \markdownSetup{rendererPrototypes={
13626         untickedBox = {\$ \mdlgwhtsquare$},
13627     }}
13628 }{
13629     \RequirePackage{amssymb}
13630     \markdownSetup{rendererPrototypes={
13631         untickedBox = {\$ \square$},
13632     }}
13633 }
13634 \RequirePackage{csvsimple}
13635 \RequirePackage{fancyvrb}
13636 \RequirePackage{graphicx}
13637 \markdownSetup{rendererPrototypes={

```

```

13638  hardLineBreak = {\},
13639  leftBrace = {\textbraceleft},
13640  rightBrace = {\textbraceright},
13641  dollarSign = {\textdollar},
13642  underscore = {\textunderscore},
13643  circumflex = {\textasciicircum},
13644  backslash = {\textbackslash},
13645  tilde = {\textasciitilde},
13646  pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by T<sub>E</sub>X during the typesetting. Therefore, even if we don't know whether a span of text is part of math formula or not when we are parsing markdown,<sup>34</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.

```

13647  codeSpan = {%
13648    \ifmmode
13649      \text{#1}%
13650    \else
13651      \texttt{#1}%
13652    \fi
13653  }}}
13654  \ExplSyntaxOn
13655  \markdownSetup{
13656    rendererPrototypes = {
13657      contentBlock = {
13658        \str_case:nnF
13659          { #1 }
13660          {
13661            { csv }
13662            {
13663              \begin{table}
13664                \begin{center}
13665                  \csvautotabular{#3}
13666                \end{center}
13667                \tl_if_empty:nF
13668                  { #4 }
13669                  { \caption{#4} }
13670              \end{table}
13671            }
13672            { tex } { \markdownEscape{#3} }
13673          }

```

---

<sup>34</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.



```

13674     { \markdownInput{#3} }
13675   },
13676 },
13677 }
13678 \ExplSyntaxOff
13679 \markdownSetup{rendererPrototypes={
13680   image = {%
13681     \begin{figure}%
13682       \begin{center}%
13683         \includegraphics[alt={#1}]{#3}%
13684       \end{center}%
13685       \ifx\empty#4\empty\else
13686         \caption{#4}%
13687       \fi
13688     \end{figure}},
13689   ulBegin = {\begin{itemize}},
13690   ulEnd = {\end{itemize}},
13691   olBegin = {\begin{enumerate}},
13692   olItem = {\item{}},
13693   olItemWithNumber = {\item[#1.]},
13694   olEnd = {\end{enumerate}},
13695   dlBegin = {\begin{description}},
13696   dlItem = {\item[#1]},
13697   dlEnd = {\end{description}},
13698   emphasis = {\emph{#1}},
13699   tickedBox = {\$\boxtimes$},
13700   halfTickedBox = {\$\boxdot$}}

```

If HTML identifiers appear after a heading, we make them produce `\label` macros.

```

13701 \ExplSyntaxOn
13702 \seq_new:N
13703   \l_@@_header_identifiers_seq
13704 \markdownSetup
13705   {
13706     rendererPrototypes = {
13707       headerAttributeContextBegin = {
13708         \markdownSetup
13709         {
13710           rendererPrototypes = {
13711             attributeIdentifier = {
13712               \seq_put_right:Nn
13713                 \l_@@_header_identifiers_seq
13714                 { #1 }
13715             },
13716           },
13717         }
13718       },
13719       headerAttributeContextEnd = {

```

```

13720     \seq_map_inline:Nn
13721     \l_@@_header_identifiers_seq
13722     { \label { ##1 } }
13723     \seq_clear:N
13724     \l_@@_header_identifiers_seq
13725   },
13726 },
13727 }

```

If the `unnumbered` HTML class (or the `{-}` shorthand) appears after a heading the heading and all its subheadings will be unnumbered.

```

13728 \bool_new:N
13729 \l_@@_header_unnumbered_bool
13730 \markdownSetup
13731 {
13732   rendererPrototypes = {
13733     headerAttributeContextBegin += {
13734       \markdownSetup
13735       {
13736         rendererPrototypes = {
13737           attributeClassName = {
13738             \bool_if:nT
13739             {
13740               \str_if_eq_p:nn
13741               { ##1 }
13742               { unnumbered } &&
13743               ! \l_@@_header_unnumbered_bool
13744             }
13745             {
13746               \group_begin:
13747               \bool_set_true:N
13748               \l_@@_header_unnumbered_bool
13749               \c@secnumdepth = 0
13750               \markdownSetup
13751               {
13752                 rendererPrototypes = {
13753                   sectionBegin = {
13754                     \group_begin:
13755                   },
13756                   sectionEnd = {
13757                     \group_end:
13758                   },
13759                 },
13760               }
13761             }
13762           },
13763         },

```

```

13764     }
13765   },
13766 },
13767 }
13768 \ExplSyntaxOff
13769 \markdownSetup{rendererPrototypes={
13770   superscript = {\textsuperscript{#1}},
13771   subscript = {\textsubscript{#1}},
13772   blockQuoteBegin = {\begin{quotation}},
13773   blockQuoteEnd = {\end{quotation}},
13774   inputVerbatim = {\VerbatimInput{#1}},
13775   thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
13776   note = {\footnote{#1}}}}

```

### 3.3.4.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

13777 \RequirePackage{ltxcmds}
13778 \ExplSyntaxOn
13779 \cs_gset:Npn
13780   \markdownRendererInputFencedCodePrototype#1#2#3
13781   {
13782     \tl_if_empty:nTF
13783       { #2 }
13784     { \markdownRendererInputVerbatim{#1} }

```

Otherwise, extract the first word of the infostring and treat it as the name of the programming language in which the code block is written.

```

13785     {
13786       \regex_extract_once:nnN
13787         { \w* }
13788         { #2 }
13789         \l_tmpa_seq
13790       \seq_pop_left:NN
13791         \l_tmpa_seq
13792         \l_tmpa_tl

```

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

```

13793     \ltx@ifpackageloaded
13794       { minted }
13795     {
13796       \catcode`\#=6\relax
13797       \exp_args:NV
13798         \inputminted
13799         \l_tmpa_tl
13800         { #1 }
13801       \catcode`\#=12\relax
13802     }

```

```

13803         {
    When the listings package is loaded, use it for syntax highlighting.
13804         \ltx@ifpackageloaded
13805         { listings }
13806         { \lstinputlisting[language=\l_tmpa_tl]{#1} }

    When neither the listings package nor the minted package is loaded, act as though
    no infostring were given.
13807         { \markdownRendererInputFencedCode{#1}{}{} }
13808     }
13809 }
13810 }
13811 \ExplSyntaxOff

    Support the nesting of strong emphasis.
13812 \ExplSyntaxOn
13813 \def\markdownLATEXStrongEmphasis#1{%
13814     \str_if_in:NnTF
13815     \f@series
13816     { b }
13817     { \textnormal{#1} }
13818     { \textbf{#1} }
13819 }
13820 \ExplSyntaxOff
13821 \markdownSetup{rendererPrototypes={strongEmphasis={%
13822     \protect\markdownLATEXStrongEmphasis{#1}}}}

    Support LATEX document classes that do not provide chapters.
13823 \@ifundefined{chapter}{%
13824     \markdownSetup{rendererPrototypes = {
13825         headingOne = {\section{#1}},
13826         headingTwo = {\subsection{#1}},
13827         headingThree = {\subsubsection{#1}},
13828         headingFour = {\paragraph{#1}},
13829         headingFive = {\subparagraph{#1}}}}
13830 }{%
13831     \markdownSetup{rendererPrototypes = {
13832         headingOne = {\chapter{#1}},
13833         headingTwo = {\section{#1}},
13834         headingThree = {\subsection{#1}},
13835         headingFour = {\subsubsection{#1}},
13836         headingFive = {\paragraph{#1}},
13837         headingSix = {\subparagraph{#1}}}}
13838 }%

```

### 3.3.4.2 Tickboxes

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

```

13839 \markdownSetup{
13840   rendererPrototypes = {
13841     ulItem = {%
13842       \futurelet\markdownLaTeXCheckbox\markdownLaTeXULItem
13843     },
13844   },
13845 }
13846 \def\markdownLaTeXULItem{%
13847   \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
13848     \item[\markdownLaTeXCheckbox]%
13849     \expandafter\@gobble
13850   \else
13851     \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
13852       \item[\markdownLaTeXCheckbox]%
13853       \expandafter\expandafter\expandafter\@gobble
13854     \else
13855       \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
13856         \item[\markdownLaTeXCheckbox]%
13857         \expandafter\expandafter\expandafter\expandafter
13858         \expandafter\expandafter\expandafter\@gobble
13859       \else
13860         \item{}%
13861       \fi
13862     \fi
13863   \fi
13864 }

```

### 3.3.4.3 HTML elements

If the `html` option is enabled and we are using  $\text{T}_{\text{E}}\text{X}_{4\text{ht}}$ <sup>35</sup>, we will pass HTML elements to the output HTML document unchanged.

