

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

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

Version 3.11.4-0-g260a31e6  
2025-06-24

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>	<b>3</b>	<b>Implementation</b>	<b>167</b>
1.1	Requirements . . . . .	2	3.1	Lua Implementation . . .	167
1.2	Feedback . . . . .	6	3.2	Plain T <sub>E</sub> X Implementation	381
1.3	Acknowledgements . . . .	7	3.3	L <sup>A</sup> T <sub>E</sub> X Implementation . .	424
<b>2</b>	<b>Interfaces</b>	<b>7</b>	3.4	ConT <sub>E</sub> Xt Implementation	465
2.1	Lua Interface . . . . .	7			
2.2	Plain T <sub>E</sub> X Interface . . . .	53		<b>References</b>	<b>476</b>
2.3	L <sup>A</sup> T <sub>E</sub> X Interface . . . . .	153			
2.4	ConT <sub>E</sub> Xt Interface . . . .	162		<b>Index</b>	<b>477</b>

## 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 . .	49
3	A sequence diagram of typesetting a document using the Lua CLI . . . . .	50
4	An example directed graph . . . . .	76
5	An example mindmap . . . . .	77
6	An example UML sequence diagram . . . . .	78
7	The banner of the Markdown package . . . . .	79
8	A pushdown automaton that recognizes T <sub>E</sub> X comments . . . . .	261

## 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.

---

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

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

Section 2 describes the interfaces exposed by the package. Section 3 describes the implementation of the package. The technical documentation contains only a limited number of tutorials and code examples. You can find more of these in the user manual.<sup>3</sup>

```

1 local metadata = {
2   version   = "(((VERSION)))",
3   comment   = "A module for the conversion from markdown "
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 LuaTeX engine (though not necessarily in the LuaMetaTeX engine).

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

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

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

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

**Kpathsea** A package that implements the loading of third-party Lua libraries and looking up files in the TeX directory structure.

---

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

```
16 ;(function()
```

If Kpathsea has not been loaded before or if LuaTeX has not yet been initialized, configure Kpathsea on top of loading it. Since ConTeXt 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.

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

All the abovelisted modules are statically linked into the current version of the LuaTeX 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 TeX Live  $\geq 2020$ .

```
24 hard lua-uni-algos
25 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 `jeekyllData` option is enabled.

```
26 hard lua-tinyyaml
```

### 1.1.2 Plain TeX Requirements

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

**expl3** A package that enables the expl3 language [2] from the L<sup>A</sup>T<sub>E</sub>X3 kernel in TeX Live  $\leq 2019$ . It is used to implement reflection capabilities that allow us to enumerate and inspect high-level concepts such as options, renderers, and renderer prototypes.

```
27 hard l3kernel
28 \unprotect
29 \expandafter\ifx\csname ExplSyntaxOn\endcsname\relax
30   \input expl3-generic
31 \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.

Note that this support for TeX engines other than LuaTeX comes with some limitations with respect to file and directory names. Specifically, the filenames of your .tex files may not contain spaces<sup>4</sup>. If `-output-directory` is provided, it may not contain spaces either.

32 `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<sub>ε</sub> format is loaded, a TeX engine that extends ε-TeX, and all the plain TeX prerequisites (see Section 1.1.2).

```
33 \NeedsTeXFormat{LaTeX2e}
34 \RequirePackage{expl3}
```

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 2.3.4) 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.

---

<sup>4</sup>See <https://github.com/Witiko/markdown/issues/573>.

35 `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.

36 `soft graphics`

**enumitem and paralist** Packages that provide macros for the default renderer prototypes for tight and fancy lists.

The package `paralist` will be used unless the option `experimental` has been enabled, in which case, the package `enumitem` will be used. Furthermore, enabling any test phase [3] will also cause `enumitem` to be used. In a future major version, `enumitem` will replace `paralist` altogether.

37 `soft enumitem`

38 `soft paralist`

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

39 `soft fancyvrb`

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

40 `soft csvsimple`

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

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

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

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

44 `soft amsmath`

45 `soft amssymb`

**graphicx** A package that provides extended support for graphics. It is used in the `witiko/diagrams`, and `witiko/graphicx/http` plain T<sub>E</sub>X themes, see Section 2.2.3.

46 `soft graphics`

47 `soft epstopdf` # required by ``graphics`` and ``graphicx``, which load ``epsopdf-base.sty``

48 `soft epstopdf-pkg` # required by ``graphics`` and ``graphicx``, which load ``epsopdf-base.sty``

**soul and xcolor** Packages that are used in the default renderer prototypes for strike-throughs and marked text in pdfTeX.

```
49 soft soul
50 soft xcolor
```

**lua-ul and luacolor** Packages that are used in the default renderer prototypes for strike-throughs and marked text in LuaTeX.

```
51 soft lua-ul
52 soft luacolor
```

**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.

```
53 soft ltxcmds
```

**luaxml** A package that is used to convert HTML to L<sup>A</sup>T<sub>E</sub>X in the default renderer prototypes for content blocks, raw blocks, and inline raw spans.

```
54 soft luaxml
```

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

```
55 soft verse
```

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

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

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

## 1.2 Feedback

Please use the Markdown project page on GitHub<sup>5</sup> to report bugs and submit feature requests. If you do not want to report a bug or request a feature but are simply in need of assistance, you might want to consider posting your question to the T<sub>E</sub>X-L<sup>A</sup>T<sub>E</sub>X Stack Exchange.<sup>6</sup> community question answering web site under the `markdown` tag.

---

<sup>5</sup>See <https://github.com/witiko/markdown/issues>.

<sup>6</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 [4] 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 T<sub>E</sub>X implementation of the package draws inspiration from several sources including the source code of L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>, the minted package by Geoffrey M. Poore, which likewise tackles the issue of interfacing with an external interpreter from T<sub>E</sub>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 T<sub>E</sub>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 T<sub>E</sub>X *token renderers* is exposed by the Lua layer. The plain T<sub>E</sub>X layer exposes the conversion capabilities of Lua as T<sub>E</sub>X macros. The L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt layers provide syntactic sugar on top of plain T<sub>E</sub>X macros. The user can interface with any and all layers.

### 2.1 Lua Interface

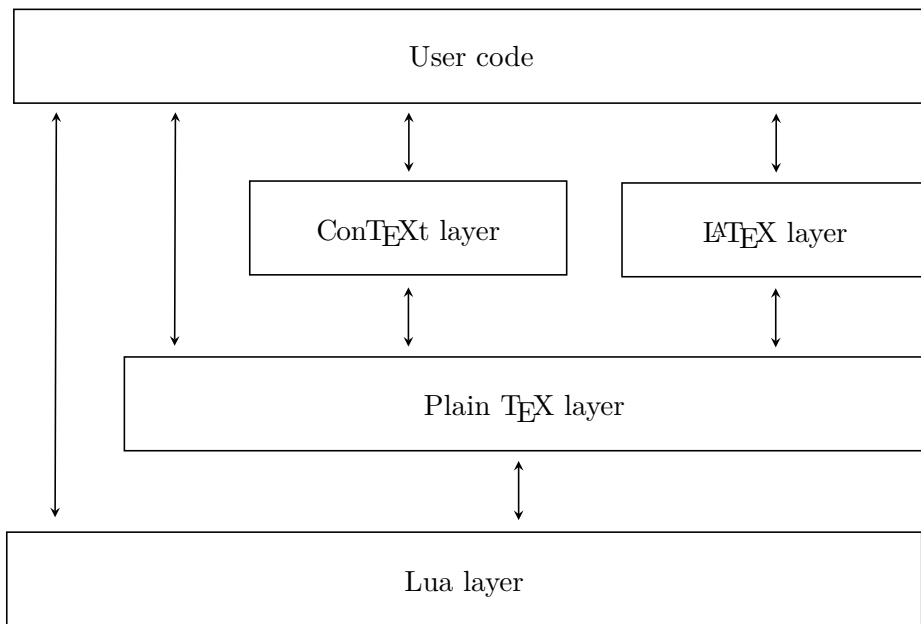
The Lua interface provides the conversion from UTF-8 encoded markdown to plain T<sub>E</sub>X. This interface is used by the plain T<sub>E</sub>X implementation (see Section 3.2) and will be of interest to the developers of other packages and Lua modules.

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

```
56 local M = {metadata = metadata}
```

#### 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`



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

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 TeX output using the default options and prints the TeX output:

```

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

```

### 2.1.2 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.

```

57 local walkable_syntax = {

```



```

58  Block = {
59      "Blockquote",
60      "Verbatim",
61      "ThematicBreak",
62      "BulletList",
63      "OrderedList",
64      "DisplayHtml",
65      "Heading",
66  },
67  BlockOrParagraph = {
68      "Block",
69      "Paragraph",
70      "Plain",
71  },
72  Inline = {
73      "Str",
74      "Space",
75      "Endline",
76      "EndlineBreak",
77      "LinkAndEmph",
78      "Code",
79      "AutoLinkUrl",
80      "AutoLinkEmail",
81      "AutoLinkRelativeReference",
82      "InlineHtml",
83      "HtmlEntity",
84      "EscapedChar",
85      "Smart",
86      "Symbol",
87  },
88 }

```

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.

```
89 local defaultOptions = {}
```

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

```
90 \ExplSyntaxOn
91 \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.

```
92 \prop_new:N \g_@@_lua_option_types_prop
93 \prop_new:N \g_@@_default_lua_options_prop
94 \seq_new:N \g_@@_option_layers_seq
95 \tl_const:Nn \c_@@_option_layer_lua_tl { lua }
96 \seq_gput_right:NV
97   \g_@@_option_layers_seq
98   \c_@@_option_layer_lua_tl
99 \cs_new:Nn
100   \@@_add_lua_option:nnn
101   {
102     \@@_add_option:Vnnn
103     \c_@@_option_layer_lua_tl
104     { #1 }
105     { #2 }
106     { #3 }
107   }
108 \cs_new:Nn
109   \@@_add_option:nnnn
110   {
111     \seq_gput_right:cn
112       { g_@@_ #1 _options_seq }
113       { #2 }
114     \prop_gput:cnn
115       { g_@@_ #1 _option_types_prop }
116       { #2 }
117       { #3 }
118     \prop_gput:cnn
119       { g_@@_default_ #1 _options_prop }
120       { #2 }
121       { #4 }
122     \@@_typecheck_option:n
123       { #2 }
124   }
```

```

125 \cs_generate_variant:Nn
126   \@@_add_option:nnnn
127   { Vnnn }
128 \tl_const:Nn \c_@@_option_value_true_tl { true }
129 \tl_const:Nn \c_@@_option_value_false_tl { false }
130 \cs_new:Nn \@@_typecheck_option:n
131   {
132     \@@_get_option_type:nN
133     { #1 }
134     \l_tmpa_tl
135     \str_case_e:Vn
136     \l_tmpa_tl
137     {
138       { \c_@@_option_type_boolean_tl }
139       {
140         \@@_get_option_value:nN
141         { #1 }
142         \l_tmpa_tl
143         \bool_if:nF
144         {
145           \str_if_eq_p:VV
146           \l_tmpa_tl
147           \c_@@_option_value_true_tl ||
148           \str_if_eq_p:VV
149           \l_tmpa_tl
150           \c_@@_option_value_false_tl
151         }
152         {
153           \msg_error:nnnV
154           { markdown }
155           { failed-typecheck-for-boolean-option }
156           { #1 }
157           \l_tmpa_tl
158         }
159       }
160     }
161   }
162 \msg_new:nnn
163   { markdown }
164   { failed-typecheck-for-boolean-option }
165   {
166     Option~#1~has~value~#2,~
167     but~a~boolean~(true~or~false)~was~expected.
168   }
169 \cs_generate_variant:Nn
170   \str_case_e:nn
171   { Vn }

```

```

172 \cs_generate_variant:Nn
173   \msg_error:nnnn
174   { nnnV }
175 \seq_new:N
176   \g_@@_option_types_seq
177 \tl_const:Nn
178   \c_@@_option_type_clist_tl
179   { clist }
180 \seq_gput_right:NV
181   \g_@@_option_types_seq
182   \c_@@_option_type_clist_tl
183 \tl_const:Nn
184   \c_@@_option_type_counter_tl
185   { counter }
186 \seq_gput_right:NV
187   \g_@@_option_types_seq
188   \c_@@_option_type_counter_tl
189 \tl_const:Nn
190   \c_@@_option_type_boolean_tl
191   { boolean }
192 \seq_gput_right:NV
193   \g_@@_option_types_seq
194   \c_@@_option_type_boolean_tl
195 \tl_const:Nn
196   \c_@@_option_type_number_tl
197   { number }
198 \seq_gput_right:NV
199   \g_@@_option_types_seq
200   \c_@@_option_type_number_tl
201 \tl_const:Nn
202   \c_@@_option_type_path_tl
203   { path }
204 \seq_gput_right:NV
205   \g_@@_option_types_seq
206   \c_@@_option_type_path_tl
207 \tl_const:Nn
208   \c_@@_option_type_slice_tl
209   { slice }
210 \seq_gput_right:NV
211   \g_@@_option_types_seq
212   \c_@@_option_type_slice_tl
213 \tl_const:Nn
214   \c_@@_option_type_string_tl
215   { string }
216 \seq_gput_right:NV
217   \g_@@_option_types_seq
218   \c_@@_option_type_string_tl

```

```

219 \cs_new:Nn
220   \@@_get_option_type:nN
221   {
222     \bool_set_false:N
223       \l_tmpa_bool
224     \seq_map_inline:Nn
225       \g_@@_option_layers_seq
226       {
227         \prop_get:cnNT
228           { g_@@_ ##1 _option_types_prop }
229           { #1 }
230         \l_tmpa_tl
231         {
232           \bool_set_true:N
233             \l_tmpa_bool
234           \seq_map_break:
235         }
236       }
237     \bool_if:NF
238       \l_tmpa_bool
239       {
240         \msg_error:nnn
241           { markdown }
242           { undefined-option }
243           { #1 }
244       }
245     \seq_if_in:NVF
246       \g_@@_option_types_seq
247       \l_tmpa_tl
248       {
249         \msg_error:nnnV
250           { markdown }
251           { unknown-option-type }
252           { #1 }
253         \l_tmpa_tl
254       }
255     \tl_set_eq:NN
256       #2
257       \l_tmpa_tl
258   }
259 \msg_new:nnn
260   { markdown }
261   { unknown-option-type }
262   {
263     Option~#1~has~unknown~type~#2.
264   }
265 \msg_new:nnn

```

```

266 { markdown }
267 { undefined-option }
268 {
269     Option~#1~is~undefined.
270 }
271 \cs_new:Nn
272   \@@_get_default_option_value:nN
273   {
274     \bool_set_false:N
275       \l_tmpa_bool
276     \seq_map_inline:Nn
277       \g_@@_option_layers_seq
278       {
279         \prop_get:cnNT
280           { g_@@_default_ ##1 _options_prop }
281           { #1 }
282         #2
283         {
284           \bool_set_true:N
285             \l_tmpa_bool
286           \seq_map_break:
287         }
288       }
289     \bool_if:NF
290       \l_tmpa_bool
291       {
292         \msg_error:nnn
293           { markdown }
294           { undefined-option }
295           { #1 }
296       }
297   }
298 \cs_new:Nn
299   \@@_get_option_value:nN
300   {
301     \@@_option_tl_to_csname:nN
302       { #1 }
303     \l_tmpa_tl
304     \cs_if_free:cTF
305       { \l_tmpa_tl }
306       {
307         \@@_get_default_option_value:nN
308           { #1 }
309         #2
310       }
311     {
312       \@@_get_option_type:nN

```

```

313         { #1 }
314         \l_tmpa_tl
315     \str_if_eq:NNTF
316         \c_@@_option_type_counter_tl
317         \l_tmpa_tl
318     {
319         \@@_option_tl_to_csname:nN
320         { #1 }
321         \l_tmpa_tl
322         \tl_set:Nx
323             #2
324         { \the \cs:w \l_tmpa_tl \cs_end: } % noqa: W200
325     }
326     {
327         \@@_option_tl_to_csname:nN
328         { #1 }
329         \l_tmpa_tl
330         \tl_set:Nv
331             #2
332         { \l_tmpa_tl }
333     }
334 }
335 }
336 \cs_new:Nn \@@_option_tl_to_csname:nN
337 {
338     \tl_set:Nn
339         \l_tmpa_tl
340         { \str_uppercase:n { #1 } }
341     \tl_set:Nx
342         #2
343     {
344         markdownOption
345         \tl_head:f { \l_tmpa_tl }
346         \tl_tail:n { #1 }
347     }
348 }

```

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.

```

349 \cs_new:Nn \@@_with_various_cases:nn
350 {
351     \seq_clear:N
352     \l_tmpa_seq
353     \seq_map_inline:Nn
354         \g_@@_cases_seq
355     {

```

```

356     \tl_set:Nn
357     \l_tmpa_tl
358     { #1 }
359     \use:c { ##1 }
360     \l_tmpa_tl
361     \seq_put_right:NV
362     \l_tmpa_seq
363     \l_tmpa_tl
364   }
365   \seq_map_inline:Nn
366   \l_tmpa_seq
367   { #2 }
368 }

```

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

```

369 \seq_new:N \g_@@_cases_seq
370 \cs_new:Nn \@@_camel_case:N % noqa: w401
371 {
372   \regex_replace_all:nnN
373   { _ ([a-z]) }
374   { \c { str_uppercase:n } \cB\{ \1 \cE\} }
375   #1
376   \tl_set:Nx
377   #1
378   { #1 }
379 }
380 \seq_gput_right:Nn \g_@@_cases_seq { @@_camel_case:N }
381 \cs_new:Nn \@@_snake_case:N % noqa: w401
382 {
383   \regex_replace_all:nnN
384   { ([a-z])([A-Z]) }
385   { \1 _ \c { str_lowercase:n } \cB\{ \2 \cE\} }
386   #1
387   \tl_set:Nx
388   #1
389   { #1 }
390 }
391 \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.