```

13865 \@ifundefined{HCode}{}{
13866   \markdownSetup{
13867     rendererPrototypes = {
13868       inlineHtmlTag = {%
13869         \ifvmode
13870           \IgnorePar
13871           \EndP
13872         \fi
13873         \HCode{#1}%
13874       },
13875       inputBlockHtmlElement = {%
13876         \ifvmode

```

---

<sup>35</sup>See <https://tug.org/tex4ht/>.

```

13877     \IgnorePar
13878     \fi
13879     \EndP
13880     \special{t4ht*<#1}%
13881     \par
13882     \ShowPar
13883   },
13884 },
13885 }
13886 }

```

### 3.3.4.4 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.

```

13887 \newcount\markdownLaTeXCitationsCounter
13888
13889 % Basic implementation
13890 \RequirePackage{gobble}
13891 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
13892   \advance\markdownLaTeXCitationsCounter by 1\relax
13893   \ifx\relax#4\relax
13894     \ifx\relax#5\relax
13895       \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
13896         \relax
13897         \cite{#1#2#6}% No prenotes/postnotes, just accumulate cites
13898         \expandafter\expandafter\expandafter
13899         \expandafter\expandafter\expandafter\expandafter
13900         \@gobblethree
13901       \fi
13902     \else% Before a postnote (#5), dump the accumulator
13903       \ifx\relax#1\relax\else
13904         \cite{#1}%
13905       \fi
13906       \cite[#5]{#6}%
13907     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
13908       \relax
13909     \else
13910       \expandafter\expandafter\expandafter
13911       \expandafter\expandafter\expandafter\expandafter
13912       \expandafter\expandafter\expandafter
13913       \expandafter\expandafter\expandafter\expandafter
13914       \markdownLaTeXBasicCitations
13915     \fi
13916     \expandafter\expandafter\expandafter

```

```

13917     \expandafter\expandafter\expandafter\expandafter{%
13918     \expandafter\expandafter\expandafter
13919     \expandafter\expandafter\expandafter\expandafter}%
13920     \expandafter\expandafter\expandafter
13921     \expandafter\expandafter\expandafter\expandafter{%
13922     \expandafter\expandafter\expandafter
13923     \expandafter\expandafter\expandafter\expandafter}%
13924     \expandafter\expandafter\expandafter
13925     \@gobblethree
13926     \fi
13927 \else% Before a prenote (#4), dump the accumulator
13928     \ifx\relax#1\relax\else
13929         \cite{#1}%
13930     \fi
13931     \ifnum\markdownLaTeXCitationsCounter>1\relax
13932         \space % Insert a space before the prenote in later citations
13933     \fi
13934     #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
13935     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
13936     \relax
13937     \else
13938         \expandafter\expandafter\expandafter
13939         \expandafter\expandafter\expandafter\expandafter
13940         \markdownLaTeXBasicCitations
13941     \fi
13942     \expandafter\expandafter\expandafter{%
13943     \expandafter\expandafter\expandafter}%
13944     \expandafter\expandafter\expandafter{%
13945     \expandafter\expandafter\expandafter}%
13946     \expandafter
13947     \@gobblethree
13948     \fi\markdownLaTeXBasicCitations{#1#2#6},}
13949 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
13950
13951 % Natbib implementation
13952 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
13953     \advance\markdownLaTeXCitationsCounter by 1\relax
13954     \ifx\relax#3\relax
13955         \ifx\relax#4\relax
13956             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
13957             \relax
13958             \citep{#1,#5}% No prenotes/postnotes, just accumulate cites
13959             \expandafter\expandafter\expandafter
13960             \expandafter\expandafter\expandafter\expandafter
13961             \@gobbletwo
13962         \fi
13963     \else% Before a postnote (#4), dump the accumulator

```

```

13964     \ifx\relax#1\relax\else
13965         \citep{#1}%
13966     \fi
13967     \citep[] [#4]{#5}%
13968     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
13969     \relax
13970     \else
13971         \expandafter\expandafter\expandafter
13972         \expandafter\expandafter\expandafter\expandafter
13973         \expandafter\expandafter\expandafter
13974         \expandafter\expandafter\expandafter\expandafter
13975         \markdownLaTeXNatbibCitations
13976     \fi
13977     \expandafter\expandafter\expandafter
13978     \expandafter\expandafter\expandafter\expandafter{%
13979     \expandafter\expandafter\expandafter
13980     \expandafter\expandafter\expandafter\expandafter}%
13981     \expandafter\expandafter\expandafter
13982     \@gobbletwo
13983 \fi
13984 \else% Before a prenote (#3), dump the accumulator
13985     \ifx\relax#1\relax\relax\else
13986         \citep{#1}%
13987     \fi
13988     \citep[#3] [#4]{#5}%
13989     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
13990     \relax
13991     \else
13992         \expandafter\expandafter\expandafter
13993         \expandafter\expandafter\expandafter\expandafter
13994         \markdownLaTeXNatbibCitations
13995     \fi
13996     \expandafter\expandafter\expandafter{%
13997     \expandafter\expandafter\expandafter}%
13998     \expandafter
13999     \@gobbletwo
14000 \fi\markdownLaTeXNatbibCitations{#1,#5}}
14001 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
14002 \advance\markdownLaTeXCitationsCounter by 1\relax
14003 \ifx\relax#3\relax
14004     \ifx\relax#4\relax
14005         \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
14006         \relax
14007         \citet{#1,#5}% No prenotes/postnotes, just accumulate cites
14008         \expandafter\expandafter\expandafter
14009         \expandafter\expandafter\expandafter\expandafter
14010         \@gobbletwo

```



```

14011     \fi
14012 \else% After a prenote or a postnote, dump the accumulator
14013     \ifx\relax#1\relax\else
14014         \citet{#1}%
14015     \fi
14016     , \citet[#3][#4]{#5}%
14017     \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
14018     \relax
14019     ,
14020 \else
14021     \ifnum
14022     \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
14023     \relax
14024     ,
14025     \fi
14026 \fi
14027 \expandafter\expandafter\expandafter
14028 \expandafter\expandafter\expandafter\expandafter
14029 \markdownLaTeXNatbibTextCitations
14030 \expandafter\expandafter\expandafter
14031 \expandafter\expandafter\expandafter\expandafter{%
14032 \expandafter\expandafter\expandafter
14033 \expandafter\expandafter\expandafter\expandafter}%
14034 \expandafter\expandafter\expandafter
14035 \@gobbletwo
14036 \fi
14037 \else% After a prenote or a postnote, dump the accumulator
14038     \ifx\relax#1\relax\relax\else
14039         \citet{#1}%
14040     \fi
14041     , \citet[#3][#4]{#5}%
14042     \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
14043     \relax
14044     ,
14045 \else
14046     \ifnum
14047     \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
14048     \relax
14049     ,
14050     \fi
14051 \fi
14052 \expandafter\expandafter\expandafter
14053 \markdownLaTeXNatbibTextCitations
14054 \expandafter\expandafter\expandafter{%
14055 \expandafter\expandafter\expandafter}%
14056 \expandafter
14057 \@gobbletwo

```

```

14058 \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
14059
14060 % BibLaTeX implementation
14061 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
14062 \advance\markdownLaTeXCitationsCounter by 1\relax
14063 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
14064 \relax
14065 \autocites#1[#3][#4]{#5}%
14066 \expandafter\@gobbletwo
14067 \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
14068 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
14069 \advance\markdownLaTeXCitationsCounter by 1\relax
14070 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
14071 \relax
14072 \textcites#1[#3][#4]{#5}%
14073 \expandafter\@gobbletwo
14074 \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}}
14075
14076 \markdownSetup{rendererPrototypes = {
14077 cite = {%
14078 \markdownLaTeXCitationsCounter=1%
14079 \def\markdownLaTeXCitationsTotal{#1}%
14080 \@ifundefined{autocites}{%
14081 \@ifundefined{citep}{%
14082 \expandafter\expandafter\expandafter
14083 \markdownLaTeXBasicCitations
14084 \expandafter\expandafter\expandafter{%
14085 \expandafter\expandafter\expandafter}%
14086 \expandafter\expandafter\expandafter{%
14087 \expandafter\expandafter\expandafter}%
14088 }{%
14089 \expandafter\expandafter\expandafter
14090 \markdownLaTeXNatbibCitations
14091 \expandafter\expandafter\expandafter{%
14092 \expandafter\expandafter\expandafter}%
14093 }%
14094 }{%
14095 \expandafter\expandafter\expandafter
14096 \markdownLaTeXBibLaTeXCitations
14097 \expandafter{\expandafter}%
14098 }},
14099 textCite = {%
14100 \markdownLaTeXCitationsCounter=1%
14101 \def\markdownLaTeXCitationsTotal{#1}%
14102 \@ifundefined{autocites}{%
14103 \@ifundefined{citep}{%
14104 \expandafter\expandafter\expandafter

```

```

14105     \markdownLaTeXBasicTextCitations
14106     \expandafter\expandafter\expandafter{%
14107     \expandafter\expandafter\expandafter}%
14108     \expandafter\expandafter\expandafter{%
14109     \expandafter\expandafter\expandafter}%
14110   }{%
14111     \expandafter\expandafter\expandafter
14112     \markdownLaTeXNatbibTextCitations
14113     \expandafter\expandafter\expandafter{%
14114     \expandafter\expandafter\expandafter}%
14115   }%
14116 }{%
14117   \expandafter\expandafter\expandafter
14118   \markdownLaTeXBibLaTeXTextCitations
14119   \expandafter{\expandafter}%
14120 }}}}
```

### 3.3.4.5 Links

Here is an implementation for hypertext links and relative references.

```

14121 \RequirePackage{url}
14122 \RequirePackage{expl3}
14123 \ExplSyntaxOn
14124 \def\markdownRendererLinkPrototype#1#2#3#4{
14125   \tl_set:Nn \l_tmpa_tl { #1 }
14126   \tl_set:Nn \l_tmpb_tl { #2 }
14127   \bool_set:Nn
14128     \l_tmpa_bool
14129     {
14130       \tl_if_eq_p:NN
14131         \l_tmpa_tl
14132         \l_tmpb_tl
14133     }
14134   \tl_set:Nn \l_tmpa_tl { #4 }
14135   \bool_set:Nn
14136     \l_tmpb_bool
14137     {
14138       \tl_if_empty_p:N
14139         \l_tmpa_tl
14140     }

```

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.

```

14141   \bool_if:nTF
14142     {
14143       \l_tmpa_bool && \l_tmpb_bool
14144     }

```

```

14145     {
14146     \markdownLaTeXRendererAutolink { #2 } { #3 }
14147     }{
14148     \markdownLaTeXRendererDirectOrIndirectLink
14149     { #1 } { #2 } { #3 } { #4 }
14150     }
14151 }
14152 \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.
14153 \tl_set:Nn
14154 \l_tmpa_tl
14155 { #2 }
14156 \tl_trim_spaces:N
14157 \l_tmpa_tl
14158 \tl_set:Nx
14159 \l_tmpb_tl
14160 {
14161 \tl_range:Nnn
14162 \l_tmpa_tl
14163 { 1 }
14164 { 1 }
14165 }
14166 \str_if_eq:NNTF
14167 \l_tmpb_tl
14168 \c_hash_str
14169 {
14170 \tl_set:Nx
14171 \l_tmpb_tl
14172 {
14173 \tl_range:Nnn
14174 \l_tmpa_tl
14175 { 2 }
14176 { -1 }
14177 }
14178 \exp_args:NV
14179 \ref
14180 \l_tmpb_tl
14181 }{
14182 \url { #2 }
14183 }
14184 }
14185 \ExplSyntaxOff
14186 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
14187 #1\footnote{\ifx\empty#4\empty\else#4: \fi\url{#3}}

```

### 3.3.4.6 Tables

Here is a basic implementation of tables. If the booktabs package is loaded, then it is used to produce horizontal lines.

```
14188 \newcount\markdownLaTeXRowCounter
14189 \newcount\markdownLaTeXRowTotal
14190 \newcount\markdownLaTeXColumnCounter
14191 \newcount\markdownLaTeXColumnTotal
14192 \newtoks\markdownLaTeXTable
14193 \newtoks\markdownLaTeXTableAlignment
14194 \newtoks\markdownLaTeXTableEnd
14195 \AtBeginDocument{%
14196   \@ifpackageloaded{booktabs}{%
14197     \def\markdownLaTeXTopRule{\toprule}%
14198     \def\markdownLaTeXMidRule{\midrule}%
14199     \def\markdownLaTeXBottomRule{\bottomrule}%
14200   }{%
14201     \def\markdownLaTeXTopRule{\hline}%
14202     \def\markdownLaTeXMidRule{\hline}%
14203     \def\markdownLaTeXBottomRule{\hline}%
14204   }%
14205 }
14206 \markdownSetup{rendererPrototypes={
14207   table = {%
14208     \markdownLaTeXTable={}%
14209     \markdownLaTeXTableAlignment={}%
14210     \markdownLaTeXTableEnd={%
14211       \markdownLaTeXBottomRule
14212       \end{tabular}}%
14213     \ifx\empty#1\empty\else
14214       \addto@hook\markdownLaTeXTable{%
14215         \begin{table}
14216         \centering}%
14217       \addto@hook\markdownLaTeXTableEnd{%
14218         \caption{#1}
14219         \end{table}}%
14220     \fi
14221     \addto@hook\markdownLaTeXTable{\begin{tabular}}%
14222     \markdownLaTeXRowCounter=0%
14223     \markdownLaTeXRowTotal=#2%
14224     \markdownLaTeXColumnTotal=#3%
14225     \markdownLaTeXRenderTableRow
14226   }
14227 }}
14228 \def\markdownLaTeXRenderTableRow#1{%
14229   \markdownLaTeXColumnCounter=0%
14230   \ifnum\markdownLaTeXRowCounter=0\relax
14231     \markdownLaTeXReadAlignments#1%
```

```

14232 \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
14233 \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
14234 \the\markdownLaTeXTableAlignment}}%
14235 \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
14236 \else
14237 \markdownLaTeXRenderTableCell#1%
14238 \fi
14239 \ifnum\markdownLaTeXRowCounter=1\relax
14240 \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
14241 \fi
14242 \advance\markdownLaTeXRowCounter by 1\relax
14243 \ifnum\markdownLaTeXRowCounter>\markdownLaTeXRowTotal\relax
14244 \the\markdownLaTeXTable
14245 \the\markdownLaTeXTableEnd
14246 \expandafter\@gobble
14247 \fi\markdownLaTeXRenderTableRow}
14248 \def\markdownLaTeXReadAlignments#1{%
14249 \advance\markdownLaTeXColumnCounter by 1\relax
14250 \if#1d%
14251 \addto@hook\markdownLaTeXTableAlignment{1}%
14252 \else
14253 \addto@hook\markdownLaTeXTableAlignment{#1}%
14254 \fi
14255 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
14256 \expandafter\@gobble
14257 \fi\markdownLaTeXReadAlignments}
14258 \def\markdownLaTeXRenderTableCell#1{%
14259 \advance\markdownLaTeXColumnCounter by 1\relax
14260 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
14261 \addto@hook\markdownLaTeXTable{#1&}%
14262 \else
14263 \addto@hook\markdownLaTeXTable{#1\\}%
14264 \expandafter\@gobble
14265 \fi\markdownLaTeXRenderTableCell}

```

### 3.3.4.7 Line Blocks

Here is a basic implementation of line blocks. If the `verse` package is loaded, then it is used to produce the verses.

```

14266
14267 \markdownIfOption{lineBlocks}{%
14268 \RequirePackage{verse}
14269 \markdownSetup{rendererPrototypes={
14270 lineBlockBegin = {%
14271 \begingroup
14272 \def\markdownRendererHardLineBreak{\}%
14273 \begin{verse}%

```

```

14274     },
14275     lineBlockEnd = {%
14276         \end{verse}%
14277     \endgroup
14278     },
14279 }}
14280 }{}
14281

```

### 3.3.4.8 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.

```

14282 \ExplSyntaxOn
14283 \keys_define:nn
14284 { markdown/jekyllData }
14285 {
14286     author .code:n = { \author{#1} },
14287     date   .code:n = { \date{#1}   },
14288     title  .code:n = { \title{#1}  },
14289 }

```

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.

```

14290 \markdownSetup{
14291     rendererPrototypes = {
14292         jekyllDataEnd = {
14293             \AddToHook{begindocument/end}{\maketitle}
14294         },
14295     },
14296 }
14297 \ExplSyntaxOff

```

### 3.3.4.9 Strike-Through

If the `strikeThrough` option is enabled, we will load the `soulutf8` package and use it to implement strike-throughs.