```
392 \@@_add_lua_option:nnn
393   { eagerCache }
394   { boolean }
395   { true }
396 defaultOptions.eagerCache = true
```

`experimental=true, false`

default: `false`

**true**      Experimental features that are planned to be the new default in the next major release of the Markdown package will be enabled.

At the moment, this just means that the version `experimental` of the theme `witiko/markdown/defaults` will be loaded and warnings for hard-deprecated features will become errors. However, the effects may extend to other areas in the future as well.

**false**      Experimental features will be disabled.

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

`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>7</sup>. This was the default behavior until version 3.0.0 of the Markdown package.

```
402 \@@_add_lua_option:nnn
403   { singletonCache }
404   { boolean }
405   { true }

406 defaultOptions.singletonCache = true

407 local singletonCache = {
408   convert = nil,
409   options = nil,
410 }
```

`unicodeNormalization=true, false`

default: `true`

**true** Markdown documents will be normalized using one of the four Unicode normalization forms<sup>8</sup> before conversion. The Unicode normalization norm used is determined by option `unicodeNormalizationForm`.

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

```
411 \@@_add_lua_option:nnn
412   { unicodeNormalization }
413   { boolean }
414   { true }

415 defaultOptions.unicodeNormalization = true
```

---

<sup>7</sup>See <https://github.com/witiko/markdown/pull/226#issuecomment-1599641634>.

<sup>8</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.

```
416 \@@_add_lua_option:nnn
417   { unicodeNormalizationForm }
418   { string }
419   { nfc }
420 defaultOptions.unicodeNormalizationForm = "nfc"
```

### 2.1.5 File and Directory Names

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

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

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

```
421 \str_new:N
422   \g_@@_unquoted_jobname_str
423 \str_set:NV
424   \g_@@_unquoted_jobname_str
425   \c_sys_jobname_str
426 \regex_replace_all:nnN
427   { \A ("|') ( .* ) ("|') \Z }
```

```

428 { \2 }
429 \g_@@_unquoted_jobname_str
430 \@@_add_lua_option:nnn
431 { cacheDir }
432 { path }
433 {
434     \markdownOptionOutputDir
435     / _markdown_
436     \str_use:N
437     \g_@@_unquoted_jobname_str
438 }
439 defaultOptions.cacheDir = "."

```

**contentBlocksLanguageMap**=*<filename>*

default: markdown-languages.json

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

```

440 \@@_add_lua_option:nnn
441 { contentBlocksLanguageMap }
442 { path }
443 { markdown-languages.json }
444 defaultOptions.contentBlocksLanguageMap = "markdown-languages.json"

```

**debugExtensionsFileName**=*<filename>*

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.

```

445 \@@_add_lua_option:nnn
446 { debugExtensionsFileName }
447 { path }
448 {
449     \markdownOptionOutputDir
450     /
451     \str_use:N
452     \g_@@_unquoted_jobname_str
453     .debug-extensions.json
454 }
455 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  $\text{\TeX}$  document that contains markdown documents without invoking Lua using the `frozenCache` plain  $\text{\TeX}$  option. As a result, the plain  $\text{\TeX}$  document becomes more portable, but further changes in the order and the content of markdown documents will not be reflected.

```
456 \@@_add_lua_option:nnn
457   { frozenCacheFileName }
458   { path }
459   { \markdownOptionCacheDir / frozenCache.tex }
460 defaultOptions.frozenCacheFileName = "frozenCache.tex"
```

### 2.1.6 Parser Options

`autoIdentifiers`=true, false default: false

**true** Enable the Pandoc auto identifiers syntax extension<sup>9</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`.

```
461 \@@_add_lua_option:nnn
462   { autoIdentifiers }
463   { boolean }
464   { false }
465 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.

---

<sup>9</sup>See [https://pandoc.org/MANUAL.html#extension-auto\\_identifiers](https://pandoc.org/MANUAL.html#extension-auto_identifiers).

```

466 \@@_add_lua_option:nnn
467   { blankBeforeBlockquote }
468   { boolean }
469   { false }

470 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.

```

471 \@@_add_lua_option:nnn
472   { blankBeforeCodeFence }
473   { boolean }
474   { false }

475 defaultOptions.blankBeforeCodeFence = false

```

`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.

```

476 \@@_add_lua_option:nnn
477   { blankBeforeDivFence }
478   { boolean }
479   { false }

480 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.

```

481 \@@_add_lua_option:nnn
482   { blankBeforeHeading }
483   { boolean }
484   { false }

485 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.

```
486 \@@_add_lua_option:nnn
487   { blankBeforeList }
488   { boolean }
489   { false }
490 defaultOptions.blankBeforeList = false
```

`bracketedSpans=true, false` default: false

- `true`      Enable the Pandoc bracketed span syntax extension<sup>10</sup>:

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

- `false`      Disable the Pandoc bracketed span syntax extension.

```
491 \@@_add_lua_option:nnn
492   { bracketedSpans }
493   { boolean }
494   { false }
495 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.

```
496 \@@_add_lua_option:nnn
497   { breakableBlockquotes }
498   { boolean }
499   { true }
500 defaultOptions.breakableBlockquotes = true
```

---

<sup>10</sup>See [https://pandoc.org/MANUAL.html#extension-bracketed\\_spans](https://pandoc.org/MANUAL.html#extension-bracketed_spans).

`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.

```
501 \@@_add_lua_option:nnn
502 { citationNbsps }
503 { boolean }
504 { true }

505 defaultOptions.citationNbsps = true
```

`citations=true, false`

default: false

- true** Enable the Pandoc citation syntax extension<sup>11</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.

```
506 \@@_add_lua_option:nnn
507 { citations }
508 { boolean }
509 { false }

510 defaultOptions.citations = false
```

---

<sup>11</sup>See <https://pandoc.org/MANUAL.html#extension-citations>.



`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.``
```

```
511 \@@_add_lua_option:nnn  
512 { codeSpans }  
513 { boolean }  
514 { true }  
  
515 defaultOptions.codeSpans = true
```

`contentBlocks=true, false`

default: false

**true**

: Enable the iA Writer content blocks syntax extension [5]:

```
``` 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.

```
516 \@@_add_lua_option:nnn  
517 { contentBlocks }  
518 { boolean }  
519 { false }  
  
520 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
```

```
521 \@@_add_lua_option:nnn
522   { contentLevel }
523   { string }
524   { block }
525 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.

```
526 \@@_add_lua_option:nnn
527   { debugExtensions }
528   { boolean }
529   { false }
530 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.

```

531 \@@_add_lua_option:nnn
532   { definitionLists }
533   { boolean }
534   { false }

535 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.

```

536 \@@_add_lua_option:nnn
537   { ensureJekyllData }
538   { boolean }
539   { false }

540 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}

```

```

541 \@@_add_lua_option:nnn
542   { expectJekyllData }
543   { boolean }

```

```
544 { false }
```

```
545 defaultOptions.expectJekyllData = false
```

`extensions`= $\langle filenames \rangle$

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 T<sub>E</sub>X 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:

```
546 metadata.user_extension_api_version = 2
```

```
547 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.

```
548 \cs_generate_variant:Nn
549   \@@_add_lua_option:nnn
550   { nnV }
551 \@@_add_lua_option:nnV
552   { extensions }
553   { clist }
554   \c_empty_clist
555 defaultOptions.extensions = {}
```

`fancyLists=true, false`

default: false

`true` Enable the Pandoc fancy list syntax extension<sup>12</sup>:

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

`false` Disable the Pandoc fancy list syntax extension.

```
556 \@@_add_lua_option:nnn
557   { fancyLists }
558   { boolean }
559   { false }
560 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>
```

---

<sup>12</sup>See <https://pandoc.org/MANUAL.html#org-fancy-lists>.

```

<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.

```

561 \@@_add_lua_option:nnn
562 { fencedCode }
563 { boolean }
564 { true }

565 defaultOptions.fencedCode = true

```

**fencedCodeAttributes=true, false**      default: false

**true**      Enable the Pandoc fenced code attribute syntax extension<sup>13</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.

```

566 \@@_add_lua_option:nnn
567 { fencedCodeAttributes }
568 { boolean }
569 { false }

570 defaultOptions.fencedCodeAttributes = false

```

**fencedDivs=true, false**      default: false

**true**      Enable the Pandoc fenced div syntax extension<sup>14</sup>:

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

<sup>14</sup>See [https://pandoc.org/MANUAL.html#extension-fenced\\_divs](https://pandoc.org/MANUAL.html#extension-fenced_divs).

```

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

And another.
:::

```

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

```

571 \@@_add_lua_option:nnn
572   { fencedDivs }
573   { boolean }
574   { false }
575 defaultOptions.fencedDivs = false

```

**finalizeCache**=true, false default: false

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

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

```

576 \@@_add_lua_option:nnn
577   { finalizeCache }
578   { boolean }
579   { false }
580 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 T<sub>E</sub>X macro **\markdownFrozenCache***<number>* that will typeset markdown document number *<number>*.

```

581 \@@_add_lua_option:nnn
582   { frozenCacheCounter }
583   { counter }
584   { 0 }
585 defaultOptions.frozenCacheCounter = 0

```



`gfmAutoIdentifiers=true, false` default: false

**true** Enable the Pandoc GitHub-flavored auto identifiers syntax extension<sup>15</sup>:

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

`# 123 Sesame Street`

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

See also the option `autoIdentifiers`.

```
586 \@@_add_lua_option:nnn
587   { gfmAutoIdentifiers }
588   { boolean }
589   { false }
590 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.

```
591 \@@_add_lua_option:nnn
592   { hashEnumerators }
593   { boolean }
594   { false }
595 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}`  
`=====`

---

<sup>15</sup>See [https://pandoc.org/MANUAL.html#extension-gfm\\_auto\\_identifiers](https://pandoc.org/MANUAL.html#extension-gfm_auto_identifiers).

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

```
596 \@@_add_lua_option:nnn
597   { headerAttributes }
598   { boolean }
599   { false }

600 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.

```
601 \@@_add_lua_option:nnn
602   { html }
603   { boolean }
604   { true }

605 defaultOptions.html = true
```

**hybrid=true, false**      default: **false**

**true**      Disable the escaping of special plain T<sub>E</sub>X characters, which makes it possible to intersperse your markdown markup with T<sub>E</sub>X code. The intended usage is in documents prepared manually by a human author. In such documents, it can often be desirable to mix T<sub>E</sub>X and markdown markup freely.

**false**      Enable the escaping of special plain T<sub>E</sub>X characters outside verbatim environments, so that they are not interpreted by T<sub>E</sub>X. This is encouraged when typesetting automatically generated content or markdown documents that were not prepared with this package in mind.

The **hybrid** option makes it difficult to untangle T<sub>E</sub>X input from markdown text, which makes documents written with the **hybrid** option less interoperable and more difficult to read for authors. Therefore, the option has been soft-deprecated in version 3.7.1 of the Markdown package: It will never be removed but using it prints a warning and is discouraged.

Consider one of the following better alternatives for mixing T<sub>E</sub>X and markdown:

- With the `contentBlocks` option, authors can move large blocks of TeX code to separate files and include them in their markdown documents as external resources:

```
Here is a mathematical formula:

/math-formula.tex
```

- With the `rawAttribute` option, authors can denote raw text spans and code blocks that will be interpreted as TeX code:

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

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

- With options `texMathDollars`, `texMathSingleBackslash`, and `texMathDoubleBackslash`, authors can freely type TeX commands between dollar signs or backslash-escaped brackets:

```
$H_2 O$ is a liquid.

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

```
606 \@@_add_lua_option:nnn
607   { hybrid }
608   { boolean }
609   { false }
610 defaultOptions.hybrid = false
```

`inlineCodeAttributes=true, false`

default: false

`true` Enable the Pandoc inline code span attribute extension<sup>16</sup>:

```
`<$>`{.haskell}
```

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

```
611 \@@_add_lua_option:nnn
612 { inlineCodeAttributes }
613 { boolean }
614 { false }

615 defaultOptions.inlineCodeAttributes = false
```

`inlineNotes=true, false`

default: false

`true` Enable the Pandoc inline note syntax extension<sup>17</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.

```
616 \@@_add_lua_option:nnn
617 { inlineNotes }
618 { boolean }
619 { false }

620 defaultOptions.inlineNotes = false
```

`jeekyllData=true, false`

default: false

`true` Enable the Pandoc YAML metadata block syntax extension<sup>18</sup> for entering metadata in YAML:

```
---
title: 'This is the title: it contains a colon'
author:
- Author One
- Author Two
keywords: [nothing, nothingness]
```

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

<sup>17</sup>See [https://pandoc.org/MANUAL.html#extension-inline\\_notes](https://pandoc.org/MANUAL.html#extension-inline_notes).

<sup>18</sup>See [https://pandoc.org/MANUAL.html#extension-yaml\\_metadata\\_block](https://pandoc.org/MANUAL.html#extension-yaml_metadata_block).

```
abstract: |
  This is the abstract.

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

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

```
621 \@@_add_lua_option:nnn
622 { jekyllData }
623 { boolean }
624 { false }

625 defaultOptions.jekyllData = false
```

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

**true** Enable the Pandoc link and image attribute syntax extension<sup>19</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.

```
626 \@@_add_lua_option:nnn
627 { linkAttributes }
628 { boolean }
629 { false }

630 defaultOptions.linkAttributes = false
```

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

**true** Enable the Pandoc line block syntax extension<sup>20</sup>:

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

<sup>19</sup>See [https://pandoc.org/MANUAL.html#extension-link\\_attributes](https://pandoc.org/MANUAL.html#extension-link_attributes).

<sup>20</sup>See [https://pandoc.org/MANUAL.html#extension-line\\_blocks](https://pandoc.org/MANUAL.html#extension-line_blocks).

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

```
631 \@@_add_lua_option:nnn
632 { lineBlocks }
633 { boolean }
634 { false }
635 defaultOptions.lineBlocks = false
```

**mark**=true, false default: false

**true**      Enable the Pandoc mark syntax extension<sup>21</sup>:

This ==is highlighted text.==

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

```
636 \@@_add_lua_option:nnn
637 { mark }
638 { boolean }
639 { false }
640 defaultOptions.mark = false
```

**notes**=true, false default: false

**true**      Enable the Pandoc note syntax extension<sup>22</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.

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

<sup>22</sup>See <https://pandoc.org/MANUAL.html#extension-footnotes>.

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

```
641 \@@_add_lua_option:nnn
642   { notes }
643   { boolean }
644   { false }

645 defaultOptions.notes = false
```

**pipeTables=true, false**      default: false

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

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

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

```
646 \@@_add_lua_option:nnn
647   { pipeTables }
648   { boolean }
649   { false }

650 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.

```
651 \@@_add_lua_option:nnn
652   { preserveTabs }
653   { boolean }
654   { true }

655 defaultOptions.preserveTabs = true
```

`rawAttribute=true, false`

default: `false`

`true` Enable the Pandoc raw attribute syntax extension<sup>23</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}
\[distance[i] =
    \begin{dcases}
      a & b \\
      c & d
    \end{dcases}
\]
```

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.

```
656 \@@_add_lua_option:nnn
657   { rawAttribute }
658   { boolean }
659   { false }

660 defaultOptions.rawAttribute = false
```

`relativeReferences=true, false`

default: `false`

`true` Enable relative references<sup>24</sup> in autolinks:

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

Conclusion {#conclusion}
=====

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

<sup>23</sup>See [https://pandoc.org/MANUAL.html#extension-raw\\_attribute](https://pandoc.org/MANUAL.html#extension-raw_attribute).

<sup>24</sup>See <https://datatracker.ietf.org/doc/html/rfc3986#section-4.2>.



**false**      Disable relative references in autolinks.

```
661 \@@_add_lua_option:nnn
662 { relativeReferences }
663 { boolean }
664 { false }
665 defaultOptions.relativeReferences = false
```

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

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

```
666 \@@_add_lua_option:nnn
667 { shiftHeadings }
668 { number }
669 { 0 }
670 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.

```
671 \@@_add_lua_option:nnn
672 { slice }
673 { slice }
674 { ^~$ }
675 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.

```
676 \@@_add_lua_option:nnn
677   { smartEllipses }
678   { boolean }
679   { false }

680 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 `\markdownRendererOliItemWithNumber` TeX macro.

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

```
681 \@@_add_lua_option:nnn
682   { startNumber }
683   { boolean }
684   { true }

685 defaultOptions.startNumber = true
```

`strikeThrough=true, false` default: false

`true` Enable the Pandoc strike-through syntax extension<sup>25</sup>:

This ~~is deleted text.~~

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

```
686 \@@_add_lua_option:nnn
687   { strikeThrough }
688   { boolean }
689   { false }

690 defaultOptions.strikeThrough = false
```

---

<sup>25</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.

```
691 \@@_add_lua_option:nnn
692   { stripIndent }
693   { boolean }
694   { false }
695 defaultOptions.stripIndent = false
```

`subscripts=true, false`

default: false

**true** Enable the Pandoc subscript syntax extension<sup>26</sup>:

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

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

```
696 \@@_add_lua_option:nnn
697   { subscripts }
698   { boolean }
699   { false }
700 defaultOptions.subscripts = false
```

---

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

`superscripts=true, false`

default: false

**true** Enable the Pandoc superscript syntax extension<sup>27</sup>:

`210` is 1024.

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

```
701 \@@_add_lua_option:nnn
702 { superscripts }
703 { boolean }
704 { false }
705 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.

```
706 \@@_add_lua_option:nnn
707 { tableAttributes }
708 { boolean }
709 { false }
710 defaultOptions.tableAttributes = false
```

---

<sup>27</sup>See <https://pandoc.org/MANUAL.html#extension-superscript-subscript>.

`tableCaptions=true, false`

default: `false`

`true`

: Enable the Pandoc table caption syntax extension<sup>28</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.

```
711 \@@_add_lua_option:nnn
712   { tableCaptions }
713   { boolean }
714   { false }

715 defaultOptions.tableCaptions = false
```

`taskLists=true, false`

default: `false`

`true` Enable the Pandoc task list syntax extension<sup>29</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.