```

14298 \markdownIfOption{strikeThrough}{%
14299     \RequirePackage{soulutf8}%
14300     \markdownSetup{
14301         rendererPrototypes = {
14302             strikeThrough = {%
14303                 \st{#1}%
14304             },

```

```

14305     }
14306   }
14307 }{}

```

### 3.3.4.10 Marked Text

If the `mark` option is enabled, we will load the `soulutf8` package and use it to implement marked text.

```

14308 \markdownIfOption{mark}{%
14309   \RequirePackage{soulutf8}%
14310   \markdownSetup{
14311     rendererPrototypes = {
14312       mark = {%
14313         \hl{#1}%
14314       },
14315     }
14316   }
14317 }{}

```

### 3.3.4.11 Image Attributes

If the `linkAttributes` option is enabled, we will load the `graphicx` package. Furthermore, in image attribute contexts, we will make attributes in the form  $\langle key \rangle = \langle value \rangle$  set the corresponding keys of the `graphicx` package to the corresponding values.

```

14318 \ExplSyntaxOn
14319 \@@_if_option:nT
14320 { linkAttributes }
14321 {
14322   \RequirePackage{graphicx}
14323   \markdownSetup{
14324     rendererPrototypes = {
14325       imageAttributeContextBegin = {
14326         \group_begin:
14327         \markdownSetup{
14328           rendererPrototypes = {
14329             attributeKeyValue = {
14330               \setkeys
14331                 { Gin }
14332                 { { ##1 } = { ##2 } }
14333             },
14334           },
14335         }
14336       },
14337       imageAttributeContextEnd = {
14338         \group_end:
14339       },

```



```

14340     },
14341     }
14342   }
14343 \ExplSyntaxOff

```

### 3.3.4.12 Raw Attributes

In the raw block and inline raw span renderer prototypes, default to the plain TeX renderer prototypes, translating raw attribute `latex` to `tex`.

```

14344 \ExplSyntaxOn
14345 \cs_gset:Npn
14346   \markdownRendererInputRawInlinePrototype#1#2
14347   {
14348     \str_case:nnF
14349       { #2 }
14350       {
14351         { latex }
14352         {
14353           \@@_plain_tex_default_input_raw_inline:nn
14354             { #1 }
14355             { tex }
14356         }
14357       }
14358     {
14359       \@@_plain_tex_default_input_raw_inline:nn
14360         { #1 }
14361         { #2 }
14362     }
14363   }
14364 \cs_gset:Npn
14365   \markdownRendererInputRawBlockPrototype#1#2
14366   {
14367     \str_case:nnF
14368       { #2 }
14369       {
14370         { latex }
14371         {
14372           \@@_plain_tex_default_input_raw_block:nn
14373             { #1 }
14374             { tex }
14375         }
14376       }
14377     {
14378       \@@_plain_tex_default_input_raw_block:nn
14379         { #1 }
14380         { #2 }
14381     }

```

```

14382 }
14383 \ExplSyntaxOff
14384 \fi % Closes \markdownIfOption{plain}{\iffalse}{\iftrue}

```

### 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.

```

14385 \newcommand\markdownMakeOther{%
14386   \count0=128\relax
14387   \loop
14388     \catcode\count0=11\relax
14389     \advance\count0 by 1\relax
14390   \ifnum\count0<256\repeat}%

```

## 3.4 ConTeXt Implementation

The ConTeXt implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV ConTeXt formats *seem* to implement (the documentation is scarce) the majority of the plain TeX format required by the plain TeX implementation. As a consequence, we can directly reuse the existing plain TeX implementation after supplying the missing plain TeX macros.

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.

```

14391 \def\markdownMakeOther{%
14392   \count0=128\relax
14393   \loop
14394     \catcode\count0=11\relax
14395     \advance\count0 by 1\relax
14396   \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 ConTeXt.

```

14397   \catcode`|=12}%

```

### 3.4.1 Typesetting Markdown

The `\inputmarkdown` macro is defined to accept an optional argument with options recognized by the ConTeXt interface (see Section 2.4.2).

```

14398 \long\def\inputmarkdown{%
14399   \dosingleempty

```

```

14400 \doinputmarkdown}%
14401 \long\def\doinputmarkdown[#1]#2{%
14402 \begingroup
14403 \iffirstargument
14404 \setupmarkdown[#1]%
14405 \fi
14406 \markdownInput{#2}%
14407 \endgroup}%

```

The `\startmarkdown` and `\stopmarkdown` macros are implemented using the `\markdownReadAndConvert` macro.

In Knuth's  $\TeX$ , trailing spaces are removed very early on when a line is being put to the input buffer. [13, sec. 31]. According to Eijkhout [14, sec. 2.2], this is because “these spaces are hard to see in an editor”. At the moment, there is no option to suppress this behavior in (Lua) $\TeX$ , but Con $\TeX$ t MkIV funnels all input through its own input handler. This makes it possible to suppress the removal of trailing spaces in Con $\TeX$ t MkIV and therefore to insert hard line breaks into markdown text.

```

14408 \startluacode
14409 document.markdown_buffering = false
14410 local function preserve_trailing_spaces(line)
14411   if document.markdown_buffering then
14412     line = line:gsub("[ \t][ \t]$", "\t\t")
14413   end
14414   return line
14415 end
14416 resolvers.installinputlinehandler(preserve_trailing_spaces)
14417 \stopluacode
14418 \begingroup
14419 \catcode\|=0%
14420 \catcode\|=12%
14421 |gdef|startmarkdown{%
14422 |ctxlua{document.markdown_buffering = true}%
14423 |markdownReadAndConvert{\stopmarkdown}%
14424 |stopmarkdown}%
14425 |gdef|stopmarkdown{%
14426 |ctxlua{document.markdown_buffering = false}%
14427 |markdownEnd}%
14428 |endgroup

```

### 3.4.2 Themes

This section overrides the plain  $\TeX$  implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in Con $\TeX$ t themes provided with the Markdown package.

```

14429 \ExplSyntaxOn
14430 \cs_gset:Nn

```

```

14431 \@@_load_theme:nn
14432 {
    Determine whether a file named t-markdowntheme<munged theme name>.tex
    exists. If it does, load it. Otherwise, try loading a plain TEX theme instead.
14433 \file_if_exist:nTF
14434 { t - markdown theme #2.tex }
14435 {
14436 \msg_info:nnn
14437 { markdown }
14438 { loading-context-theme }
14439 { #1 }
14440 \usemodule
14441 [ t ]
14442 [ markdown theme #2 ]
14443 }
14444 {
14445 \@@_plain_tex_load_theme:nn
14446 { #1 }
14447 { #2 }
14448 }
14449 }
14450 \msg_new:nnn
14451 { markdown }
14452 { loading-context-theme }
14453 { Loading~ConTeXt~Markdown~theme~#1 }
14454 \ExplSyntaxOff

```

The `witiko/markdown/defaults` ConT<sub>E</sub>Xt theme provides default definitions for token renderer prototypes. First, the ConT<sub>E</sub>Xt theme loads the plain T<sub>E</sub>X theme with the default definitions for plain T<sub>E</sub>X:

```
14455 \markdownLoadPlainTeXTheme
```

Next, the ConT<sub>E</sub>Xt theme overrides some of the plain T<sub>E</sub>X definitions. See Section 3.4.3 for the actual definitions.

### 3.4.3 Token Renderer Prototypes

The following configuration should be considered placeholder. If the option `plain` has been enabled (see Section 2.2.2.3), none of the definitions will take effect.

```

14456 \markdownIfOption{plain}{\iffalse}{\iftrue}
14457 \def\markdownRendererHardLineBreakPrototype{\blank}%
14458 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
14459 \def\markdownRendererRightBracePrototype{\textbraceright}%
14460 \def\markdownRendererDollarSignPrototype{\textdollar}%
14461 \def\markdownRendererPercentSignPrototype{\percent}%
14462 \def\markdownRendererUnderscorePrototype{\textunderscore}%
14463 \def\markdownRendererCircumflexPrototype{\textcircumflex}%

```

```

14464 \def\markdownRendererBackslashPrototype{\textbackslash}%
14465 \def\markdownRendererTildePrototype{\textasciitilde}%
14466 \def\markdownRendererPipePrototype{\char`|}%
14467 \def\markdownRendererLinkPrototype#1#2#3#4{%
14468   \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
14469   \fi\texttt<\hyphenatedurl{#3}>}}%
14470 \usemodule[database]
14471 \defineseparatedlist
14472   [MarkdownConTeXtCSV]
14473   [separator={,},
14474   before=\bTABLE,after=\eTABLE,
14475   first=\bTR,last=\eTR,
14476   left=\bTD,right=\eTD]
14477 \def\markdownConTeXtCSV{csv}
14478 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
14479   \def\markdownConTeXtCSV@arg{#1}%
14480   \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
14481     \placetable[] [tab:#1]{#4}{%
14482       \processseparatedfile [MarkdownConTeXtCSV] [#3]}%
14483   \else
14484     \markdownInput{#3}%
14485   \fi}%
14486 \def\markdownRendererImagePrototype#1#2#3#4{%
14487   \placefigure[] []{#4}{\externalfigure[#3]}%
14488 \def\markdownRendererULBeginPrototype{\startitemize}%
14489 \def\markdownRendererULBeginTightPrototype{\startitemize[packed]}%
14490 \def\markdownRendererULItemPrototype{\item}%
14491 \def\markdownRendererULEndPrototype{\stopitemize}%
14492 \def\markdownRendererULEndTightPrototype{\stopitemize}%
14493 \def\markdownRendererOLBeginPrototype{\startitemize[n]}%
14494 \def\markdownRendererOLBeginTightPrototype{\startitemize[packed,n]}%
14495 \def\markdownRendererOLItemPrototype{\item}%
14496 \def\markdownRendererOLItemWithNumberPrototype#1{\sym{#1.}}%
14497 \def\markdownRendererOLEndPrototype{\stopitemize}%
14498 \def\markdownRendererOLEndTightPrototype{\stopitemize}%
14499 \definedescription
14500   [MarkdownConTeXtDlItemPrototype]
14501   [location=hanging,
14502   margin=standard,
14503   headstyle=bold]%
14504 \definestartstop
14505   [MarkdownConTeXtDlPrototype]
14506   [before=\blank,
14507   after=\blank]%
14508 \definestartstop
14509   [MarkdownConTeXtDlTightPrototype]
14510   [before=\blank\startpacked,

```

```

14511   after=\stoppacked\blank]%
14512 \def\markdownRendererDlBeginPrototype{%
14513   \startMarkdownConTeXtDlPrototype}%
14514 \def\markdownRendererDlBeginTightPrototype{%
14515   \startMarkdownConTeXtDlTightPrototype}%
14516 \def\markdownRendererDlItemPrototype#1{%
14517   \startMarkdownConTeXtDlItemPrototype{#1}}%
14518 \def\markdownRendererDlItemEndPrototype{%
14519   \stopMarkdownConTeXtDlItemPrototype}%
14520 \def\markdownRendererDlEndPrototype{%
14521   \stopMarkdownConTeXtDlPrototype}%
14522 \def\markdownRendererDlEndTightPrototype{%
14523   \stopMarkdownConTeXtDlTightPrototype}%
14524 \def\markdownRendererEmphasisPrototype#1{\em#1}%
14525 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
14526 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
14527 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
14528 \def\markdownRendererLineBlockBeginPrototype{%
14529   \begingroup
14530     \def\markdownRendererHardLineBreak{
14531       }%
14532     \startlines
14533   }%
14534 \def\markdownRendererLineBlockEndPrototype{%
14535     \stoplines
14536   \endgroup
14537 }%
14538 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%

```

### 3.4.3.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

14539 \ExplSyntaxOn
14540 \cs_gset:Npn
14541   \markdownRendererInputFencedCodePrototype#1#2#3
14542   {
14543     \tl_if_empty:nTF
14544       { #2 }
14545       { \markdownRendererInputVerbatim{#1} }

```

Otherwise, extract the first word of the infostring and treat it as the name of the programming language in which the code block is written. This name is then used in the ConTeXt `\definetying` macro, which allows the user to set up code highlighting mapping as follows:

```

\definetying [latex]
\setuptying  [latex] [option=TEX]

```

```

\starttext
  \startmarkdown
~~~ latex
\documentclass{article}
\begin{document}
  Hello world!
\end{document}
~~~
  \stopmarkdown
\stoptext

```

```

14546     {
14547         \regex_extract_once:nnN
14548         { \w* }
14549         { #2 }
14550         \l_tmpa_seq
14551         \seq_pop_left:NN
14552         \l_tmpa_seq
14553         \l_tmpa_tl
14554         \typefile[\l_tmpa_tl] []{#1}
14555     }
14556 }
14557 \ExplSyntaxOff
14558 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
14559 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
14560 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
14561 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
14562 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
14563 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
14564 \def\markdownRendererThematicBreakPrototype{%
14565     \blackrule[height=1pt, width=\hsize]}%
14566 \def\markdownRendererNotePrototype#1{\footnote{#1}}%
14567 \def\markdownRendererTickedBoxPrototype{${\boxtimes}$}
14568 \def\markdownRendererHalfTickedBoxPrototype{${\boxdot}$}
14569 \def\markdownRendererUntickedBoxPrototype{${\square}$}
14570 \def\markdownRendererStrikeThroughPrototype#1{\overstrikes{#1}}
14571 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
14572 \def\markdownRendererSubscriptPrototype#1{\low{#1}}
14573 \def\markdownRendererDisplayMathPrototype#1{%
14574     \startformula#1\stopformula}%

```

### 3.4.3.2 Tables

There is a basic implementation of tables.

```

14575 \newcount\markdownConTeXtRowCounter

```

```

14576 \newcount\markdownConTeXtRowTotal
14577 \newcount\markdownConTeXtColumnCounter
14578 \newcount\markdownConTeXtColumnTotal
14579 \newtoks\markdownConTeXtTable
14580 \newtoks\markdownConTeXtTableFloat
14581 \def\markdownRendererTablePrototype#1#2#3{%
14582   \markdownConTeXtTable={}%
14583   \ifx\empty#1\empty
14584     \markdownConTeXtTableFloat={%
14585       \the\markdownConTeXtTable}%
14586   \else
14587     \markdownConTeXtTableFloat={%
14588       \placetable{#1}{\the\markdownConTeXtTable}}%
14589   \fi
14590   \begingroup
14591   \setupTABLE[r][each][topframe=off,bottomframe=off,
14592     leftframe=off,rightframe=off]
14593   \setupTABLE[c][each][topframe=off,bottomframe=off,
14594     leftframe=off,rightframe=off]
14595   \setupTABLE[r][1][topframe=on,bottomframe=on]
14596   \setupTABLE[r][#1][bottomframe=on]
14597   \markdownConTeXtRowCounter=0%
14598   \markdownConTeXtRowTotal=#2%
14599   \markdownConTeXtColumnTotal=#3%
14600   \markdownConTeXtRenderTableRow}
14601 \def\markdownConTeXtRenderTableRow#1{%
14602   \markdownConTeXtColumnCounter=0%
14603   \ifnum\markdownConTeXtRowCounter=0\relax
14604     \markdownConTeXtReadAlignments#1%
14605     \markdownConTeXtTable={\bTABLE}%
14606   \else
14607     \markdownConTeXtTable=\expandafter{%
14608       \the\markdownConTeXtTable\bTR}%
14609     \markdownConTeXtRenderTableCell#1%
14610     \markdownConTeXtTable=\expandafter{%
14611       \the\markdownConTeXtTable\eTR}%
14612   \fi
14613   \advance\markdownConTeXtRowCounter by 1\relax
14614   \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
14615     \markdownConTeXtTable=\expandafter{%
14616       \the\markdownConTeXtTable\eTABLE}%
14617     \the\markdownConTeXtTableFloat
14618   \endgroup
14619   \expandafter\gobbleoneargument
14620   \fi\markdownConTeXtRenderTableRow}
14621 \def\markdownConTeXtReadAlignments#1{%
14622   \advance\markdownConTeXtColumnCounter by 1\relax

```



```

14623 \if#1d%
14624 \setupTABLE[c] [\the\markdownConTeXtColumnCounter] [align=right]
14625 \fi\if#1l%
14626 \setupTABLE[c] [\the\markdownConTeXtColumnCounter] [align=right]
14627 \fi\if#1c%
14628 \setupTABLE[c] [\the\markdownConTeXtColumnCounter] [align=middle]
14629 \fi\if#1r%
14630 \setupTABLE[c] [\the\markdownConTeXtColumnCounter] [align=left]
14631 \fi
14632 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
14633 \else
14634 \expandafter\gobbleoneargument
14635 \fi\markdownConTeXtReadAlignments}
14636 \def\markdownConTeXtRenderTableCell#1{%
14637 \advance\markdownConTeXtColumnCounter by 1\relax
14638 \markdownConTeXtTable=\expandafter{%
14639 \the\markdownConTeXtTable\bTD#1\eTD}%
14640 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
14641 \else
14642 \expandafter\gobbleoneargument
14643 \fi\markdownConTeXtRenderTableCell}

```

### 3.4.3.3 Raw Attributes

In the raw block and inline raw span renderer prototypes, default to the plain TeX renderer prototypes, translating raw attribute `context` to `tex`.