```
716 \@@_add_lua_option:nnn
717   { taskLists }
718   { boolean }
719   { false }

720 defaultOptions.taskLists = false
```

<sup>28</sup>See [https://pandoc.org/MANUAL.html#extension-table\\_captions](https://pandoc.org/MANUAL.html#extension-table_captions).

<sup>29</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.

```
721 \@@_add_lua_option:nnn
722   { texComments }
723   { boolean }
724   { false }
725 defaultOptions.texComments = false
```

`texMathDollars=true, false`

default: false

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

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

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

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

```
726 \@@_add_lua_option:nnn
727   { texMathDollars }
728   { boolean }
729   { false }
730 defaultOptions.texMathDollars = false
```

---

<sup>30</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>31</sup>:

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

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

```
731 \@@_add_lua_option:nnn
732   { texMathDoubleBackslash }
733   { boolean }
734   { false }

735 defaultOptions.texMathDoubleBackslash = false
```

`texMathSingleBackslash=true, false` default: false

**true** Enable the Pandoc single backslash math syntax extension<sup>32</sup>:

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

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

```
736 \@@_add_lua_option:nnn
737   { texMathSingleBackslash }
738   { boolean }
739   { false }

740 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>31</sup>See [https://pandoc.org/MANUAL.html#extension-tex\\_math\\_double\\_backslash](https://pandoc.org/MANUAL.html#extension-tex_math_double_backslash).

<sup>32</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.

```

741 \@@_add_lua_option:nnn
742   { tightLists }
743   { boolean }
744   { true }

745 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.

```

746 \@@_add_lua_option:nnn
747   { underscores }
748   { boolean }
749   { true }
750 \ExplSyntaxOff

751 defaultOptions.underscores = true

```

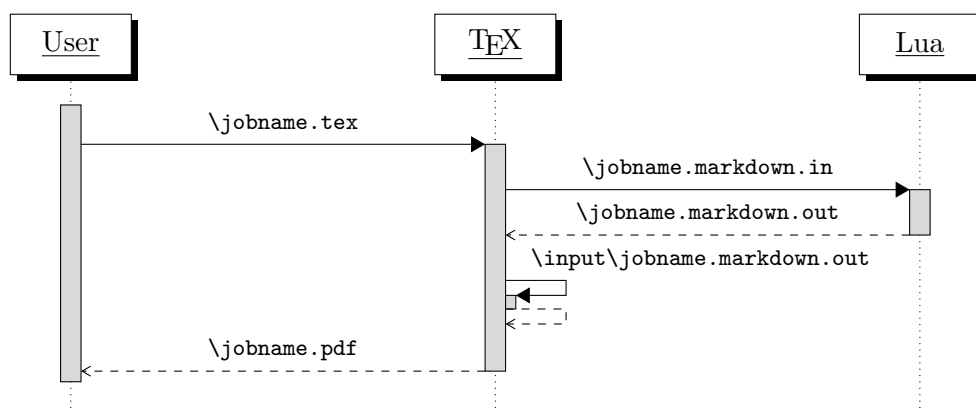


### 2.1.7 Command-Line Interface

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

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

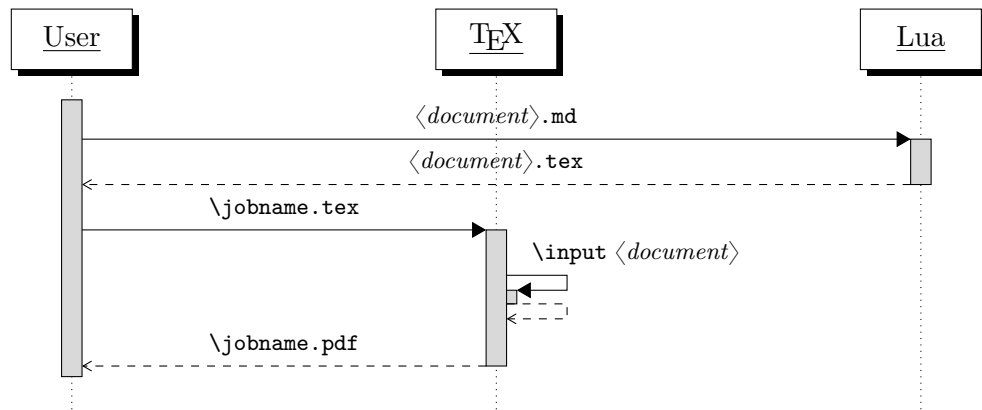
A solution to the above problems is to decouple the conversion from the typesetting. For this reason, a command-line Lua interface for converting a markdown document to  $\text{\TeX}$  is also provided, see Figure 3.



**Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the  $\text{\TeX}$  interface**

```

752 .TH MARKDOWN2TEX 1 "(((LASTMODIFIED)))"
753 .SH NAME
754 markdown2tex \- convert .md files to .tex
755 .SH SYNOPSIS
</lua-cli-manpage> <*lua-cli>
756 local HELP_STRING = "Usage: " .. [[
</lua-cli> <*lua-cli,lua-cli-manpage>
757 markdown2tex [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
758
</lua-cli,lua-cli-manpage> <*lua-cli-manpage>
759 .SH DESCRIPTION
  
```



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

```

760 % \end{macrocode}
761 </lua-cli-manpage>
762 <*lua-cli, lua-cli-manpage>
763 % \begin{macrocode}
764 OPTIONS are documented in Section 2.2.1 of the Markdown Package User
765 Manual (https://ctan.org/pkg/markdown).
766
767 When OUTPUT_FILE is unspecified, the result of the conversion will be
768 written to the standard output. When INPUT_FILE is also unspecified, the
769 result of the conversion will be read from the standard input.
770 % \end{macrocode}
771 </lua-cli, lua-cli-manpage>
772 <*lua-cli>
773 % \begin{macrocode}
774
775 Report bugs to: witiko@mail.muni.cz
776 Markdown package home page: <https://github.com/witiko/markdown>]]
777
778 local VERSION_STRING = [[
779 markdown2tex (Markdown) ]] .. metadata.version .. [[
780
781 Copyright (C) ]] .. table.concat(metadata.copyright,
782                                     "\nCopyright (C) ") .. [[
783
784 License: ]] .. metadata.license
785
786 local function warn(s)
787   io.stderr:write("Warning: " .. s .. "\n")
788 end

```

```

789
790 local function error(s)
791   io.stderr:write("Error: " .. s .. "\n")
792   os.exit(1)
793 end

```

To make it easier to copy-and-paste options from Pandoc [6] 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 [7] also show that snake\_case is faster to read than camelCase.

```

794 local function camel_case(option_name)
795   local cased_option_name = option_name:gsub("_(%l)", function(match)
796     return match:sub(2, 2):upper()
797   end)
798   return cased_option_name
799 end
800
801 local function snake_case(option_name)
802   local cased_option_name = option_name:gsub("%l%u", function(match)
803     return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()
804   end)
805   return cased_option_name
806 end
807
808 local cases = {camel_case, snake_case}
809 local various_case_options = {}
810 for option_name, _ in pairs(defaultOptions) do
811   for _, case in ipairs(cases) do
812     various_case_options[case(option_name)] = option_name
813   end
814 end
815
816 local process_options = true
817 local options = {}
818 local input_filename
819 local output_filename
820 for i = 1, #arg do
821   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.

```

822     if arg[i] == "--" then
823       process_options = false
824       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.

```

825     elseif arg[i]:match("=") then
826         local key, value = arg[i]:match("(.-)=(.*)")
827         if defaultOptions[key] == nil and
828             various_case_options[key] ~= nil then
829             key = various_case_options[key]
830         end

```

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

```

831     local default_type = type(defaultOptions[key])
832     if default_type == "boolean" then
833         options[key] = (value == "true")
834     elseif default_type == "number" then
835         options[key] = tonumber(value)
836     elseif default_type == "table" then
837         options[key] = {}
838         for item in value:gmatch("[^,]+") do
839             table.insert(options[key], item)
840         end
841     else
842         if default_type ~= "string" then
843             if default_type == "nil" then
844                 warn('Option "' .. key .. '" not recognized.')
845             else
846                 warn('Option "' .. key .. '" type not recognized, ' ..
847                     'please file a report to the package maintainer.')
848             end
849             warn('Parsing the ' .. 'value "' .. value .. '" of option "' ..
850                 key .. '" as a string.')
851         end
852         options[key] = value
853     end
854     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.

```

855     elseif arg[i] == "--help" or arg[i] == "-h" then
856         print(HELP_STRING)
857         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.

```

858     elseif arg[i] == "--version" or arg[i] == "-v" then
859         print(VERSION_STRING)
860         os.exit()
861     end
862 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 T<sub>E</sub>X document.

```

863 if input_filename == nil then
864     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 T<sub>E</sub>X document that will result from the conversion.

```

865 elseif output_filename == nil then
866     output_filename = arg[i]
867 else
868     error('Unexpected argument: "' .. arg[i] .. '".')
869 end
870 ::continue::
871 end

```

The command-line Lua interface is implemented by the files [markdown-cli.lua](#) and [markdown2tex.lua](#), which can be invoked from the command line as follows:

```
markdown2tex 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.

```

872 \def\markdownLastModified{((LASTMODIFIED))}%
873 \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  $\text{\TeX}$  characters have the expected category codes, when  $\text{\input}$ ting the file.

## 2.2.1 Typesetting Markdown and YAML

The interface exposes the  $\text{\markdownBegin}$ ,  $\text{\markdownEnd}$ ,  $\text{\yamlBegin}$ ,  $\text{\yamlEnd}$ ,  $\text{\markinline}$ ,  $\text{\markdownInput}$ ,  $\text{\yamlInput}$ , and  $\text{\markdownEscape}$  macros.

### 2.2.1.1 Typesetting Markdown and YAML directly

The  $\text{\markdownBegin}$  macro marks the beginning of a markdown document fragment and the  $\text{\markdownEnd}$  macro marks its end.

```
874 \let\markdownBegin\relax
875 \let\markdownEnd\relax
```

You may prepend your own code to the  $\text{\markdownBegin}$  macro and redefine the  $\text{\markdownEnd}$  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  $\text{\markdownEnd}$  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  $\text{\markdownEnd}$  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  $\text{\TeX}$  [8, p. 46]. As a corollary, the  $\text{\markdownBegin}$  macro also ignores them.

The  $\text{\markdownBegin}$  and  $\text{\markdownEnd}$  macros will also consume the rest of the lines at which they appear. In the following example plain  $\text{\TeX}$  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  $\text{\markdownBegin}$  and  $\text{\markdownEnd}$  macros.

The following example plain  $\text{\TeX}$  code showcases the usage of the  $\text{\markdownBegin}$  and  $\text{\markdownEnd}$  macros:

```

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

```

The `\yamlBegin` macro marks the beginning of an YAML document fragment and the `\yamlEnd` macro marks its end.

```

876 \let\uyamlBegin\uyamlBegin
877 \def\uyamlEnd{\uyamlEnd\endgroup}

```

The `\yamlBegin` and `\yamlEnd` macros are subject to the same limitations as 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
\uyamlBegin
title: _Hello_ **world** ...
author: John Doe
\uyamlEnd
\bye

```

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

```

\input markdown
\uyamlSetup{jekyllData, expectJekyllData, ensureJekyllData}
\uyamlBegin
title: _Hello_ **world** ...
author: John Doe
\uyamlEnd
\bye

```

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

```

878 \let\markinline\markinline

```

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.

### 2.2.1.2 Typesetting Markdown and YAML from external documents

You can use the `\markdownInput` macro to include markdown documents, similarly to how you might use the `\input` TeX primitive to include TeX 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 TeX.

```
879 \let\markdownInput\relax
```

The macro `\markdownInput` is not subject to the limitations of the `\markdownBegin` and `\markdownEnd` macros.

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

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

You can use the `\yamlInput` macro to include YAML documents, similarly to how you might use the `\input` TeX primitive to include TeX documents. The `\yamlInput` macro accepts a single parameter with the filename of a YAML document and expands to the result of the conversion of the input YAML document to plain TeX.

```
880 \def\yamlInput#1{%
881   \begingroup
882   \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
883   \markdownInput{#1}%
884   \endgroup
885 }%
```

The macro `\yamlInput` is also not subject to the limitations of the `\markdownBegin` and `\markdownEnd` macros.

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



```

\input markdown
\yamlInput{hello.yml}
\bye

```

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

```

\input markdown
\yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}
\markdownInput{hello.yml}
\bye

```

### 2.2.1.3 Typesetting TeX from inside Markdown and YAML documents

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

```
886 \let\markdownEscape\relax
```

### 2.2.2 Options

The plain TeX options are represented by TeX commands. Some of them map directly to the options recognized by the Lua interface (see Section 2.1.3), while some of them are specific to the plain TeX interface.

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

```

887 \ExplSyntaxOn
888 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
889 \cs_generate_variant:Nn
890   \tl_const:Nn
891   { NV }
892 \tl_if_exist:NF
893   \c_@@_top_layer_tl
894   {
895     \tl_const:NV
896       \c_@@_top_layer_tl
897       \c_@@_option_layer_plain_tex_tl
898   }

```

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

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

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

```

900 \prop_new:N \g_@@_plain_tex_option_types_prop
901 \prop_new:N \g_@@_default_plain_tex_options_prop
902 \seq_gput_right:NV
903   \g_@@_option_layers_seq
904   \c_@@_option_layer_plain_tex_tl
905 \cs_new:Nn
906   \@@_add_plain_tex_option:nnn
907   {
908     \@@_add_option:Vnnn
909     \c_@@_option_layer_plain_tex_tl
910     { #1 }
911     { #2 }
912     { #3 }
913   }

```

The plain TeX options may be also be specified via the `\markdownSetup` macro. Here, the plain TeX options are represented by a comma-delimited list of  $\langle key \rangle = \langle value \rangle$  pairs. For boolean options, the  $= \langle value \rangle$  part is optional, and  $\langle key \rangle$  will be interpreted as  $\langle key \rangle = \text{true}$  if the  $= \langle value \rangle$  part has been omitted. The `\markdownSetup` macro receives the options to set up as its only argument.

```

914 \cs_new:Nn
915   \@@_setup:n
916   {
917     \keys_set:nn
918       { markdown/options }
919       { #1 }
920   }
921 \cs_gset_eq:NN
922   \markdownSetup
923   \@@_setup:n

```

The command `\yamlSetup` is also available as an alias for the command `\markdownSetup`.

```

924 \cs_gset_eq:NN
925   \yamlSetup
926   \markdownSetup

```

The `\markdownIfOption{<name>}{<iftrue>}{<iffalse>}` macro is provided for testing, whether the value of `\markdownOption{<name>}` is `true`. If the value is `true`, then  $\langle iftrue \rangle$  is expanded, otherwise  $\langle iffalse \rangle$  is expanded.

```

927 \prg_new_conditional:Nnn % noqa: w401
928   \@@_if_option:n
929   { TF, T, F }
930   {

```

```

931 \@@_get_option_type:nN
932 { #1 }
933 \l_tmpa_tl
934 \str_if_eq:NNF
935 \l_tmpa_tl
936 \c_@@_option_type_boolean_tl
937 {
938   \msg_error:nxxx
939   { markdown }
940   { expected-boolean-option }
941   { #1 }
942   { \l_tmpa_tl }
943 }
944 \@@_get_option_value:nN
945 { #1 }
946 \l_tmpa_tl
947 \str_if_eq:NNTF
948 \l_tmpa_tl
949 \c_@@_option_value_true_tl
950 { \prg_return_true: }
951 { \prg_return_false: }
952 }
953 \msg_new:nnn
954 { markdown }
955 { expected-boolean-option }
956 {
957   Option~#1~has~type~#2,~
958   but~a~boolean~was~expected.
959 }
960 \let
961 \markdownIfOption
962 \@@_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  $\text{\TeX}$  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  $\text{\TeX}$  document without invoking Lua. As a result, the plain  $\text{\TeX}$  document becomes more portable, but further changes in the order and the content of markdown documents will not be reflected. It defaults to `false`.

```

963 \@@_add_plain_tex_option:nnn
964 { frozenCache }

```

```

965 { boolean }
966 { 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  $\text{\TeX}$  document to populate and finalize the cache.
5. Enable the `frozenCache` option.
6. Publish the source code of the plain  $\text{\TeX}$  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  $\text{\TeX}$  source. It defaults to `\jobname.markdown.in`.

The expansion of this macro must not contain quotation marks (") or backslash symbols (\).

```

967 \@@_add_plain_tex_option:nnn
968 { inputTempFileName }
969 { path }
970 {
971   \str_use:N
972   \g_@@_unquoted_jobname_str
973   .markdown.in
974 }

```

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  $\text{\TeX}$  implementation. The option defaults to `.` or, since  $\text{\TeX}$  Live 2024, to the value of the `-output-directory` option of your  $\text{\TeX}$  engine.

In  $\text{\MikTeX}$ , this automatic detection is currently only supported with  $\text{\LuaTeX}$ <sup>33</sup>. If you need to use  $\text{\MikTeX}$  and cannot use  $\text{\LuaTeX}$ , you can either a) fix the automatic detection by setting the environmental variable `TEXMF_OUTPUT_DIRECTORY` manually or by setting the `\markdownOptionOutputDir` option manually.

The path must be set to the same value as the `-output-directory` option of your  $\text{\TeX}$  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.

```

975 \@@_add_plain_tex_option:nnn
976 { outputDir }
977 { path }
978 { . }

```

---

<sup>33</sup>See <https://github.com/MiKTeX/miktex/issues/1630>.

### 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 TeX 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 TeX formats should only use the plain TeX default definitions or whether they should also use the format-specific default definitions. Whereas plain TeX 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 TeX 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.

```
979 \@@_add_plain_tex_option:nnn  
980   { plain }  
981   { boolean }  
982   { 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.

```

983 \@@_add_plain_tex_option:nnn
984   { noDefaults }
985   { boolean }
986   { 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 [9] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```

987 \seq_gput_right:Nn
988   \g_@@_plain_tex_options_seq
989   { stripPercentSigns }
990 \prop_gput:Nnn
991   \g_@@_plain_tex_option_types_prop
992   { stripPercentSigns }
993   { boolean }
994 \prop_gput:Nnx
995   \g_@@_default_plain_tex_options_prop
996   { stripPercentSigns }
997   { 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.

```

998 \cs_new:Nn
999   \@@_define_option_commands_and_keyvals:
1000   {
1001     \seq_map_inline:Nn

```

```

1002     \g_@@_option_layers_seq
1003     {
1004         \seq_map_inline:cn
1005         { g_@@_ ##1 _options_seq }
1006         {
1007             \@@_define_option_command:n
1008             { #####1 }

```

To make it easier to copy-and-paste options from Pandoc [6] 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 [7] also show that snake\_case is faster to read than camelCase.

```

1009             \@@_with_various_cases:nn
1010             { #####1 }
1011             {
1012                 \@@_define_option_keyval:nnn
1013                 { ##1 }
1014                 { #####1 }
1015                 { #####1 }
1016             }
1017         }
1018     }
1019 }
1020 \cs_new:Nn
1021   \@@_define_option_command:n
1022   {

```

Use the `lt3luabridge` library to determine the default value of the `\markdownOptionOutputDir` macro.

```

1023   \str_if_eq:nnTF
1024   { #1 }
1025   { outputDir }
1026   { \@@_define_option_command_output_dir: }
1027   {

```

Do not override options defined before loading the package.

```

1028   \@@_option_tl_to_csname:nN
1029   { #1 }
1030   \l_tmpa_tl
1031   \cs_if_exist:cF
1032   { \l_tmpa_tl }
1033   {
1034       \@@_get_default_option_value:nN
1035       { #1 }
1036       \l_tmpa_tl
1037       \@@_set_option_value:nV
1038       { #1 }
1039       \l_tmpa_tl

```

```

1040     }
1041   }
1042 }
1043 \ExplSyntaxOff
1044 \input lt3luabridge.tex

```

Use the `lt3luabridge` library to determine the default value of the `\markdownOptionOutputDir` macro by using one of the following:

1. The `status.output_directory` variable [1, Section 10.2], which is available since LuaTeX 1.18.0 from TeX Live 2024 and in other TeX distributions like MikTeX since ca March 2024. We are only able to read this variable in LuaTeX and not other TeX engines.
2. The `TEXMF_OUTPUT_DIRECTORY` environmental variable, which is available since TeX Live 2024. We are only able to read this variable in TeX Live and not some other TeX distributions like MikTeX.

```

1045 \ExplSyntaxOn
1046 \cs_new:Nn
1047   \@@_define_option_command_output_dir:
1048 {
1049   \cs_if_free:NT
1050     \markdownOptionOutputDir
1051   {
1052     \bool_if:nTF
1053     {
1054       \cs_if_exist_p:N
1055         \luabridge_tl_set:Nn &&
1056       (
1057         \int_compare_p:nNn
1058           { \g_luabridge_method_int }
1059           =
1060           { \c_luabridge_method_directlua_int } ||
1061         \sys_if_shell_unrestricted_p:
1062       )
1063     }
1064   {

```

Set most catcodes to category 12 (other) to ensure that special characters in the output directory name such as backslashes (`\`) are not interpreted as control sequences.

```

1065     \group_begin:
1066     \cctab_select:N
1067     \c_str_cctab
1068     \luabridge_tl_set:Nn
1069     \l_tmpa_tl
1070     {

```



```

1071             print(
1072                 (status.output_directory)
1073                 or~os.getenv("TEXMF_OUTPUT_DIRECTORY")
1074                 or~"."
1075             )
1076         }
1077         \tl_gset:NV
1078         \markdownOptionOutputDir
1079         \l_tmpa_tl
1080         \group_end:
1081     }
1082     {
1083         \tl_gset:Nn
1084         \markdownOptionOutputDir
1085         { . }
1086     }
1087 }
1088 }
1089 \cs_new:Nn
1090 \@@_set_option_value:nn
1091 {
1092     \@@_define_option:n
1093     { #1 }
1094     \@@_get_option_type:nN
1095     { #1 }
1096     \l_tmpa_tl
1097     \str_if_eq:NNTF
1098     \c_@@_option_type_counter_tl
1099     \l_tmpa_tl
1100     {
1101         \@@_option_tl_to_csname:nN
1102         { #1 }
1103         \l_tmpa_tl
1104         \int_gset:cn
1105         { \l_tmpa_tl }
1106         { #2 }
1107     }
1108     {
1109         \@@_option_tl_to_csname:nN
1110         { #1 }
1111         \l_tmpa_tl
1112         \cs_set:cpn
1113         { \l_tmpa_tl }
1114         { #2 }
1115     }
1116 }
1117 \cs_generate_variant:Nn

```

```

1118 \@@_set_option_value:nn
1119 { nV }
1120 \cs_new:Nn
1121 \@@_define_option:n
1122 {
1123   \@@_option_tl_to_csname:nN
1124   { #1 }
1125   \l_tmpa_tl
1126   \cs_if_free:cT
1127   { \l_tmpa_tl }
1128   {
1129     \@@_get_option_type:nN
1130     { #1 }
1131     \l_tmpb_tl
1132     \str_if_eq:NNT
1133     \c_@@_option_type_counter_tl
1134     \l_tmpb_tl
1135     {
1136       \@@_option_tl_to_csname:nN
1137       { #1 }
1138       \l_tmpa_tl
1139       \int_new:c
1140       { \l_tmpa_tl }
1141     }
1142   }
1143 }
1144 \cs_new:Nn
1145 \@@_define_option_keyval:nnn
1146 {
1147   \prop_get:cnN
1148   { g_@@_ #1 _option_types_prop }
1149   { #2 }
1150   \l_tmpa_tl
1151   \str_if_eq:VVTF
1152   \l_tmpa_tl
1153   \c_@@_option_type_boolean_tl
1154   {
1155     \keys_define:nn
1156     { markdown/options }
1157     {

```

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

```

1158       #3 .code:n = {
1159         \tl_set:Nx
1160         \l_tmpa_tl
1161         {

```

```

1162         \str_case:nnF
1163         { ##1 }
1164         {
1165             { yes } { true }
1166             { no } { false }
1167         }
1168         { ##1 }
1169     }
1170     \@@_set_option_value:nV
1171     { #2 }
1172     \l_tmpa_tl
1173 },
1174 #3 .default:n = { true },
1175 }
1176 }
1177 {
1178     \keys_define:nn
1179     { markdown/options }
1180     {
1181         #3 .code:n = {
1182             \@@_set_option_value:nn
1183             { #2 }
1184             { ##1 }
1185         },
1186     }
1187 }

```

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.

```

1188     \str_if_eq:VVT
1189     \l_tmpa_tl
1190     \c_@@_option_type_clist_tl
1191     {
1192         \tl_set:Nn
1193         \l_tmpa_tl
1194         { #3 }
1195         \tl_reverse:N
1196         \l_tmpa_tl
1197         \str_if_eq:enF
1198         {
1199             \tl_head:V
1200             \l_tmpa_tl
1201         }
1202         { s }

```

```

1203         {
1204             \msg_error:nnn
1205             { markdown }
1206             { malformed-name-for-clist-option }
1207             { #3 }
1208         }
1209     \tl_set:Nx
1210     \l_tmpa_tl
1211     {
1212         \tl_tail:V
1213         \l_tmpa_tl
1214     }
1215     \tl_reverse:N
1216     \l_tmpa_tl
1217     \tl_put_right:Nn
1218     \l_tmpa_tl
1219     {
1220         .code:n = {
1221             \@@_get_option_value:nN
1222             { #2 }
1223             \l_tmpa_tl
1224             \clist_set:NV
1225             \l_tmpa_clist
1226             { \l_tmpa_tl , { ##1 } }
1227             \@@_set_option_value:nV
1228             { #2 }
1229             \l_tmpa_clist
1230         }
1231     }
1232     \keys_define:nV
1233     { markdown/options }
1234     \l_tmpa_tl
1235 }
1236 }
1237 \cs_generate_variant:Nn
1238 \clist_set:Nn
1239 { NV }
1240 \cs_generate_variant:Nn
1241 \keys_define:nn
1242 { nV }
1243 \prg_generate_conditional_variant:Nnn
1244 \str_if_eq:nn
1245 { en }
1246 { p, F }
1247 \msg_new:nnn
1248 { markdown }
1249 { malformed-name-for-clist-option }

```

```

1250 {
1251   Clist~option~name~#1~does~not~end~with~-s.
1252 }

```

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

```

1253 \str_if_eq:VVT
1254   \c_@@_top_layer_tl
1255   \c_@@_option_layer_plain_tex_tl
1256   {
1257     \@@_define_option_commands_and_keyvals:
1258   }
1259 \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>`, optionally followed by `@<theme version>`, load a  $\text{\TeX}$  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 must be *qualified* and contain no underscores or at signs (@). Themes are inspired by the Beamer  $\text{\LaTeX}$  package, which provides similar functionality with its `\usetheme` macro [10, Section 15.1].

A theme name is qualified if and only if it contains at least one forward slash. Theme names must be qualified to minimize naming conflicts between different themes with a similar purpose. The preferred format of a theme name is `<theme author>/<theme purpose>/<private naming scheme>`, where the *private naming scheme* may contain additional forward slashes. For example, a theme by a user `witiko` for the MU theme of the Beamer document class may have the name `witiko/beamer/MU`.

Theme names are munged to allow structure inside theme names without dictating where the themes should be located inside the  $\text{\TeX}$  directory structure. For example, loading a theme named `witiko/beamer/MU` would load a  $\text{\TeX}$  document package named `markdownthemewitiko_beamer_MU.tex`.

If `@<theme version>` is specified after `<theme name>`, then the text *theme version* will be available in the macro `\markdownThemeVersion` when the theme is loaded. If `@<theme version>` is not specified, the macro `\markdownThemeVersion` will contain the text `latest` [11].

```

1260 \ExplSyntaxOn
1261 \keys_define:nn
1262   { markdown/options }

```

```

1263 {
1264   theme .code:n = {
1265     \@@_set_theme:n
1266     { #1 }
1267   },
1268   import .code:n = {
1269     \tl_set:Nn
1270     \l_tmpa_tl
1271     { #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.

```

1272     \tl_replace_all:NnV
1273     \l_tmpa_tl
1274     { / }
1275     \c_backslash_str
1276     \keys_set:nV
1277     { markdown/options/import }
1278     \l_tmpa_tl
1279   },
1280 }

```

To keep track of the current theme when themes are nested, we will maintain the stacks `\g_@@_theme_names_seq` and `\g_@@_theme_versions_seq` stack of theme names and versions, respectively. For convenience, the name of the current theme and version is also available in the macros `\g_@@_current_theme_tl` and `\markdownThemeVersion`, respectively.

```

1281 \seq_new:N
1282 \g_@@_theme_names_seq
1283 \seq_new:N
1284 \g_@@_theme_versions_seq
1285 \tl_new:N
1286 \g_@@_current_theme_tl
1287 \tl_gset:Nn
1288 \g_@@_current_theme_tl
1289 { }
1290 \seq_gput_right:NV
1291 \g_@@_theme_names_seq
1292 \g_@@_current_theme_tl
1293 \cs_new:Npn
1294 \markdownThemeVersion
1295 { }
1296 \seq_gput_right:NV
1297 \g_@@_theme_versions_seq

```

```

1298 \g_@@_current_theme_tl
1299 \cs_new:Nn
1300 \@@_set_theme:n
1301 {

```

First, we validate the theme name.

```

1302 \str_if_in:nnF
1303 { #1 }
1304 { / }
1305 {
1306 \msg_error:nnn
1307 { markdown }
1308 { unqualified-theme-name }
1309 { #1 }
1310 }
1311 \str_if_in:nnT
1312 { #1 }
1313 { _ }
1314 {
1315 \msg_error:nnn
1316 { markdown }
1317 { underscores-in-theme-name }
1318 { #1 }
1319 }

```

Next, we extract the theme version.

```

1320 \str_if_in:nnTF
1321 { #1 }
1322 { @ }
1323 {
1324 \regex_extract_once:nnN
1325 { (.*?) @ (.*?) }
1326 { #1 }
1327 \l_tmpa_seq
1328 \seq_gpop_left:NN
1329 \l_tmpa_seq
1330 \l_tmpa_tl
1331 \seq_gpop_left:NN
1332 \l_tmpa_seq
1333 \l_tmpa_tl
1334 \tl_gset:NV
1335 \g_@@_current_theme_tl
1336 \l_tmpa_tl
1337 \seq_gpop_left:NN
1338 \l_tmpa_seq
1339 \l_tmpa_tl
1340 \cs_gset:Npe
1341 \markdownThemeVersion

```

```

1342         {
1343             \tl_use:N
1344             \l_tmpa_tl
1345         }
1346     }
1347     {
1348         \tl_gset:Nn
1349         \g_@@_current_theme_tl
1350         { #1 }
1351         \cs_gset:Npn
1352         \markdownThemeVersion
1353         { latest }
1354     }

```

Next, we munge the theme name.

```

1355     \str_set:NV
1356     \l_tmpa_str
1357     \g_@@_current_theme_tl
1358     \str_replace_all:Nnn
1359     \l_tmpa_str
1360     { / }
1361     { _ }

```

Finally, we load the theme. Before loading the theme, we push down the current name and version of the theme on the stack.

```

1362     \tl_set:NV
1363     \l_tmpa_tl
1364     \g_@@_current_theme_tl
1365     \tl_put_right:Nn
1366     \g_@@_current_theme_tl
1367     { / }
1368     \seq_gput_right:NV
1369     \g_@@_theme_names_seq
1370     \g_@@_current_theme_tl
1371     \seq_gput_right:NV
1372     \g_@@_theme_versions_seq
1373     \markdownThemeVersion
1374     \@@_load_theme:VeV
1375     \l_tmpa_tl
1376     { \markdownThemeVersion }
1377     \l_tmpa_str

```

After the theme has been loaded, we recover the name and version of the previous theme from the stack.

```

1378     \seq_gpop_right:NN
1379     \g_@@_theme_names_seq
1380     \l_tmpa_tl
1381     \seq_get_right:NN

```



```

1382     \g_@@_theme_names_seq
1383     \l_tmpa_tl
1384     \tl_gset:NV
1385     \g_@@_current_theme_tl
1386     \l_tmpa_tl
1387     \seq_gpop_right:NN
1388     \g_@@_theme_versions_seq
1389     \l_tmpa_tl
1390     \seq_get_right:NN
1391     \g_@@_theme_versions_seq
1392     \l_tmpa_tl
1393     \cs_gset:Npe
1394     \markdownThemeVersion
1395     {
1396         \tl_use:N
1397         \l_tmpa_tl
1398     }
1399 }
1400 \msg_new:nnnn
1401 { markdown }
1402 { unqualified-theme-name }
1403 { Won't load theme with unqualified name~#1 }
1404 { Theme names must contain at least one forward slash }
1405 \msg_new:nnnn
1406 { markdown }
1407 { underscores-in-theme-name }
1408 { Won't load theme with an underscore in its name~#1 }
1409 { Theme names must not contain underscores in their names }
1410 \cs_generate_variant:Nn
1411 \tl_replace_all:Nnn
1412 { NnV }
1413 \cs_generate_variant:Nn
1414 \cs_gset:Npn
1415 { Npe }

```

We also define the prop `\g_@@_plain_tex_built_in_themes_prop` that contains the code of built-in themes. This is a packaging optimization, so that built-in themes does not need to be distributed in many small files.

```

1416 \prop_new:N
1417 \g_@@_plain_tex_built_in_themes_prop

```

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

**witiko/diagrams** A theme that typesets fenced code blocks with the infostrings `dot`, `mermaid`, and `plantuml` as figures with diagrams produced with the command `dot` from Graphviz tools, the command `mmdc` from the npm package `@mermaid-js/mermaid-cli`, and the command `plantuml` from the package PlantUML, respectively. The key-value attribute `caption` can be used to

specify the caption of the figure. The remaining attributes are treated as image attributes.

```
\documentclass{article}
\usepackage[import=witiko/diagrams@v2, relativeReferences]{markdown}
\begin{document}
\begin{markdown}
``` dot {caption="An example directed graph" width=12cm #dot}
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"];
}
```

``` mermaid {caption="An example mindmap" width=9cm #mermaid}
mindmap
  root )base-idea(
    sub<br/>idea 1
      ((?))
    sub<br/>idea 2
      ((?))
    sub<br/>idea 3
      ((?))
    sub<br/>idea 4
      ((?))
```

```

---

``` plantuml {caption="An example UML sequence diagram" width=7cm #plantuml}
@startuml
' Define participants (actors)
participant "Client" as C
participant "Server" as S
participant "Database" as DB

' Diagram title
title Simple Request-Response Flow

' Messages
C -> S: Send Request
note over S: Process request

alt Request is valid
    S -> DB: Query Data
    DB -> S: Return Data
    S -> C: Respond with Data
else Request is invalid
    S -> C: Return Error
end
@enduml
---

See the diagrams in figures <#dot>, <#mermaid>, and <#plantuml>.
\end{markdown}
\end{document}

```

Typesetting the above document produces the output shown in figures 4, 5, and 6.