```

14644 \ExplSyntaxOn
14645 \cs_gset:Npn
14646 \markdownRendererInputRawInlinePrototype#1#2
14647 {
14648 \str_case:nnF
14649 { #2 }
14650 {
14651 { latex }
14652 {
14653 \@@_plain_tex_default_input_raw_inline:nn
14654 { #1 }
14655 { context }
14656 }
14657 }
14658 {
14659 \@@_plain_tex_default_input_raw_inline:nn
14660 { #1 }
14661 { #2 }
14662 }
14663 }
14664 \cs_gset:Npn

```

```

14665 \markdownRendererInputRawBlockPrototype#1#2
14666 {
14667   \str_case:nnF
14668     { #2 }
14669     {
14670       { context }
14671       {
14672         \@_plain_tex_default_input_raw_block:nn
14673           { #1 }
14674           { tex }
14675         }
14676       }
14677     {
14678       \@_plain_tex_default_input_raw_block:nn
14679         { #1 }
14680         { #2 }
14681       }
14682     }
14683 \cs_gset_eq:NN
14684   \markdownRendererInputRawBlockPrototype
14685   \markdownRendererInputRawInlinePrototype
14686 \fi % Closes \markdownIfOption{plain}{\iffalse}{\iftrue}`
14687 \ExplSyntaxOff
14688 \stopmodule
14689 \protect

```

At the end of the ConT<sub>E</sub>Xt module, we load the `witiko/markdown/defaults` ConT<sub>E</sub>Xt theme with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```

14690 \markdownIfOption{noDefaults}{}{
14691   \setupmarkdown[theme=witiko/markdown/defaults]
14692 }
14693 \stopmodule
14694 \protect

```

## References

- [1] LuaT<sub>E</sub>X development team. *LuaT<sub>E</sub>X reference manual*. Version 1.10 (stable). July 23, 2021. URL: <https://www.pragma-ade.com/general/manuals/luatex.pdf> (visited on 09/30/2022).
- [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] John MacFarlane. *Pandoc. a universal document converter*. 2022. URL: <https://pandoc.org/> (visited on 10/05/2022).
- [5] Bonita Sharif and Jonathan I. Maletic. “An Eye Tracking Study on camelCase and under\_score Identifier Styles.” In: *2010 IEEE 18th International Conference on Program Comprehension*. 2010, pp. 196–205. DOI: [10.1109/ICPC.2010.41](https://doi.org/10.1109/ICPC.2010.41).
- [6] 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.
- [7] 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).
- [8] 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).
- [9] Vít Starý Novotný et al. *Convert control sequence with a variable number of undelimited parameters into a token list*. URL: <https://tex.stackexchange.com/q/716362/70941> (visited on 04/28/2024).
- [10] 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).
- [11] Roberto Ierusalimsky. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.
- [12] 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).
- [13] Donald Ervin Knuth. *T<sub>E</sub>X: The Program*. Vol. B. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. xvi, 594. ISBN: 978-0-201-13437-7.
- [14] 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: 978-0-201-56882-0.

## Index

|                                    |                |
|------------------------------------|----------------|
| <code>autoIdentifiers</code>       | 21, 33, 75, 89 |
| <code>blankBeforeBlockquote</code> | 21             |
| <code>blankBeforeCodeFence</code>  | 21             |
| <code>blankBeforeDivFence</code>   | 22             |

|                                   |  |
|-----------------------------------|--|
| blankBeforeHeading                | 22   |
| blankBeforeList                   | 22   |
| bracketedSpans                    | 23, 77                                     |
| breakableBlockquotes              | 23   |
| cacheDir                          | 4, 17, 19, 56, 57, 137, 150, 350, 370, 386 |
| citationNbsps                     | 23   |
| citations                         | 24, 80                                     |
| codeSpans                         | 24   |
| contentBlocks                     | 20, 25                                     |
| contentBlocksLanguageMap          | 20   |
| contentLevel                      | 25   |
| debugExtensions                   | 9, 20, 26, 294                             |
| debugExtensionsFileName           | 20, 26                                     |
| defaultOptions                    | 10, 50, 348, 350                           |
| definitionLists                   | 26, 84                                     |
| eagerCache                        | 17, 348                                    |
| ensureJekyllData                  | 27   |
| entities.char_entity              | 196  |
| entities.dec_entity               | 196  |
| entities.hex_entity               | 196  |
| entities.hex_entity_with_x_char   | 196  |
| escape_minimal                    | 200  |
| escape_programmatic_text          | 200  |
| escape_typographic_text           | 200  |
| expandtabs                        | 254  |
| expectJekyllData                  | 27, 27                                     |
| extensions                        | 28, 145, 299                               |
| extensions.bracketed_spans        | 299  |
| extensions.citations              | 300  |
| extensions.content_blocks         | 304  |
| extensions.definition_lists       | 308  |
| extensions.fancy_lists            | 310  |
| extensions.fenced_code            | 316  |
| extensions.fenced_divs            | 321  |
| extensions.header_attributes      | 325  |
| extensions.inline_code_attributes | 327  |
| extensions.jekyll_data            | 344  |
| extensions.line_blocks            | 327  |
| extensions.link_attributes        | 329  |
| extensions.mark                   | 328  |

|   |   |
|---|---|
| extensions.notes                              | 331   |
| extensions.pipe_table                         | 333   |
| extensions.raw_inline                         | 337   |
| extensions.strike_through                     | 338   |
| extensions.subscripts                         | 339   |
| extensions.superscripts                       | 339   |
| extensions.tex_math                           | 340   |
| fancyLists                                    | 30, 98–103, 387                                 |
| fencedCode                                    | 30, 38, 81, 87, 103, 384                        |
| fencedCodeAttributes                          | 31, 75, 87                                      |
| fencedDiv                                     | 88  |
| fencedDivs                                    | 31, 40  |
| finalizeCache                                 | 17, 20, 32, 32, 56, 57, 137, 348, 349           |
| frozenCache                                   | 20, 32, 57, 137, 139, 140, 384, 386             |
| frozenCacheCounter                            | 32, 349, 377, 378                               |
| frozenCacheFileName                           | 20, 32, 56, 349                                 |
| gfmAutoIdentifiers                            | 21, 32, 75, 89                                  |
| hashEnumerators                               | 33  |
| headerAttributes                              | 33, 40, 75, 89                                  |
| html  | 34, 91, 92, 397                                 |
| hybrid  | 34, 39, 45, 47, 59, 68, 105, 137, 201, 255, 377 |
| inlineCodeAttributes                          | 34, 75, 82                                      |
| inlineNotes                                   | 35  |
| \input  | 54  |
| \inputmarkdown                                | 142, 143, 410                                   |
| inputTempFileName                             | 57, 59, 371, 372, 374, 375                      |
| iterlines                                     | 254   |
| jeekyllData                                   | 3, 27, 28, 35, 112–116                          |
| \l_file_search_path_seq                       | 376   |
| languages_json                                | 305, 305  |
| lineBlocks                                    | 36, 94  |
| linkAttributes                                | 36, 75, 93, 96, 275, 408                        |
| mark  | 37, 96, 408                                     |
| \markdown                                     | 135, 135, 136, 381                              |
| markdown                                      | 134, 134, 135, 379, 380                         |
| markdown*                                     | 134, 134–136, 379                               |
| \markdown_jeekyll_data_concatenate_address:NN | 365   |