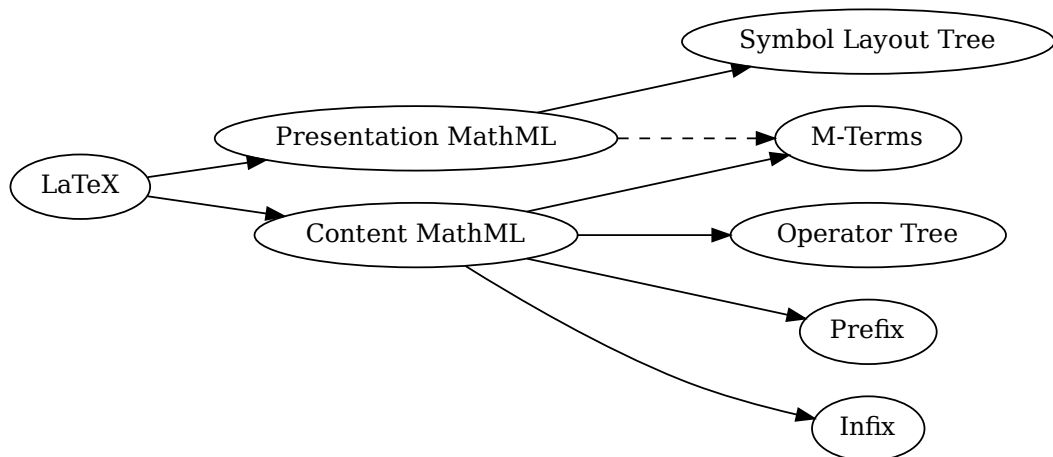
The theme requires a Unix-like operating system with GNU Diffutils, Graphviz, the npm package [@mermaid-js/mermaid-cli](#), and PlantUML installed. All these packages are already included in the Docker image [witiko/markdown](#); consult [Dockerfile](#) to see how they are installed. The theme also requires shell access unless the [frozenCache](#) plain T<sub>E</sub>X option is enabled.

**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}

```



**Figure 4: An example directed graph**

```

\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 7. 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.

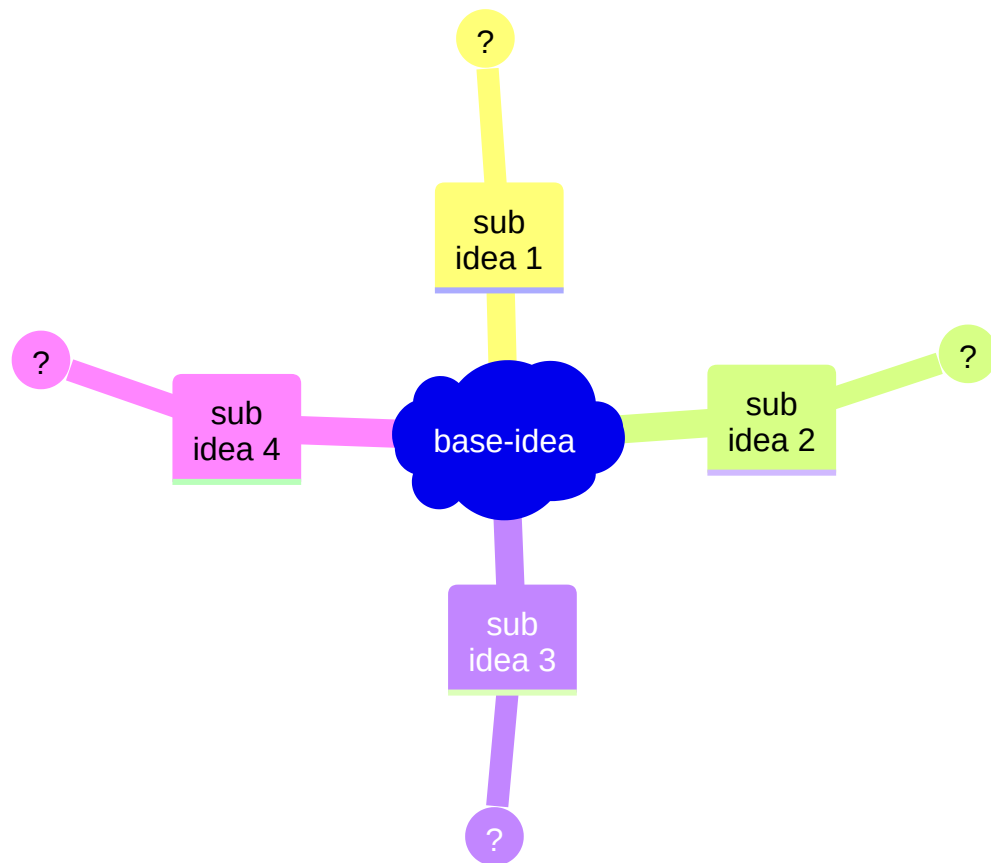
**witiko/tilde** A theme that makes tilde (~) always typeset the non-breaking space even when the `hybrid` Lua option is disabled.

```

\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”.



**Figure 5: An example mindmap**

## Simple Request-Response Flow

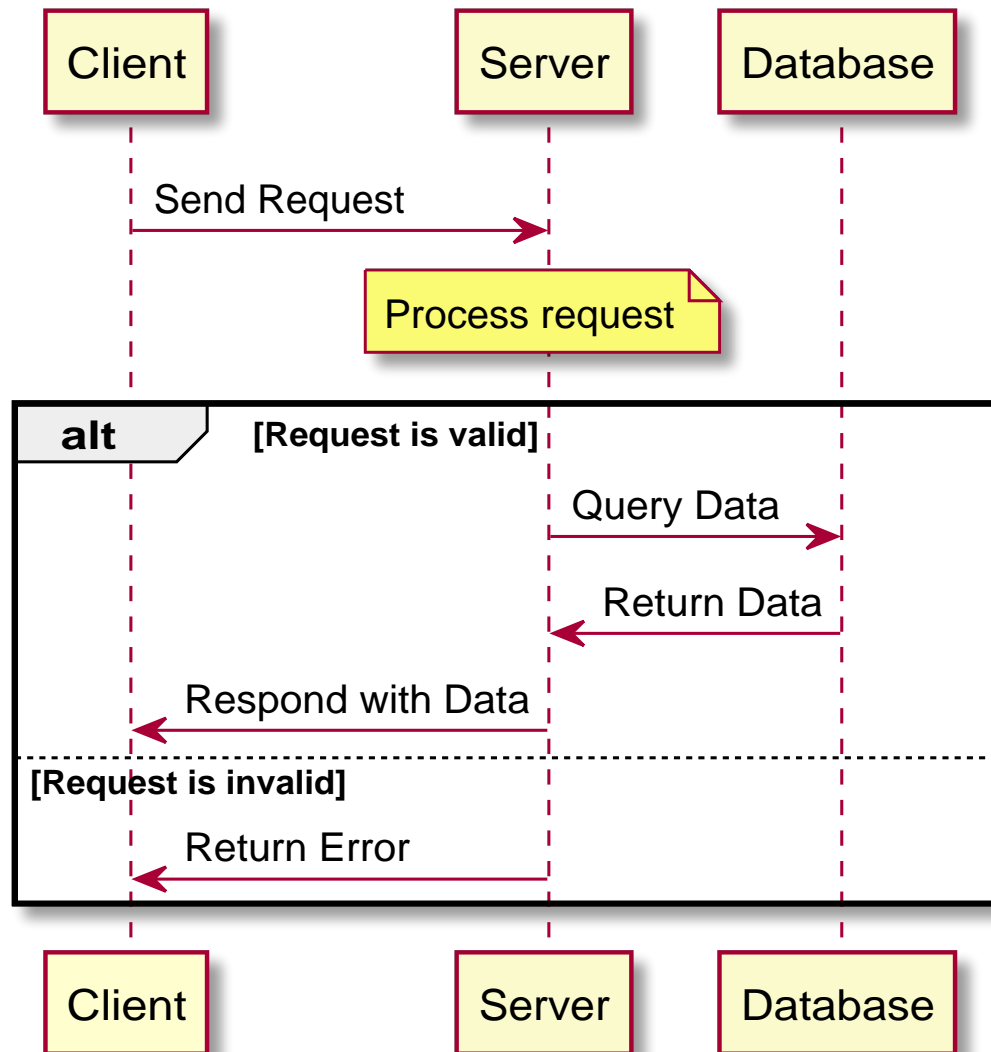


Figure 6: An example UML sequence diagram

```

\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 7: The banner of the Markdown package**

**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.

```

1418 \prop_new:N
1419   \g_@@_snippets_prop
1420 \cs_new:Nn
1421   \@@_setup_snippet:nn
1422   {
1423     \tl_if_empty:nT
1424       { #1 }
1425     {
1426       \msg_error:nnn
1427         { markdown }

```

```

1428         { empty-snippet-name }
1429         { #1 }
1430     }
1431     \tl_set:NV
1432     \l_tmpa_tl
1433     \g_@@_current_theme_tl
1434     \tl_put_right:Nn
1435     \l_tmpa_tl
1436     { #1 }
1437     \@@_if_snippet_exists:nT
1438     { #1 }
1439     {
1440         \msg_warning:nnV
1441         { markdown }
1442         { redefined-snippet }
1443         \l_tmpa_tl
1444     }
1445     \keys_precompile:nnN
1446     { markdown/options }
1447     { #2 }
1448     \l_tmpb_tl
1449     \prop_gput:NVV
1450     \g_@@_snippets_prop
1451     \l_tmpa_tl
1452     \l_tmpb_tl
1453 }
1454 \cs_gset_eq:NN
1455 \markdownSetupSnippet
1456 \@@_setup_snippet:nn
1457 \msg_new:nnnn
1458 { markdown }
1459 { empty-snippet-name }
1460 { Empty~snippet~name~#1 }
1461 { Pick~a~non-empty~name~for~your~snippet }
1462 \msg_new:nnn
1463 { markdown }
1464 { redefined-snippet }
1465 { Redefined~snippet~#1 }

```

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

```

1466 \tl_new:N
1467 \l_@@_current_snippet_tl
1468 \prg_new_conditional:Nnn
1469 \@@_if_snippet_exists:n
1470 { TF, T }
1471 {
1472     \tl_set:NV

```



```

1473     \l_@@_current_snippet_tl
1474     \g_@@_current_theme_tl
1475     \tl_put_right:Nn
1476     \l_@@_current_snippet_tl
1477     { #1 }
1478     \prop_if_in:NVTF
1479     \g_@@_snippets_prop
1480     \l_@@_current_snippet_tl
1481     { \prg_return_true: }
1482     { \prg_return_false: }
1483   }
1484   \cs_gset_eq:NN
1485   \markdownIfSnippetExists
1486   \@@_if_snippet_exists:nTF

```

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

```

1487   \keys_define:nn
1488   { markdown/options }
1489   {
1490     snippet .code:n = {
1491       \tl_set:NV
1492       \l_tmpa_tl
1493       \g_@@_current_theme_tl
1494       \tl_put_right:Nn
1495       \l_tmpa_tl
1496       { #1 }
1497       \@@_if_snippet_exists:nTF
1498       { #1 }
1499       {
1500         \prop_get:NVN
1501         \g_@@_snippets_prop
1502         \l_tmpa_tl
1503         \l_tmpb_tl
1504         \tl_use:N
1505         \l_tmpb_tl
1506       }
1507       {
1508         \msg_error:nnV
1509         { markdown }
1510         { undefined-snippet }
1511         \l_tmpa_tl
1512       }
1513     }
1514   }
1515   \msg_new:nnn
1516   { markdown }
1517   { undefined-snippet }
1518   { Can't~invoke~undefined~snippet~#1 }

```

1519 \ExplSyntaxOff

Here is how we can use snippets to store options and invoke them later in L<sup>A</sup>T<sub>E</sub>X:

```
\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 `jdoe/lists` theme, we could import the `jdoe/lists` theme and use the qualified name `jdoe/lists/romanNumerals` to invoke the snippet:

```
\markdownSetup{import=jdoe/lists}
\begin{markdown}[snippet=jdoe/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` L<sup>A</sup>T<sub>E</sub>X option that allows us to import the `romanNumerals` snippet to the current namespace for easier access:

```

\markdownSetup{
  import = {
    jdoe/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 `jdoe/lists` theme. For example, we can make the snippet `jdoe/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,
},
}

```

```

1520 \ExplSyntaxOn
1521 \tl_new:N
1522   \l_@@_import_current_theme_tl
1523 \keys_define:nn
1524   { markdown/options/import }
1525   {

```

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

```

1526     unknown .default:n = {},
1527     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.

```

1528     \tl_set_eq:NN
1529       \l_@@_import_current_theme_tl
1530       \l_keys_key_str
1531     \tl_replace_all:NVN
1532       \l_@@_import_current_theme_tl
1533       \c_backslash_str
1534       { / }

```

Here, we import the snippets.

```

1535     \clist_map_inline:nn
1536       { #1 }
1537       {
1538         \regex_extract_once:nnNTF
1539           { ^(.*)\s+as\s+(.*)$ }
1540           { ##1 }
1541         \l_tmpa_seq
1542         {
1543           \seq_pop:NN
1544             \l_tmpa_seq
1545             \l_tmpa_tl

```

```

1546         \seq_pop:NN
1547         \l_tmpa_seq
1548         \l_tmpa_tl
1549         \seq_pop:NN
1550         \l_tmpa_seq
1551         \l_tmpb_tl
1552     }
1553     {
1554         \tl_set:Nn
1555         \l_tmpa_tl
1556         { ##1 }
1557         \tl_set:Nn
1558         \l_tmpb_tl
1559         { ##1 }
1560     }
1561     \tl_put_left:Nn
1562     \l_tmpa_tl
1563     { / }
1564     \tl_put_left:NV
1565     \l_tmpa_tl
1566     \l_@@_import_current_theme_tl
1567     \@@_setup_snippet:Vx
1568     \l_tmpb_tl
1569     { snippet = { \l_tmpa_tl } }
1570 }

```

Here, we load the theme.

```

1571     \@@_set_theme:V
1572     \l_@@_import_current_theme_tl
1573 },
1574 }
1575 \cs_generate_variant:Nn
1576 \tl_replace_all:Nnn
1577 { NVn }
1578 \cs_generate_variant:Nn
1579 \@@_set_theme:n
1580 { V }
1581 \cs_generate_variant:Nn
1582 \@@_setup_snippet:nn
1583 { 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.

```
1584 \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.

```
1585 \prop_new:N \g_@@_renderer_arities_prop
```

```
1586 \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.

```
1587 \ExplSyntaxOn
1588 \cs_gset_protected:Npn
1589   \markdownRendererAttributeIdentifier
1590   {
1591     \markdownRendererAttributeIdentifierPrototype
1592   }
1593 \seq_gput_right:Nn
1594   \g_@@_renderers_seq
1595   { attributeIdentifier }
1596 \prop_gput:Nnn
1597   \g_@@_renderer_arities_prop
1598   { attributeIdentifier }
1599   { 1 }
```

```

1600 \cs_gset_protected:Npn
1601   \markdownRendererAttributeClassName
1602   {
1603     \markdownRendererAttributeClassNamePrototype
1604   }
1605 \seq_gput_right:Nn
1606   \g_@@_renderers_seq
1607   { attributeClassName }
1608 \prop_gput:Nnn
1609   \g_@@_renderer_arities_prop
1610   { attributeClassName }
1611   { 1 }
1612 \cs_gset_protected:Npn
1613   \markdownRendererAttributeKeyValue
1614   {
1615     \markdownRendererAttributeKeyValuePrototype
1616   }
1617 \seq_gput_right:Nn
1618   \g_@@_renderers_seq
1619   { attributeKeyValue }
1620 \prop_gput:Nnn
1621   \g_@@_renderer_arities_prop
1622   { attributeKeyValue }
1623   { 2 }
1624 \ExplSyntaxOff

```

### 2.2.5.2 Block Quote Renderers

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

```

1625 \ExplSyntaxOn
1626 \cs_gset_protected:Npn
1627   \markdownRendererBlockQuoteBegin
1628   {
1629     \markdownRendererBlockQuoteBeginPrototype
1630   }
1631 \seq_gput_right:Nn
1632   \g_@@_renderers_seq
1633   { blockQuoteBegin }
1634 \prop_gput:Nnn
1635   \g_@@_renderer_arities_prop
1636   { blockQuoteBegin }
1637   { 0 }
1638 \ExplSyntaxOff

```

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

```

1639 \ExplSyntaxOn
1640 \cs_gset_protected:Npn
1641   \markdownRendererBlockQuoteEnd
1642   {
1643     \markdownRendererBlockQuoteEndPrototype
1644   }
1645 \seq_gput_right:Nn
1646   \g_@@_renderers_seq
1647   { blockQuoteEnd }
1648 \prop_gput:Nnn
1649   \g_@@_renderer_arities_prop
1650   { blockQuoteEnd }
1651   { 0 }
1652 \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.

```

1653 \ExplSyntaxOn
1654 \cs_gset_protected:Npn
1655   \markdownRendererBracketedSpanAttributeContextBegin
1656   {
1657     \markdownRendererBracketedSpanAttributeContextBeginPrototype
1658   }
1659 \seq_gput_right:Nn
1660   \g_@@_renderers_seq
1661   { bracketedSpanAttributeContextBegin }
1662 \prop_gput:Nnn
1663   \g_@@_renderer_arities_prop
1664   { bracketedSpanAttributeContextBegin }
1665   { 0 }
1666 \cs_gset_protected:Npn
1667   \markdownRendererBracketedSpanAttributeContextEnd
1668   {
1669     \markdownRendererBracketedSpanAttributeContextEndPrototype
1670   }
1671 \seq_gput_right:Nn
1672   \g_@@_renderers_seq
1673   { bracketedSpanAttributeContextEnd }
1674 \prop_gput:Nnn
1675   \g_@@_renderer_arities_prop
1676   { bracketedSpanAttributeContextEnd }
1677   { 0 }

```



1678 \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.

```
1679 \ExplSyntaxOn
1680 \cs_gset_protected:Npn
1681   \markdownRendererUlBegin
1682   {
1683     \markdownRendererUlBeginPrototype
1684   }
1685 \seq_gput_right:Nn
1686   \g_@@_renderers_seq
1687   { ulBegin }
1688 \prop_gput:Nnn
1689   \g_@@_renderer_arities_prop
1690   { ulBegin }
1691   { 0 }
1692 \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.

```
1693 \ExplSyntaxOn
1694 \cs_gset_protected:Npn
1695   \markdownRendererUlBeginTight
1696   {
1697     \markdownRendererUlBeginTightPrototype
1698   }
1699 \seq_gput_right:Nn
1700   \g_@@_renderers_seq
1701   { ulBeginTight }
1702 \prop_gput:Nnn
1703   \g_@@_renderer_arities_prop
1704   { ulBeginTight }
1705   { 0 }
1706 \ExplSyntaxOff
```

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

```
1707 \ExplSyntaxOn
1708 \cs_gset_protected:Npn
1709   \markdownRendererUlItem
```

```

1710 {
1711     \markdownRendererUListItemPrototype
1712 }
1713 \seq_gput_right:Nn
1714 \g_@@_renderers_seq
1715 { ulItem }
1716 \prop_gput:Nnn
1717 \g_@@_renderer_arities_prop
1718 { ulItem }
1719 { 0 }
1720 \ExplSyntaxOff

```

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

```

1721 \ExplSyntaxOn
1722 \cs_gset_protected:Npn
1723 \markdownRendererUListItemEnd
1724 {
1725     \markdownRendererUListItemEndPrototype
1726 }
1727 \seq_gput_right:Nn
1728 \g_@@_renderers_seq
1729 { ulItemEnd }
1730 \prop_gput:Nnn
1731 \g_@@_renderer_arities_prop
1732 { ulItemEnd }
1733 { 0 }
1734 \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.

```

1735 \ExplSyntaxOn
1736 \cs_gset_protected:Npn
1737 \markdownRendererUListEnd
1738 {
1739     \markdownRendererUListEndPrototype
1740 }
1741 \seq_gput_right:Nn
1742 \g_@@_renderers_seq
1743 { ulEnd }
1744 \prop_gput:Nnn
1745 \g_@@_renderer_arities_prop
1746 { ulEnd }
1747 { 0 }
1748 \ExplSyntaxOff

```

The `\markdownRendererUlEndTight` 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.

```

1749 \ExplSyntaxOn
1750 \cs_gset_protected:Npn
1751   \markdownRendererUlEndTight
1752   {
1753     \markdownRendererUlEndTightPrototype
1754   }
1755 \seq_gput_right:Nn
1756   \g_@@_renderers_seq
1757   { ulEndTight }
1758 \prop_gput:Nnn
1759   \g_@@_renderer_arities_prop
1760   { ulEndTight }
1761   { 0 }
1762 \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.

```

1763 \ExplSyntaxOn
1764 \cs_gset_protected:Npn
1765   \markdownRendererCite
1766   {
1767     \markdownRendererCitePrototype
1768   }
1769 \seq_gput_right:Nn
1770   \g_@@_renderers_seq
1771   { cite }
1772 \prop_gput:Nnn
1773   \g_@@_renderer_arities_prop
1774   { cite }
1775   { 1 }
1776 \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.

```

1777 \ExplSyntaxOn
1778 \cs_gset_protected:Npn
1779   \markdownRendererTextCite
1780   {
1781     \markdownRendererTextCitePrototype
1782   }
1783 \seq_gput_right:Nn
1784   \g_@@_renderers_seq
1785   { textCite }
1786 \prop_gput:Nnn
1787   \g_@@_renderer_arities_prop
1788   { textCite }
1789   { 1 }
1790 \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.

```

1791 \ExplSyntaxOn
1792 \cs_gset_protected:Npn
1793   \markdownRendererInputVerbatim
1794   {
1795     \markdownRendererInputVerbatimPrototype
1796   }
1797 \seq_gput_right:Nn
1798   \g_@@_renderers_seq
1799   { inputVerbatim }
1800 \prop_gput:Nnn
1801   \g_@@_renderer_arities_prop
1802   { inputVerbatim }
1803   { 1 }
1804 \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.

```

1805 \ExplSyntaxOn
1806 \cs_gset_protected:Npn
1807   \markdownRendererInputFencedCode
1808   {
1809     \markdownRendererInputFencedCodePrototype
1810   }

```

```

1811 \seq_gput_right:Nn
1812 \g_@@_renderers_seq
1813 { inputFencedCode }
1814 \prop_gput:Nnn
1815 \g_@@_renderer_arities_prop
1816 { inputFencedCode }
1817 { 3 }
1818 \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.

```

1819 \ExplSyntaxOn
1820 \cs_gset_protected:Npn
1821 \markdownRendererCodeSpan
1822 {
1823   \markdownRendererCodeSpanPrototype
1824 }
1825 \seq_gput_right:Nn
1826 \g_@@_renderers_seq
1827 { codeSpan }
1828 \prop_gput:Nnn
1829 \g_@@_renderer_arities_prop
1830 { codeSpan }
1831 { 1 }
1832 \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.

```

1833 \ExplSyntaxOn
1834 \cs_gset_protected:Npn
1835 \markdownRendererCodeSpanAttributeContextBegin
1836 {
1837   \markdownRendererCodeSpanAttributeContextBeginPrototype
1838 }
1839 \seq_gput_right:Nn
1840 \g_@@_renderers_seq
1841 { codeSpanAttributeContextBegin }
1842 \prop_gput:Nnn
1843 \g_@@_renderer_arities_prop

```

```

1844 { codeSpanAttributeContextBegin }
1845 { 0 }
1846 \cs_gset_protected:Npn
1847 \markdownRendererCodeSpanAttributeContextEnd
1848 {
1849   \markdownRendererCodeSpanAttributeContextEndPrototype
1850 }
1851 \seq_gput_right:Nn
1852 \g_@@_renderers_seq
1853 { codeSpanAttributeContextEnd }
1854 \prop_gput:Nnn
1855 \g_@@_renderer_arities_prop
1856 { codeSpanAttributeContextEnd }
1857 { 0 }
1858 \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.

```

1859 \ExplSyntaxOn
1860 \cs_gset_protected:Npn
1861 \markdownRendererContentBlock
1862 {
1863   \markdownRendererContentBlockPrototype
1864 }
1865 \seq_gput_right:Nn
1866 \g_@@_renderers_seq
1867 { contentBlock }
1868 \prop_gput:Nnn
1869 \g_@@_renderer_arities_prop
1870 { contentBlock }
1871 { 4 }
1872 \ExplSyntaxOff

```

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

```

1873 \ExplSyntaxOn
1874 \cs_gset_protected:Npn
1875 \markdownRendererContentBlockOnlineImage
1876 {
1877   \markdownRendererContentBlockOnlineImagePrototype
1878 }

```

```

1879 \seq_gput_right:Nn
1880   \g_@@_renderers_seq
1881   { contentBlockOnlineImage }
1882 \prop_gput:Nnn
1883   \g_@@_renderer_arities_prop
1884   { contentBlockOnlineImage }
1885   { 4 }
1886 \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>34</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 [5] is a good starting point.

```

1887 \ExplSyntaxOn
1888 \cs_gset_protected:Npn
1889   \markdownRendererContentBlockCode
1890   {
1891     \markdownRendererContentBlockCodePrototype
1892   }
1893 \seq_gput_right:Nn
1894   \g_@@_renderers_seq
1895   { contentBlockCode }
1896 \prop_gput:Nnn
1897   \g_@@_renderer_arities_prop
1898   { contentBlockCode }
1899   { 5 }
1900 \ExplSyntaxOff

```

#### 2.2.5.10 Definition List Renderers

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

---

<sup>34</sup> Filenames other than `markdown-languages.json` may be specified using the `contentBlocksLanguageMap` Lua option.

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.

```

1901 \ExplSyntaxOn
1902 \cs_gset_protected:Npn
1903   \markdownRendererDlBegin
1904   {
1905     \markdownRendererDlBeginPrototype
1906   }
1907 \seq_gput_right:Nn
1908   \g_@@_renderers_seq
1909   { dlBegin }
1910 \prop_gput:Nnn
1911   \g_@@_renderer_arities_prop
1912   { dlBegin }
1913   { 0 }
1914 \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.

```

1915 \ExplSyntaxOn
1916 \cs_gset_protected:Npn
1917   \markdownRendererDlBeginTight
1918   {
1919     \markdownRendererDlBeginTightPrototype
1920   }
1921 \seq_gput_right:Nn
1922   \g_@@_renderers_seq
1923   { dlBeginTight }
1924 \prop_gput:Nnn
1925   \g_@@_renderer_arities_prop
1926   { dlBeginTight }
1927   { 0 }
1928 \ExplSyntaxOff

```

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

```

1929 \ExplSyntaxOn
1930 \cs_gset_protected:Npn
1931   \markdownRendererDlItem
1932   {
1933     \markdownRendererDlItemPrototype
1934   }
1935 \seq_gput_right:Nn

```



```

1936 \g_@@_renderers_seq
1937 { dlItem }
1938 \prop_gput:Nnn
1939 \g_@@_renderer_arities_prop
1940 { dlItem }
1941 { 1 }
1942 \ExplSyntaxOff

```

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

```

1943 \ExplSyntaxOn
1944 \cs_gset_protected:Npn
1945 \markdownRendererDlItemEnd
1946 {
1947   \markdownRendererDlItemEndPrototype
1948 }
1949 \seq_gput_right:Nn
1950 \g_@@_renderers_seq
1951 { dlItemEnd }
1952 \prop_gput:Nnn
1953 \g_@@_renderer_arities_prop
1954 { dlItemEnd }
1955 { 0 }
1956 \ExplSyntaxOff

```

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

```

1957 \ExplSyntaxOn
1958 \cs_gset_protected:Npn
1959 \markdownRendererDlDefinitionBegin
1960 {
1961   \markdownRendererDlDefinitionBeginPrototype
1962 }
1963 \seq_gput_right:Nn
1964 \g_@@_renderers_seq
1965 { dlDefinitionBegin }
1966 \prop_gput:Nnn
1967 \g_@@_renderer_arities_prop
1968 { dlDefinitionBegin }
1969 { 0 }
1970 \ExplSyntaxOff

```

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

```

1971 \ExplSyntaxOn
1972 \cs_gset_protected:Npn
1973 \markdownRendererDlDefinitionEnd

```

```

1974 {
1975   \markdownRendererDlDefinitionEndPrototype
1976 }
1977 \seq_gput_right:Nn
1978 \g_@@_renderers_seq
1979 { dlDefinitionEnd }
1980 \prop_gput:Nnn
1981 \g_@@_renderer_arities_prop
1982 { dlDefinitionEnd }
1983 { 0 }
1984 \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.

```

1985 \ExplSyntaxOn
1986 \cs_gset_protected:Npn
1987   \markdownRendererDlEnd
1988   {
1989     \markdownRendererDlEndPrototype
1990   }
1991 \seq_gput_right:Nn
1992 \g_@@_renderers_seq
1993 { dlEnd }
1994 \prop_gput:Nnn
1995 \g_@@_renderer_arities_prop
1996 { dlEnd }
1997 { 0 }
1998 \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.

```

1999 \ExplSyntaxOn
2000 \cs_gset_protected:Npn
2001   \markdownRendererDlEndTight
2002   {
2003     \markdownRendererDlEndTightPrototype
2004   }
2005 \seq_gput_right:Nn
2006 \g_@@_renderers_seq
2007 { dlEndTight }
2008 \prop_gput:Nnn
2009 \g_@@_renderer_arities_prop
2010 { dlEndTight }

```

```

2011 { 0 }
2012 \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.

```

2013 \ExplSyntaxOn
2014 \cs_gset_protected:Npn
2015   \markdownRendererEllipsis
2016   {
2017     \markdownRendererEllipsisPrototype
2018   }
2019 \seq_gput_right:Nn
2020   \g_@@_renderers_seq
2021   { ellipsis }
2022 \prop_gput:Nnn
2023   \g_@@_renderer_arities_prop
2024   { ellipsis }
2025   { 0 }
2026 \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.

```

2027 \ExplSyntaxOn
2028 \cs_gset_protected:Npn
2029   \markdownRendererEmphasis
2030   {
2031     \markdownRendererEmphasisPrototype
2032   }
2033 \seq_gput_right:Nn
2034   \g_@@_renderers_seq
2035   { emphasis }
2036 \prop_gput:Nnn
2037   \g_@@_renderer_arities_prop
2038   { emphasis }
2039   { 1 }
2040 \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.

```

2041 \ExplSyntaxOn
2042 \cs_gset_protected:Npn
2043   \markdownRendererStrongEmphasis
2044   {
2045     \markdownRendererStrongEmphasisPrototype
2046   }
2047 \seq_gput_right:Nn
2048   \g_@@_renderers_seq
2049   { strongEmphasis }
2050 \prop_gput:Nnn
2051   \g_@@_renderer_arities_prop
2052   { strongEmphasis }
2053   { 1 }
2054 \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.

```

2055 \ExplSyntaxOn
2056 \cs_gset_protected:Npn
2057   \markdownRendererFencedCodeAttributeContextBegin
2058   {
2059     \markdownRendererFencedCodeAttributeContextBeginPrototype
2060   }
2061 \seq_gput_right:Nn
2062   \g_@@_renderers_seq
2063   { fencedCodeAttributeContextBegin }
2064 \prop_gput:Nnn
2065   \g_@@_renderer_arities_prop
2066   { fencedCodeAttributeContextBegin }
2067   { 0 }
2068 \cs_gset_protected:Npn
2069   \markdownRendererFencedCodeAttributeContextEnd
2070   {
2071     \markdownRendererFencedCodeAttributeContextEndPrototype
2072   }
2073 \seq_gput_right:Nn
2074   \g_@@_renderers_seq
2075   { fencedCodeAttributeContextEnd }
2076 \prop_gput:Nnn
2077   \g_@@_renderer_arities_prop
2078   { fencedCodeAttributeContextEnd }
2079   { 0 }

```

2080 \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.

```
2081 \ExplSyntaxOn
2082 \cs_gset_protected:Npn
2083   \markdownRendererFencedDivAttributeContextBegin
2084   {
2085     \markdownRendererFencedDivAttributeContextBeginPrototype
2086   }
2087 \seq_gput_right:Nn
2088   \g_@@_renderers_seq
2089   { fencedDivAttributeContextBegin }
2090 \prop_gput:Nnn
2091   \g_@@_renderer_arities_prop
2092   { fencedDivAttributeContextBegin }
2093   { 0 }
2094 \cs_gset_protected:Npn
2095   \markdownRendererFencedDivAttributeContextEnd
2096   {
2097     \markdownRendererFencedDivAttributeContextEndPrototype
2098   }
2099 \seq_gput_right:Nn
2100   \g_@@_renderers_seq
2101   { fencedDivAttributeContextEnd }
2102 \prop_gput:Nnn
2103   \g_@@_renderer_arities_prop
2104   { fencedDivAttributeContextEnd }
2105   { 0 }
2106 \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.

```
2107 \ExplSyntaxOn
2108 \cs_gset_protected:Npn
2109   \markdownRendererHeaderAttributeContextBegin
2110   {
```

```

2111     \markdownRendererHeaderAttributeContextBeginPrototype
2112   }
2113   \seq_gput_right:Nn
2114     \g_@@_renderers_seq
2115     { headerAttributeContextBegin }
2116   \prop_gput:Nnn
2117     \g_@@_renderer_arities_prop
2118     { headerAttributeContextBegin }
2119     { 0 }
2120   \cs_gset_protected:Npn
2121     \markdownRendererHeaderAttributeContextEnd
2122     {
2123       \markdownRendererHeaderAttributeContextEndPrototype
2124     }
2125   \seq_gput_right:Nn
2126     \g_@@_renderers_seq
2127     { headerAttributeContextEnd }
2128   \prop_gput:Nnn
2129     \g_@@_renderer_arities_prop
2130     { headerAttributeContextEnd }
2131     { 0 }
2132   \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.

```

2133   \ExplSyntaxOn
2134   \cs_gset_protected:Npn
2135     \markdownRendererHeadingOne
2136     {
2137       \markdownRendererHeadingOnePrototype
2138     }
2139   \seq_gput_right:Nn
2140     \g_@@_renderers_seq
2141     { headingOne }
2142   \prop_gput:Nnn
2143     \g_@@_renderer_arities_prop
2144     { headingOne }
2145     { 1 }
2146   \ExplSyntaxOff

```

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

```

2147   \ExplSyntaxOn
2148   \cs_gset_protected:Npn
2149     \markdownRendererHeadingTwo

```

```

2150 {
2151   \markdownRendererHeadingTwoPrototype
2152 }
2153 \seq_gput_right:Nn
2154 \g_@@_renderers_seq
2155 { headingTwo }
2156 \prop_gput:Nnn
2157 \g_@@_renderer_arities_prop
2158 { headingTwo }
2159 { 1 }
2160 \ExplSyntaxOff

```

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

```

2161 \ExplSyntaxOn
2162 \cs_gset_protected:Npn
2163   \markdownRendererHeadingThree
2164   {
2165     \markdownRendererHeadingThreePrototype
2166   }
2167 \seq_gput_right:Nn
2168 \g_@@_renderers_seq
2169 { headingThree }
2170 \prop_gput:Nnn
2171 \g_@@_renderer_arities_prop
2172 { headingThree }
2173 { 1 }
2174 \ExplSyntaxOff

```

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

```

2175 \ExplSyntaxOn
2176 \cs_gset_protected:Npn
2177   \markdownRendererHeadingFour
2178   {
2179     \markdownRendererHeadingFourPrototype
2180   }
2181 \seq_gput_right:Nn
2182 \g_@@_renderers_seq
2183 { headingFour }
2184 \prop_gput:Nnn
2185 \g_@@_renderer_arities_prop
2186 { headingFour }
2187 { 1 }
2188 \ExplSyntaxOff

```

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

```

2189 \ExplSyntaxOn
2190 \cs_gset_protected:Npn
2191   \markdownRendererHeadingFive
2192   {
2193     \markdownRendererHeadingFivePrototype
2194   }
2195 \seq_gput_right:Nn
2196   \g_@@_renderers_seq
2197   { headingFive }
2198 \prop_gput:Nnn
2199   \g_@@_renderer_arities_prop
2200   { headingFive }
2201   { 1 }
2202 \ExplSyntaxOff

```

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

```

2203 \ExplSyntaxOn
2204 \cs_gset_protected:Npn
2205   \markdownRendererHeadingSix
2206   {
2207     \markdownRendererHeadingSixPrototype
2208   }
2209 \seq_gput_right:Nn
2210   \g_@@_renderers_seq
2211   { headingSix }
2212 \prop_gput:Nnn
2213   \g_@@_renderer_arities_prop
2214   { headingSix }
2215   { 1 }
2216 \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.