|  |                                 |
|--|---------------------------------|
| <code>\markdown_jekyll_data_pop:</code>                          | 366                             |
| <code>\markdown_jekyll_data_push:nN</code>                       | 366                             |
| <code>\markdown_jekyll_data_push_address_segment:n</code>        | 364                             |
| <code>\markdown_jekyll_data_set_keyval:Nn</code>                 | 366                             |
| <code>\markdown_jekyll_data_set_keyvals:nn</code>                | 366                             |
| <code>\markdown_jekyll_data_update_address_tls:</code>           | 365                             |
| <code>\markdownBegin</code>                                      | 52, 52–54, 133, 135, 142        |
| <code>\markdownCleanup</code>                                    | 370                             |
| <code>\markdownEnd</code>  | 52, 52–54, 133, 135, 142        |
| <code>\markdownError</code>                                      | 132, 132                        |
| <code>\markdownEscape</code>                                     | 52, 54, 378                     |
| <code>\markdownIfOption</code>                                   | 55                              |
| <code>\markdownIfSnippetExists</code>                            | 70                              |
| <code>\markdownInfo</code>                                       | 132, 132                        |
| <code>\markdownInput</code>                                      | 52, 54, 134, 136, 143, 376, 379 |
| <code>\markdownInputFileStream</code>                            | 370                             |
| <code>\markdownInputPlainTeX</code>                              | 379                             |
| <code>\markdownLoadPlainTeXTheme</code>                          | 138, 144, 359                   |
| <code>\markdownLuaExecute</code>                                 | 373, 376                        |
| <code>\markdownLuaOptions</code>                                 | 368, 370                        |
| <code>\markdownMakeOther</code>                                  | 133, 410                        |
| <code>\markdownOptionFinalizeCache</code>                        | 56                              |
| <code>\markdownOptionFrozenCache</code>                          | 56                              |
| <code>\markdownOptionHybrid</code>                               | 59                              |
| <code>\markdownOptionInputTempFileName</code>                    | 57                              |
| <code>\markdownOptionNoDefaults</code>                           | 58                              |
| <code>\markdownOptionOutputDir</code>                            | 57, 57, 60                      |
| <code>\markdownOptionPlain</code>                                | 58                              |
| <code>\markdownOptionStripPercentSigns</code>                    | 59                              |
| <code>\markdownOutputFileStream</code>                           | 370                             |
| <code>\markdownPrepare</code>                                    | 370                             |
| <code>\markdownPrepareLuaOptions</code>                          | 368                             |
| <code>\markdownReadAndConvert</code>                             | 133, 371, 379–381, 411          |
| <code>\markdownReadAndConvertProcessLine</code>                  | 372, 373                        |
| <code>\markdownReadAndConvertStripPercentSigns</code>            | 371                             |
| <code>\markdownReadAndConvertTab</code>                          | 370                             |
| <code>\markdownRendererAttributeClassName</code>                 | 76                              |
| <code>\markdownRendererAttributeIdentifier</code>                | 76                              |
| <code>\markdownRendererAttributeKeyValue</code>                  | 76                              |
| <code>\markdownRendererBlockQuoteBegin</code>                    | 77                              |
| <code>\markdownRendererBlockQuoteEnd</code>                      | 77                              |
| <code>\markdownRendererBracketedSpanAttributeContextBegin</code> | 77                              |

|  |         |
|--|---------|
| <code>\markdownRendererBracketedSpanAttributeContextEnd</code> | 77      |
| <code>\markdownRendererCite</code>                             | 80, 80  |
| <code>\markdownRendererCodeSpan</code>                         | 81      |
| <code>\markdownRendererCodeSpanAttributeContextBegin</code>    | 82      |
| <code>\markdownRendererCodeSpanAttributeContextEnd</code>      | 82      |
| <code>\markdownRendererContentBlock</code>                     | 82, 83  |
| <code>\markdownRendererContentBlockCode</code>                 | 83      |
| <code>\markdownRendererContentBlockOnlineImage</code>          | 83      |
| <code>\markdownRendererDisplayMath</code>                      | 110     |
| <code>\markdownRendererDlBegin</code>                          | 84      |
| <code>\markdownRendererDlBeginTight</code>                     | 84      |
| <code>\markdownRendererDlDefinitionBegin</code>                | 85      |
| <code>\markdownRendererDlDefinitionEnd</code>                  | 85      |
| <code>\markdownRendererDlEnd</code>                            | 86      |
| <code>\markdownRendererDlEndTight</code>                       | 86      |
| <code>\markdownRendererDlItem</code>                           | 84      |
| <code>\markdownRendererDlItemEnd</code>                        | 85      |
| <code>\markdownRendererDocumentBegin</code>                    | 97      |
| <code>\markdownRendererDocumentEnd</code>                      | 97      |
| <code>\markdownRendererEllipsis</code>                         | 41, 86  |
| <code>\markdownRendererEmphasis</code>                         | 87, 120 |
| <code>\markdownRendererError</code>                            | 111     |
| <code>\markdownRendererFancyOlBegin</code>                     | 99, 99  |
| <code>\markdownRendererFancyOlBeginTight</code>                | 99      |
| <code>\markdownRendererFancyOlEnd</code>                       | 102     |
| <code>\markdownRendererFancyOlEndTight</code>                  | 102     |
| <code>\markdownRendererFancyOlItem</code>                      | 100     |
| <code>\markdownRendererFancyOlItemEnd</code>                   | 101     |
| <code>\markdownRendererFancyOlItemWithNumber</code>            | 101     |
| <code>\markdownRendererFencedCodeAttributeContextBegin</code>  | 87      |
| <code>\markdownRendererFencedCodeAttributeContextEnd</code>    | 87      |
| <code>\markdownRendererFencedDivAttributeContextBegin</code>   | 88      |
| <code>\markdownRendererFencedDivAttributeContextEnd</code>     | 88      |
| <code>\markdownRendererHalfTickedBox</code>                    | 111     |
| <code>\markdownRendererHardLineBreak</code>                    | 95      |
| <code>\markdownRendererHeaderAttributeContextBegin</code>      | 89      |
| <code>\markdownRendererHeaderAttributeContextEnd</code>        | 89      |
| <code>\markdownRendererHeadingFive</code>                      | 90      |
| <code>\markdownRendererHeadingFour</code>                      | 90      |
| <code>\markdownRendererHeadingOne</code>                       | 89      |
| <code>\markdownRendererHeadingSix</code>                       | 91      |
| <code>\markdownRendererHeadingThree</code>                     | 90      |

|  |               |
|--|---------------|
| <code>\markdownRendererHeadingTwo</code>                   | 90            |
| <code>\markdownRendererImage</code>                        | 92            |
| <code>\markdownRendererImageAttributeContextBegin</code>   | 93            |
| <code>\markdownRendererImageAttributeContextEnd</code>     | 93            |
| <code>\markdownRendererInlineHtmlComment</code>            | 91            |
| <code>\markdownRendererInlineHtmlTag</code>                | 91            |
| <code>\markdownRendererInlineMath</code>                   | 110           |
| <code>\markdownRendererInputBlockHtmlElement</code>        | 92            |
| <code>\markdownRendererInputFencedCode</code>              | 81            |
| <code>\markdownRendererInputRawBlock</code>                | 103           |
| <code>\markdownRendererInputRawInline</code>               | 103           |
| <code>\markdownRendererInputVerbatim</code>                | 81            |
| <code>\markdownRendererInterblockSeparator</code>          | 93            |
| <code>\markdownRendererJekyllDataBegin</code>              | 112           |
| <code>\markdownRendererJekyllDataBoolean</code>            | 114           |
| <code>\markdownRendererJekyllDataEmpty</code>              | 116           |
| <code>\markdownRendererJekyllDataEnd</code>                | 112           |
| <code>\markdownRendererJekyllDataMappingBegin</code>       | 113           |
| <code>\markdownRendererJekyllDataMappingEnd</code>         | 113           |
| <code>\markdownRendererJekyllDataNumber</code>             | 114           |
| <code>\markdownRendererJekyllDataProgrammaticString</code> | 115, 115      |
| <code>\markdownRendererJekyllDataSequenceBegin</code>      | 113           |
| <code>\markdownRendererJekyllDataSequenceEnd</code>        | 114           |
| <code>\markdownRendererJekyllDataString</code>             | 115–117, 127  |
| <code>\markdownRendererJekyllDataTypographicString</code>  | 115, 115, 344 |
| <code>\markdownRendererLineBlockBegin</code>               | 94            |
| <code>\markdownRendererLineBlockEnd</code>                 | 94            |
| <code>\markdownRendererLink</code>                         | 95, 120       |
| <code>\markdownRendererLinkAttributeContextBegin</code>    | 96            |
| <code>\markdownRendererLinkAttributeContextEnd</code>      | 96            |
| <code>\markdownRendererMark</code>                         | 96            |
| <code>\markdownRendererNbsp</code>                         | 97            |
| <code>\markdownRendererNote</code>                         | 98            |
| <code>\markdownRendererOlBegin</code>                      | 98            |
| <code>\markdownRendererOlBeginTight</code>                 | 98            |
| <code>\markdownRendererOlEnd</code>                        | 101           |
| <code>\markdownRendererOlEndTight</code>                   | 102           |
| <code>\markdownRendererOlItem</code>                       | 41, 99        |
| <code>\markdownRendererOlItemEnd</code>                    | 100           |
| <code>\markdownRendererOlItemWithNumber</code>             | 41, 100       |
| <code>\markdownRendererParagraphSeparator</code>           | 94            |
| <code>\markdownRendererReplacementCharacter</code>         | 104           |