```

2217 \ExplSyntaxOn
2218 \cs_gset_protected:Npn
2219   \markdownRendererInlineHtmlComment
2220   {
2221     \markdownRendererInlineHtmlCommentPrototype
2222   }

```



```

2223 \seq_gput_right:Nn
2224   \g_@@_renderers_seq
2225   { inlineHtmlComment }
2226 \prop_gput:Nnn
2227   \g_@@_renderer_arities_prop
2228   { inlineHtmlComment }
2229   { 1 }
2230 \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.

```

2231 \ExplSyntaxOn
2232 \cs_gset_protected:Npn
2233   \markdownRendererInlineHtmlTag
2234   {
2235     \markdownRendererInlineHtmlTagPrototype
2236   }
2237 \seq_gput_right:Nn
2238   \g_@@_renderers_seq
2239   { inlineHtmlTag }
2240 \prop_gput:Nnn
2241   \g_@@_renderer_arities_prop
2242   { inlineHtmlTag }
2243   { 1 }
2244 \cs_gset_protected:Npn
2245   \markdownRendererInputBlockHtmlElement
2246   {
2247     \markdownRendererInputBlockHtmlElementPrototype
2248   }
2249 \seq_gput_right:Nn
2250   \g_@@_renderers_seq
2251   { inputBlockHtmlElement }
2252 \prop_gput:Nnn
2253   \g_@@_renderer_arities_prop
2254   { inputBlockHtmlElement }
2255   { 1 }
2256 \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.

```
2257 \ExplSyntaxOn
2258 \cs_gset_protected:Npn
2259   \markdownRendererImage
2260   {
2261     \markdownRendererImagePrototype
2262   }
2263 \seq_gput_right:Nn
2264   \g_@@_renderers_seq
2265   { image }
2266 \prop_gput:Nnn
2267   \g_@@_renderer_arities_prop
2268   { image }
2269   { 4 }
2270 \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.

```
2271 \ExplSyntaxOn
2272 \cs_gset_protected:Npn
2273   \markdownRendererImageAttributeContextBegin
2274   {
2275     \markdownRendererImageAttributeContextBeginPrototype
2276   }
2277 \seq_gput_right:Nn
2278   \g_@@_renderers_seq
2279   { imageAttributeContextBegin }
2280 \prop_gput:Nnn
2281   \g_@@_renderer_arities_prop
2282   { imageAttributeContextBegin }
2283   { 0 }
2284 \cs_gset_protected:Npn
2285   \markdownRendererImageAttributeContextEnd
2286   {
2287     \markdownRendererImageAttributeContextEndPrototype
2288   }
2289 \seq_gput_right:Nn
2290   \g_@@_renderers_seq
```

```

2291 { imageAttributeContextEnd }
2292 \prop_gput:Nnn
2293 \g_@@_renderer_arities_prop
2294 { imageAttributeContextEnd }
2295 { 0 }
2296 \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.

```

2297 \ExplSyntaxOn
2298 \cs_gset_protected:Npn
2299 \markdownRendererInterblockSeparator
2300 {
2301   \markdownRendererInterblockSeparatorPrototype
2302 }
2303 \seq_gput_right:Nn
2304 \g_@@_renderers_seq
2305 { interblockSeparator }
2306 \prop_gput:Nnn
2307 \g_@@_renderer_arities_prop
2308 { interblockSeparator }
2309 { 0 }
2310 \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.

```

2311 \ExplSyntaxOn
2312 \cs_gset_protected:Npn
2313 \markdownRendererParagraphSeparator
2314 {
2315   \markdownRendererParagraphSeparatorPrototype
2316 }
2317 \seq_gput_right:Nn
2318 \g_@@_renderers_seq
2319 { paragraphSeparator }
2320 \prop_gput:Nnn
2321 \g_@@_renderer_arities_prop
2322 { paragraphSeparator }
2323 { 0 }
2324 \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.

```
2325 \ExplSyntaxOn
2326 \cs_gset_protected:Npn
2327   \markdownRendererLineBlockBegin
2328   {
2329     \markdownRendererLineBlockBeginPrototype
2330   }
2331 \seq_gput_right:Nn
2332   \g_@@_renderers_seq
2333   { lineBlockBegin }
2334 \prop_gput:Nnn
2335   \g_@@_renderer_arities_prop
2336   { lineBlockBegin }
2337   { 0 }
2338 \cs_gset_protected:Npn
2339   \markdownRendererLineBlockEnd
2340   {
2341     \markdownRendererLineBlockEndPrototype
2342   }
2343 \seq_gput_right:Nn
2344   \g_@@_renderers_seq
2345   { lineBlockEnd }
2346 \prop_gput:Nnn
2347   \g_@@_renderer_arities_prop
2348   { lineBlockEnd }
2349   { 0 }
2350 \ExplSyntaxOff
```

### 2.2.5.23 Line Break Renderers

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

```
2351 \ExplSyntaxOn
2352 \cs_gset_protected:Npn
2353   \markdownRendererSoftLineBreak
2354   {
2355     \markdownRendererSoftLineBreakPrototype
2356   }
2357 \seq_gput_right:Nn
2358   \g_@@_renderers_seq
2359   { softLineBreak }
2360 \prop_gput:Nnn
```

```

2361 \g_@@_renderer_arities_prop
2362 { softLineBreak }
2363 { 0 }
2364 \ExplSyntaxOff

```

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

```

2365 \ExplSyntaxOn
2366 \cs_gset_protected:Npn
2367 \markdownRendererHardLineBreak
2368 {
2369   \markdownRendererHardLineBreakPrototype
2370 }
2371 \seq_gput_right:Nn
2372 \g_@@_renderers_seq
2373 { hardLineBreak }
2374 \prop_gput:Nnn
2375 \g_@@_renderer_arities_prop
2376 { hardLineBreak }
2377 { 0 }
2378 \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.

```

2379 \ExplSyntaxOn
2380 \cs_gset_protected:Npn
2381 \markdownRendererLink
2382 {
2383   \markdownRendererLinkPrototype
2384 }
2385 \seq_gput_right:Nn
2386 \g_@@_renderers_seq
2387 { link }
2388 \prop_gput:Nnn
2389 \g_@@_renderer_arities_prop
2390 { link }
2391 { 4 }
2392 \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.

```

2393 \ExplSyntaxOn
2394 \cs_gset_protected:Npn
2395   \markdownRendererLinkAttributeContextBegin
2396   {
2397     \markdownRendererLinkAttributeContextBeginPrototype
2398   }
2399 \seq_gput_right:Nn
2400   \g_@@_renderers_seq
2401   { linkAttributeContextBegin }
2402 \prop_gput:Nnn
2403   \g_@@_renderer_arities_prop
2404   { linkAttributeContextBegin }
2405   { 0 }
2406 \cs_gset_protected:Npn
2407   \markdownRendererLinkAttributeContextEnd
2408   {
2409     \markdownRendererLinkAttributeContextEndPrototype
2410   }
2411 \seq_gput_right:Nn
2412   \g_@@_renderers_seq
2413   { linkAttributeContextEnd }
2414 \prop_gput:Nnn
2415   \g_@@_renderer_arities_prop
2416   { linkAttributeContextEnd }
2417   { 0 }
2418 \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.

```

2419 \ExplSyntaxOn
2420 \cs_gset_protected:Npn
2421   \markdownRendererMark
2422   {
2423     \markdownRendererMarkPrototype
2424   }
2425 \seq_gput_right:Nn
2426   \g_@@_renderers_seq
2427   { mark }
2428 \prop_gput:Nnn
2429   \g_@@_renderer_arities_prop

```

```

2430 { mark }
2431 { 1 }
2432 \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  $\text{\TeX}$  document may contain any number of markdown documents. Additionally, markdown documents may appear not only in a sequence, but several markdown documents may also be *nested*. Redefinitions of the macros should take this into account.

```

2433 \ExplSyntaxOn
2434 \cs_gset_protected:Npn
2435   \markdownRendererDocumentBegin
2436   {
2437     \markdownRendererDocumentBeginPrototype
2438   }
2439 \seq_gput_right:Nn
2440   \g_@@_renderers_seq
2441   { documentBegin }
2442 \prop_gput:Nnn
2443   \g_@@_renderer_arities_prop
2444   { documentBegin }
2445   { 0 }
2446 \cs_gset_protected:Npn
2447   \markdownRendererDocumentEnd
2448   {
2449     \markdownRendererDocumentEndPrototype
2450   }
2451 \seq_gput_right:Nn
2452   \g_@@_renderers_seq
2453   { documentEnd }
2454 \prop_gput:Nnn
2455   \g_@@_renderer_arities_prop
2456   { documentEnd }
2457   { 0 }
2458 \ExplSyntaxOff

```

### 2.2.5.28 Non-Breaking Space Renderer

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

```

2459 \ExplSyntaxOn
2460 \cs_gset_protected:Npn
2461   \markdownRendererNbsp

```

```

2462 {
2463   \markdownRendererNbspPrototype
2464 }
2465 \seq_gput_right:Nn
2466   \g_@@_renderers_seq
2467   { nbsp }
2468 \prop_gput:Nnn
2469   \g_@@_renderer_arities_prop
2470   { nbsp }
2471   { 0 }
2472 \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.

```

2473 \def\markdownRendererNote{%
2474   \markdownRendererNotePrototype}%
2475 \ExplSyntaxOn
2476 \seq_gput_right:Nn
2477   \g_@@_renderers_seq
2478   { note }
2479 \prop_gput:Nnn
2480   \g_@@_renderer_arities_prop
2481   { note }
2482   { 1 }
2483 \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.

```

2484 \ExplSyntaxOn
2485 \cs_gset_protected:Npn
2486   \markdownRendererOlBegin
2487   {
2488     \markdownRendererOlBeginPrototype
2489   }
2490 \seq_gput_right:Nn
2491   \g_@@_renderers_seq
2492   { olBegin }
2493 \prop_gput:Nnn
2494   \g_@@_renderer_arities_prop

```



```

2495 { olBegin }
2496 { 0 }
2497 \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.

```

2498 \ExplSyntaxOn
2499 \cs_gset_protected:Npn
2500   \markdownRendererOlBeginTight
2501   {
2502     \markdownRendererOlBeginTightPrototype
2503   }
2504 \seq_gput_right:Nn
2505   \g_@@_renderers_seq
2506   { olBeginTight }
2507 \prop_gput:Nnn
2508   \g_@@_renderer_arities_prop
2509   { olBeginTight }
2510   { 0 }
2511 \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`).

```

2512 \ExplSyntaxOn
2513 \cs_gset_protected:Npn
2514   \markdownRendererFancyOlBegin
2515   {
2516     \markdownRendererFancyOlBeginPrototype
2517   }
2518 \seq_gput_right:Nn
2519   \g_@@_renderers_seq
2520   { fancyOlBegin }
2521 \prop_gput:Nnn
2522   \g_@@_renderer_arities_prop
2523   { fancyOlBegin }
2524   { 2 }
2525 \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.

```

2526 \ExplSyntaxOn
2527 \cs_gset_protected:Npn
2528   \markdownRendererFancyOlBeginTight
2529   {
2530     \markdownRendererFancyOlBeginTightPrototype
2531   }
2532 \seq_gput_right:Nn
2533   \g_@@_renderers_seq
2534   { fancyOlBeginTight }
2535 \prop_gput:Nnn
2536   \g_@@_renderer_arities_prop
2537   { fancyOlBeginTight }
2538   { 2 }
2539 \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.

```

2540 \ExplSyntaxOn
2541 \cs_gset_protected:Npn
2542   \markdownRendererOlItem
2543   {
2544     \markdownRendererOlItemPrototype
2545   }
2546 \seq_gput_right:Nn
2547   \g_@@_renderers_seq
2548   { olItem }
2549 \prop_gput:Nnn
2550   \g_@@_renderer_arities_prop
2551   { olItem }
2552   { 0 }
2553 \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.

```

2554 \ExplSyntaxOn
2555 \cs_gset_protected:Npn
2556   \markdownRendererOlItemEnd
2557   {
2558     \markdownRendererOlItemEndPrototype
2559   }

```

```

2560 \seq_gput_right:Nn
2561   \g_@@_renderers_seq
2562   { olItemEnd }
2563 \prop_gput:Nnn
2564   \g_@@_renderer_arities_prop
2565   { olItemEnd }
2566   { 0 }
2567 \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.

```

2568 \ExplSyntaxOn
2569 \cs_gset_protected:Npn
2570   \markdownRendererOlItemWithNumber
2571   {
2572     \markdownRendererOlItemWithNumberPrototype
2573   }
2574 \seq_gput_right:Nn
2575   \g_@@_renderers_seq
2576   { olItemWithNumber }
2577 \prop_gput:Nnn
2578   \g_@@_renderer_arities_prop
2579   { olItemWithNumber }
2580   { 1 }
2581 \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.

```

2582 \ExplSyntaxOn
2583 \cs_gset_protected:Npn
2584   \markdownRendererFancyOlItem
2585   {
2586     \markdownRendererFancyOlItemPrototype
2587   }
2588 \seq_gput_right:Nn
2589   \g_@@_renderers_seq
2590   { fancyOlItem }
2591 \prop_gput:Nnn
2592   \g_@@_renderer_arities_prop
2593   { fancyOlItem }
2594   { 0 }
2595 \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.

```

2596 \ExplSyntaxOn
2597 \cs_gset_protected:Npn
2598   \markdownRendererFancyO1ItemEnd
2599   {
2600     \markdownRendererFancyO1ItemEndPrototype
2601   }
2602 \seq_gput_right:Nn
2603   \g_@@_renderers_seq
2604   { fancyO1ItemEnd }
2605 \prop_gput:Nnn
2606   \g_@@_renderer_arities_prop
2607   { fancyO1ItemEnd }
2608   { 0 }
2609 \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.

```

2610 \ExplSyntaxOn
2611 \cs_gset_protected:Npn
2612   \markdownRendererFancyO1ItemWithNumber
2613   {
2614     \markdownRendererFancyO1ItemWithNumberPrototype
2615   }
2616 \seq_gput_right:Nn
2617   \g_@@_renderers_seq
2618   { fancyO1ItemWithNumber }
2619 \prop_gput:Nnn
2620   \g_@@_renderer_arities_prop
2621   { fancyO1ItemWithNumber }
2622   { 1 }
2623 \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.

```

2624 \ExplSyntaxOn
2625 \cs_gset_protected:Npn
2626   \markdownRendererO1End
2627   {
2628     \markdownRendererO1EndPrototype

```

```

2629 }
2630 \seq_gput_right:Nn
2631 \g_@@_renderers_seq
2632 { olEnd }
2633 \prop_gput:Nnn
2634 \g_@@_renderer_arities_prop
2635 { olEnd }
2636 { 0 }
2637 \ExplSyntaxOff

```

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

```

2638 \ExplSyntaxOn
2639 \cs_gset_protected:Npn
2640 \markdownRendererOlEndTight
2641 {
2642   \markdownRendererOlEndTightPrototype
2643 }
2644 \seq_gput_right:Nn
2645 \g_@@_renderers_seq
2646 { olEndTight }
2647 \prop_gput:Nnn
2648 \g_@@_renderer_arities_prop
2649 { olEndTight }
2650 { 0 }
2651 \ExplSyntaxOff

```

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

```

2652 \ExplSyntaxOn
2653 \cs_gset_protected:Npn
2654 \markdownRendererFancyOlEnd
2655 {
2656   \markdownRendererFancyOlEndPrototype
2657 }
2658 \seq_gput_right:Nn
2659 \g_@@_renderers_seq
2660 { fancyOlEnd }
2661 \prop_gput:Nnn
2662 \g_@@_renderer_arities_prop
2663 { fancyOlEnd }
2664 { 0 }

```

```
2665 \ExplSyntaxOff
```

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

```
2666 \ExplSyntaxOn
2667 \cs_gset_protected:Npn
2668   \markdownRendererFancyOlEndTight
2669   {
2670     \markdownRendererFancyOlEndTightPrototype
2671   }
2672 \seq_gput_right:Nn
2673   \g_@@_renderers_seq
2674   { fancyOlEndTight }
2675 \prop_gput:Nnn
2676   \g_@@_renderer_arities_prop
2677   { fancyOlEndTight }
2678   { 0 }
2679 \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.

```
2680 \ExplSyntaxOn
2681 \cs_gset_protected:Npn
2682   \markdownRendererInputRawInline
2683   {
2684     \markdownRendererInputRawInlinePrototype
2685   }
2686 \seq_gput_right:Nn
2687   \g_@@_renderers_seq
2688   { inputRawInline }
2689 \prop_gput:Nnn
2690   \g_@@_renderer_arities_prop
2691   { inputRawInline }
2692   { 2 }
2693 \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.

```

2694 \ExplSyntaxOn
2695 \cs_gset_protected:Npn
2696   \markdownRendererInputRawBlock
2697   {
2698     \markdownRendererInputRawBlockPrototype
2699   }
2700 \seq_gput_right:Nn
2701   \g_@@_renderers_seq
2702   { inputRawBlock }
2703 \prop_gput:Nnn
2704   \g_@@_renderer_arities_prop
2705   { inputRawBlock }
2706   { 2 }
2707 \ExplSyntaxOff

```

### 2.2.5.32 Section Renderers

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

```

2708 \ExplSyntaxOn
2709 \cs_gset_protected:Npn
2710   \markdownRendererSectionBegin
2711   {
2712     \markdownRendererSectionBeginPrototype
2713   }
2714 \seq_gput_right:Nn
2715   \g_@@_renderers_seq
2716   { sectionBegin }
2717 \prop_gput:Nnn
2718   \g_@@_renderer_arities_prop
2719   { sectionBegin }
2720   { 0 }
2721 \cs_gset_protected:Npn
2722   \markdownRendererSectionEnd
2723   {
2724     \markdownRendererSectionEndPrototype
2725   }
2726 \seq_gput_right:Nn
2727   \g_@@_renderers_seq
2728   { sectionEnd }
2729 \prop_gput:Nnn
2730   \g_@@_renderer_arities_prop
2731   { sectionEnd }
2732   { 0 }
2733 \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.

```

2734 \ExplSyntaxOn
2735 \cs_gset_protected:Npn
2736   \markdownRendererReplacementCharacter
2737   {
2738     \markdownRendererReplacementCharacterPrototype
2739   }
2740 \seq_gput_right:Nn
2741   \g_@@_renderers_seq
2742   { replacementCharacter }
2743 \prop_gput:Nnn
2744   \g_@@_renderer_arities_prop
2745   { replacementCharacter }
2746   { 0 }
2747 \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`.