|  |                                |
|--|--------------------------------|
| <code>\markdownRendererSectionBegin</code>               | 104                            |
| <code>\markdownRendererSectionEnd</code>                 | 104                            |
| <code>\markdownRendererSoftLineBreak</code>              | 95                             |
| <code>\markdownRendererStrikeThrough</code>              | 107                            |
| <code>\markdownRendererStrongEmphasis</code>             | 87                             |
| <code>\markdownRendererSubscript</code>                  | 108                            |
| <code>\markdownRendererSuperscript</code>                | 108                            |
| <code>\markdownRendererTable</code>                      | 109                            |
| <code>\markdownRendererTableAttributeContextBegin</code> | 108                            |
| <code>\markdownRendererTableAttributeContextEnd</code>   | 108                            |
| <code>\markdownRendererTextCite</code>                   | 80                             |
| <code>\markdownRendererThematicBreak</code>              | 110                            |
| <code>\markdownRendererTickedBox</code>                  | 111                            |
| <code>\markdownRendererUlBegin</code>                    | 78                             |
| <code>\markdownRendererUlBeginTight</code>               | 78                             |
| <code>\markdownRendererUlEnd</code>                      | 79                             |
| <code>\markdownRendererUlEndTight</code>                 | 79                             |
| <code>\markdownRendererUlItem</code>                     | 78                             |
| <code>\markdownRendererUlItemEnd</code>                  | 79                             |
| <code>\markdownRendererUntickedBox</code>                | 111                            |
| <code>\markdownRendererWarning</code>                    | 111                            |
| <code>\markdownSetup</code>                              | 55, 55, 59, 136, 143, 381, 382 |
| <code>\markdownSetupSnippet</code>                       | 69, 69                         |
| <code>\markdownWarning</code>                            | 132, 132                       |
| <code>\markinline</code>                                 | 52, 53, 54, 134, 136, 374, 379 |
| <code>\markinlinePlainTeX</code>                         | 379                            |
| <br>   |                                |
| <code>new</code>   | 8, 17, 18, 348, 350            |
| <code>notes</code>                                       | 37, 98                         |
| <br>   |                                |
| <code>parsers</code>                                     | 216, 253                       |
| <code>parsers.punctuation</code>                         | 217                            |
| <code>pipeTables</code>                                  | 7, 38, 44, 109                 |
| <code>preserveTabs</code>                                | 38, 42, 254                    |
| <br>   |                                |
| <code>rawAttribute</code>                                | 38, 39, 103                    |
| <code>reader</code>                                      | 8, 29, 145, 216, 253, 299      |
| <code>reader-&gt;add_special_character</code>            | 8, 10, 29, 293                 |
| <code>reader-&gt;auto_link_email</code>                  | 282                            |
| <code>reader-&gt;auto_link_url</code>                    | 282                            |
| <code>reader-&gt;create_parser</code>                    | 254                            |
| <code>reader-&gt;finalize_grammar</code>                 | 289, 354                       |
| <code>reader-&gt;initialize_named_group</code>           | 293                            |

|                               |   |
|-------------------------------|---|
| reader->insert_pattern        | 8, 9, 10, 29, 289, 295                    |
| reader->lookup_note_reference | 268                                       |
| reader->lookup_reference      | 268                                       |
| reader->normalize_tag         | 253                                       |
| reader->options               | 253                                       |
| reader->parser_functions      | 254                                       |
| reader->parser_functions.name | 254                                       |
| reader->parsers               | 253, 253                                  |
| reader->register_link         | 267                                       |
| reader->update_rule           | 289, 292, 295                             |
| reader->writer                | 253                                       |
| reader.new                    | 253, 253, 354                             |
| relativeReferences            | 39  |
|                               |   |
| \setupmarkdown                | 143                                       |
| shiftHeadings                 | 7, 40                                     |
| singletonCache                | 17  |
| slice                         | 7, 40, 197, 209, 210                      |
| smartEllipses                 | 41, 86, 137                               |
| \startmarkdown                | 142, 142, 411                             |
| startNumber                   | 41, 99–101                                |
| \stopmarkdown                 | 142, 142, 411                             |
| strikeThrough                 | 41, 107, 407                              |
| stripIndent                   | 42, 254                                   |
| stripPercentSigns             | 371                                       |
| subscripts                    | 42, 108                                   |
| superscripts                  | 43, 108                                   |
| syntax                        | 290, 295                                  |
|                               |   |
| tableAttributes               | 43, 108                                   |
| tableCaptions                 | 7, 43, 44, 108                            |
| taskLists                     | 44, 111, 397                              |
| texComments                   | 45, 255                                   |
| texMathDollars                | 45, 110                                   |
| texMathDoubleBackslash        | 46, 110                                   |
| texMathSingleBackslash        | 46, 110                                   |
| tightLists                    | 46, 78, 79, 84, 86, 98, 99, 102, 103, 387 |
|                               |   |
| underscores                   | 47  |
| unicodeNormalization          | 18, 19                                    |
| unicodeNormalizationForm      | 18, 19                                    |
| util.cache                    | 146, 146                                  |
| util.cache_verbatim           | 146                                       |

|                                 |                               |
|---------------------------------|-------------------------------|
| util.encode_json_string         | 146                           |
| util.err                        | 146                           |
| util.escaper                    | 149                           |
| util.expand_tabs_in_line        | 147                           |
| util.flatten                    | 147                           |
| util.intersperse                | 148                           |
| util.map                        | 149                           |
| util.pathname                   | 150                           |
| util.rope_last                  | 148                           |
| util.rope_to_string             | 148                           |
| util.salt                       | 150                           |
| util.table_copy                 | 146                           |
| util.walk                       | 147, 148                      |
| walkable_syntax                 | 9, 20, 26, 289, 292, 294, 295 |
| writer                          | 145, 145, 197, 299            |
| writer->active_attributes       | 208, 208–210                  |
| writer->attribute_type_levels   | 208                           |
| writer->attributes              | 206                           |
| writer->block_html_element      | 205                           |
| writer->blockquote              | 205                           |
| writer->bulletitem              | 203                           |
| writer->bulletlist              | 203                           |
| writer->citations               | 300                           |
| writer->code                    | 201                           |
| writer->contentblock            | 305                           |
| writer->defer_call              | 215, 215                      |
| writer->definitionlist          | 308                           |
| writer->display_math            | 340                           |
| writer->div_begin               | 321                           |
| writer->div_end                 | 321                           |
| writer->document                | 206                           |
| writer->ellipsis                | 199                           |
| writer->emphasis                | 205                           |
| writer->error                   | 201                           |
| writer->escape                  | 201                           |
| writer->escaped_chars           | 200, 200                      |
| writer->escaped_minimal_strings | 199, 200                      |
| writer->escaped_strings         | 200                           |
| writer->escaped_uri_chars       | 199, 200                      |
| writer->fancyitem               | 311                           |
| writer->fancylist               | 310                           |

|                             |               |
|-----------------------------|---------------|
| writer->fencedCode          | 316           |
| writer->flatten_inlines     | 197, 197      |
| writer->get_state           | 215           |
| writer->hard_line_break     | 199           |
| writer->heading             | 213           |
| writer->identifier          | 201           |
| writer->image               | 202           |
| writer->infostring          | 201           |
| writer->inline_html_comment | 204           |
| writer->inline_html_tag     | 204           |
| writer->inline_math         | 340           |
| writer->interblocksep       | 198           |
| writer->is_writing          | 197, 197      |
| writer->jekyllData          | 344           |
| writer->lineblock           | 328           |
| writer->link                | 202           |
| writer->mark                | 329           |
| writer->math                | 201           |
| writer->nbsp                | 198           |
| writer->note                | 331           |
| writer->options             | 197           |
| writer->ordereditem         | 204           |
| writer->orderedlist         | 203           |
| writer->paragraph           | 198           |
| writer->paragraphsep        | 199           |
| writer->plain               | 198           |
| writer->pop_attributes      | 208, 209, 210 |
| writer->push_attributes     | 208, 209, 210 |
| writer->rawBlock            | 317           |
| writer->rawInline           | 338           |
| writer->set_state           | 215           |
| writer->slice_begin         | 197           |
| writer->slice_end           | 197           |
| writer->soft_line_break     | 199           |
| writer->space               | 198           |
| writer->span                | 299           |
| writer->strike_through      | 338           |
| writer->string              | 201           |
| writer->strong              | 205           |
| writer->subscript           | 339           |
| writer->superscript         | 340           |
| writer->table               | 334           |

|  |                      |
|--|----------------------|
| <code>writer-&gt;thematic_break</code> | <i>199</i>           |
| <code>writer-&gt;checkbox</code>       | <i>205</i>           |
| <code>writer-&gt;undosep</code>        | <i>199, 298</i>      |
| <code>writer-&gt;uri</code>            | <i>201</i>           |
| <code>writer-&gt;verbatim</code>       | <i>205</i>           |
| <code>writer-&gt;warning</code>        | <i>201</i>           |
| <code>writer.new</code>                | <i>197, 197, 354</i> |