```

2748 \ExplSyntaxOn
2749 \cs_gset_protected:Npn
2750   \markdownRendererLeftBrace
2751   {
2752     \markdownRendererLeftBracePrototype
2753   }
2754 \seq_gput_right:Nn
2755   \g_@@_renderers_seq
2756   { leftBrace }
2757 \prop_gput:Nnn
2758   \g_@@_renderer_arities_prop
2759   { leftBrace }
2760   { 0 }
2761 \cs_gset_protected:Npn
2762   \markdownRendererRightBrace
2763   {
2764     \markdownRendererRightBracePrototype
2765   }
2766 \seq_gput_right:Nn
2767   \g_@@_renderers_seq
2768   { rightBrace }
2769 \prop_gput:Nnn
2770   \g_@@_renderer_arities_prop
2771   { rightBrace }

```



```

2772 { 0 }
2773 \cs_gset_protected:Npn
2774 \markdownRendererDollarSign
2775 {
2776   \markdownRendererDollarSignPrototype
2777 }
2778 \seq_gput_right:Nn
2779 \g_@@_renderers_seq
2780 { dollarSign }
2781 \prop_gput:Nnn
2782 \g_@@_renderer_arities_prop
2783 { dollarSign }
2784 { 0 }
2785 \cs_gset_protected:Npn
2786 \markdownRendererPercentSign
2787 {
2788   \markdownRendererPercentSignPrototype
2789 }
2790 \seq_gput_right:Nn
2791 \g_@@_renderers_seq
2792 { percentSign }
2793 \prop_gput:Nnn
2794 \g_@@_renderer_arities_prop
2795 { percentSign }
2796 { 0 }
2797 \cs_gset_protected:Npn
2798 \markdownRendererAmpersand
2799 {
2800   \markdownRendererAmpersandPrototype
2801 }
2802 \seq_gput_right:Nn
2803 \g_@@_renderers_seq
2804 { ampersand }
2805 \prop_gput:Nnn
2806 \g_@@_renderer_arities_prop
2807 { ampersand }
2808 { 0 }
2809 \cs_gset_protected:Npn
2810 \markdownRendererUnderscore
2811 {
2812   \markdownRendererUnderscorePrototype
2813 }
2814 \seq_gput_right:Nn
2815 \g_@@_renderers_seq
2816 { underscore }
2817 \prop_gput:Nnn
2818 \g_@@_renderer_arities_prop

```

```

2819 { underscore }
2820 { 0 }
2821 \cs_gset_protected:Npn
2822 \markdownRendererHash
2823 {
2824     \markdownRendererHashPrototype
2825 }
2826 \seq_gput_right:Nn
2827 \g_@@_renderers_seq
2828 { hash }
2829 \prop_gput:Nnn
2830 \g_@@_renderer_arities_prop
2831 { hash }
2832 { 0 }
2833 \cs_gset_protected:Npn
2834 \markdownRendererCircumflex
2835 {
2836     \markdownRendererCircumflexPrototype
2837 }
2838 \seq_gput_right:Nn
2839 \g_@@_renderers_seq
2840 { circumflex }
2841 \prop_gput:Nnn
2842 \g_@@_renderer_arities_prop
2843 { circumflex }
2844 { 0 }
2845 \cs_gset_protected:Npn
2846 \markdownRendererBackslash
2847 {
2848     \markdownRendererBackslashPrototype
2849 }
2850 \seq_gput_right:Nn
2851 \g_@@_renderers_seq
2852 { backslash }
2853 \prop_gput:Nnn
2854 \g_@@_renderer_arities_prop
2855 { backslash }
2856 { 0 }
2857 \cs_gset_protected:Npn
2858 \markdownRendererTilde
2859 {
2860     \markdownRendererTildePrototype
2861 }
2862 \seq_gput_right:Nn
2863 \g_@@_renderers_seq
2864 { tilde }
2865 \prop_gput:Nnn

```

```

2866 \g_@@_renderer_arities_prop
2867 { tilde }
2868 { 0 }
2869 \cs_gset_protected:Npn
2870 \markdownRendererPipe
2871 {
2872   \markdownRendererPipePrototype
2873 }
2874 \seq_gput_right:Nn
2875 \g_@@_renderers_seq
2876 { pipe }
2877 \prop_gput:Nnn
2878 \g_@@_renderer_arities_prop
2879 { pipe }
2880 { 0 }
2881 \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.

```

2882 \ExplSyntaxOn
2883 \cs_gset_protected:Npn
2884 \markdownRendererStrikeThrough
2885 {
2886   \markdownRendererStrikeThroughPrototype
2887 }
2888 \seq_gput_right:Nn
2889 \g_@@_renderers_seq
2890 { strikeThrough }
2891 \prop_gput:Nnn
2892 \g_@@_renderer_arities_prop
2893 { strikeThrough }
2894 { 1 }
2895 \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.

```

2896 \ExplSyntaxOn
2897 \cs_gset_protected:Npn
2898 \markdownRendererSubscript

```

```

2899 {
2900   \markdownRendererSubscriptPrototype
2901 }
2902 \seq_gput_right:Nn
2903   \g_@@_renderers_seq
2904   { subscript }
2905 \prop_gput:Nnn
2906   \g_@@_renderer_arities_prop
2907   { subscript }
2908   { 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.

```

2909 \cs_gset_protected:Npn
2910   \markdownRendererSuperscript
2911   {
2912     \markdownRendererSuperscriptPrototype
2913   }
2914 \seq_gput_right:Nn
2915   \g_@@_renderers_seq
2916   { superscript }
2917 \prop_gput:Nnn
2918   \g_@@_renderer_arities_prop
2919   { superscript }
2920   { 1 }
2921 \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.

```

2922 \ExplSyntaxOn
2923 \cs_gset_protected:Npn
2924   \markdownRendererTableAttributeContextBegin
2925   {
2926     \markdownRendererTableAttributeContextBeginPrototype
2927   }
2928 \seq_gput_right:Nn
2929   \g_@@_renderers_seq
2930   { tableAttributeContextBegin }

```

```

2931 \prop_gput:Nnn
2932   \g_@@_renderer_arities_prop
2933   { tableAttributeContextBegin }
2934   { 0 }
2935 \cs_gset_protected:Npn
2936   \markdownRendererTableAttributeContextEnd
2937   {
2938     \markdownRendererTableAttributeContextEndPrototype
2939   }
2940 \seq_gput_right:Nn
2941   \g_@@_renderers_seq
2942   { tableAttributeContextEnd }
2943 \prop_gput:Nnn
2944   \g_@@_renderer_arities_prop
2945   { tableAttributeContextEnd }
2946   { 0 }
2947 \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.

```

2948 \ExplSyntaxOn
2949 \cs_gset_protected:Npn
2950   \markdownRendererTable
2951   {
2952     \markdownRendererTablePrototype
2953   }
2954 \seq_gput_right:Nn
2955   \g_@@_renderers_seq
2956   { table }
2957 \prop_gput:Nnn
2958   \g_@@_renderer_arities_prop
2959   { table }
2960   { 3 }
2961 \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.

```
2962 \ExplSyntaxOn
2963 \cs_gset_protected:Npn
2964   \markdownRendererInlineMath
2965   {
2966     \markdownRendererInlineMathPrototype
2967   }
2968 \seq_gput_right:Nn
2969   \g_@@_renderers_seq
2970   { inlineMath }
2971 \prop_gput:Nnn
2972   \g_@@_renderer_arities_prop
2973   { inlineMath }
2974   { 1 }
2975 \cs_gset_protected:Npn
2976   \markdownRendererDisplayMath
2977   {
2978     \markdownRendererDisplayMathPrototype
2979   }
2980 \seq_gput_right:Nn
2981   \g_@@_renderers_seq
2982   { displayMath }
2983 \prop_gput:Nnn
2984   \g_@@_renderer_arities_prop
2985   { displayMath }
2986   { 1 }
2987 \ExplSyntaxOff
```

#### 2.2.5.41 Thematic Break Renderer

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

```
2988 \ExplSyntaxOn
2989 \cs_gset_protected:Npn
2990   \markdownRendererThematicBreak
2991   {
2992     \markdownRendererThematicBreakPrototype
2993   }
2994 \seq_gput_right:Nn
2995   \g_@@_renderers_seq
2996   { thematicBreak }
```

```

2997 \prop_gput:Nnn
2998   \g_@@_renderer_arities_prop
2999   { thematicBreak }
3000   { 0 }
3001 \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.

```

3002 \ExplSyntaxOn
3003 \cs_gset_protected:Npn
3004   \markdownRendererTickedBox
3005   {
3006     \markdownRendererTickedBoxPrototype
3007   }
3008 \seq_gput_right:Nn
3009   \g_@@_renderers_seq
3010   { tickedBox }
3011 \prop_gput:Nnn
3012   \g_@@_renderer_arities_prop
3013   { tickedBox }
3014   { 0 }
3015 \cs_gset_protected:Npn
3016   \markdownRendererHalfTickedBox
3017   {
3018     \markdownRendererHalfTickedBoxPrototype
3019   }
3020 \seq_gput_right:Nn
3021   \g_@@_renderers_seq
3022   { halfTickedBox }
3023 \prop_gput:Nnn
3024   \g_@@_renderer_arities_prop
3025   { halfTickedBox }
3026   { 0 }
3027 \cs_gset_protected:Npn
3028   \markdownRendererUntickedBox
3029   {
3030     \markdownRendererUntickedBoxPrototype
3031   }
3032 \seq_gput_right:Nn
3033   \g_@@_renderers_seq
3034   { untickedBox }
3035 \prop_gput:Nnn

```

```

3036 \g_@@_renderer_arities_prop
3037 { untickedBox }
3038 { 0 }
3039 \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 four parameters:

1. The fully escaped text of the warning or error that can be directly typeset
2. The raw text of the warning or error that can be used outside typesetting for e.g. logging the warning or error.
3. The fully escaped text with more details about the warning or error that can be directly typeset. Can be empty, unlike the first two parameters.
4. The raw text with more details about the warning or error that can be used outside typesetting for e.g. logging the warning or error. Can be empty, unlike the first two parameters.

```

3040 \ExplSyntaxOn
3041 \cs_gset_protected:Npn
3042 \markdownRendererWarning
3043 {
3044   \markdownRendererWarningPrototype
3045 }
3046 \cs_gset_protected:Npn
3047 \markdownRendererError
3048 {
3049   \markdownRendererErrorPrototype
3050 }
3051 \seq_gput_right:Nn
3052 \g_@@_renderers_seq
3053 { warning }
3054 \prop_gput:Nnn
3055 \g_@@_renderer_arities_prop
3056 { warning }
3057 { 4 }
3058 \seq_gput_right:Nn
3059 \g_@@_renderers_seq
3060 { error }
3061 \prop_gput:Nnn
3062 \g_@@_renderer_arities_prop
3063 { error }
3064 { 4 }
3065 \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.

```
3066 \ExplSyntaxOn
3067 \cs_gset_protected:Npn
3068   \markdownRendererJekyllDataBegin
3069   {
3070     \markdownRendererJekyllDataBeginPrototype
3071   }
3072 \seq_gput_right:Nn
3073   \g_@@_renderers_seq
3074   { jekyllDataBegin }
3075 \prop_gput:Nnn
3076   \g_@@_renderer_arities_prop
3077   { jekyllDataBegin }
3078   { 0 }
3079 \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.

```
3080 \ExplSyntaxOn
3081 \cs_gset_protected:Npn
3082   \markdownRendererJekyllDataEnd
3083   {
3084     \markdownRendererJekyllDataEndPrototype
3085   }
3086 \seq_gput_right:Nn
3087   \g_@@_renderers_seq
3088   { jekyllDataEnd }
3089 \prop_gput:Nnn
3090   \g_@@_renderer_arities_prop
3091   { jekyllDataEnd }
3092   { 0 }
3093 \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.

```
3094 \ExplSyntaxOn
3095 \cs_gset_protected:Npn
3096   \markdownRendererJekyllDataMappingBegin
```

```

3097 {
3098   \markdownRendererJekyllDataMappingBeginPrototype
3099 }
3100 \seq_gput_right:Nn
3101   \g_@@_renderers_seq
3102   { jekyllDataMappingBegin }
3103 \prop_gput:Nnn
3104   \g_@@_renderer_arities_prop
3105   { jekyllDataMappingBegin }
3106   { 2 }
3107 \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.

```

3108 \ExplSyntaxOn
3109 \cs_gset_protected:Npn
3110   \markdownRendererJekyllDataMappingEnd
3111   {
3112     \markdownRendererJekyllDataMappingEndPrototype
3113   }
3114 \seq_gput_right:Nn
3115   \g_@@_renderers_seq
3116   { jekyllDataMappingEnd }
3117 \prop_gput:Nnn
3118   \g_@@_renderer_arities_prop
3119   { jekyllDataMappingEnd }
3120   { 0 }
3121 \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.

```

3122 \ExplSyntaxOn
3123 \cs_gset_protected:Npn
3124   \markdownRendererJekyllDataSequenceBegin
3125   {
3126     \markdownRendererJekyllDataSequenceBeginPrototype
3127   }
3128 \seq_gput_right:Nn
3129   \g_@@_renderers_seq
3130   { jekyllDataSequenceBegin }
3131 \prop_gput:Nnn
3132   \g_@@_renderer_arities_prop

```

```

3133 { jekyllDataSequenceBegin }
3134 { 2 }
3135 \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.

```

3136 \ExplSyntaxOn
3137 \cs_gset_protected:Npn
3138   \markdownRendererJekyllDataSequenceEnd
3139   {
3140     \markdownRendererJekyllDataSequenceEndPrototype
3141   }
3142 \seq_gput_right:Nn
3143   \g_@@_renderers_seq
3144   { jekyllDataSequenceEnd }
3145 \prop_gput:Nnn
3146   \g_@@_renderer_arities_prop
3147   { jekyllDataSequenceEnd }
3148   { 0 }
3149 \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.

```

3150 \ExplSyntaxOn
3151 \cs_gset_protected:Npn
3152   \markdownRendererJekyllDataBoolean
3153   {
3154     \markdownRendererJekyllDataBooleanPrototype
3155   }
3156 \seq_gput_right:Nn
3157   \g_@@_renderers_seq
3158   { jekyllDataBoolean }
3159 \prop_gput:Nnn
3160   \g_@@_renderer_arities_prop
3161   { jekyllDataBoolean }
3162   { 2 }
3163 \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.

```

3164 \ExplSyntaxOn
3165 \cs_gset_protected:Npn
3166   \markdownRendererJekyllDataNumber
3167   {
3168     \markdownRendererJekyllDataNumberPrototype
3169   }
3170 \seq_gput_right:Nn
3171   \g_@@_renderers_seq
3172   { jekyllDataNumber }
3173 \prop_gput:Nnn
3174   \g_@@_renderer_arities_prop
3175   { jekyllDataNumber }
3176   { 2 }
3177 \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 `\markdownRendererJekyllDataTypographicString` receives the scalar value after all markdown markup and special  $\text{\TeX}$  characters in the string have been replaced by  $\text{\TeX}$  macros, `\markdownRendererJekyllDataProgrammaticString` receives the raw scalar value. Therefore, whereas the `\markdownRendererJekyllDataTypographicString` macro is more appropriate for texts that are supposed to be typeset with  $\text{\TeX}$ , 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  $\text{\TeX}$ .

```

3178 \ExplSyntaxOn
3179 \cs_gset_protected:Npn
3180   \markdownRendererJekyllDataTypographicString
3181   {
3182     \markdownRendererJekyllDataTypographicStringPrototype
3183   }
3184 \cs_gset_protected:Npn
3185   \markdownRendererJekyllDataProgrammaticString
3186   {
3187     \markdownRendererJekyllDataProgrammaticStringPrototype
3188   }
3189 \seq_gput_right:Nn
3190   \g_@@_renderers_seq
3191   { jekyllDataTypographicString }
3192 \prop_gput:Nnn

```

```

3193 \g_@@_renderer_arities_prop
3194 { jekyllDataTypographicString }
3195 { 2 }
3196 \seq_gput_right:Nn
3197 \g_@@_renderers_seq
3198 { jekyllDataProgrammaticString }
3199 \prop_gput:Nnn
3200 \g_@@_renderer_arities_prop
3201 { jekyllDataProgrammaticString }
3202 { 2 }
3203 \ExplSyntaxOff

```

Before Markdown 3.7.0, the `\markdownRendererJekyllDataTypographicString` macro was named `\markdownRendererJekyllDataString` and the `\markdownRendererJekyllDataString` macro was not produced. The `\markdownRendererJekyllDataString` has been deprecated and will be removed in Markdown 4.0.0.

```

3204 \ExplSyntaxOn
3205 \cs_gset:Npn
3206 \markdownRendererJekyllDataTypographicString
3207 {
3208   \cs_if_exist:NTF
3209     \markdownRendererJekyllDataString
3210     {
3211       \@@_if_option:nTF
3212         { experimental }
3213         {
3214           \markdownError
3215           {
3216             The~jekyllDataString~renderer~has~been~deprecated,~
3217             to~be~removed~in~Markdown~4.0.0
3218           }
3219         }
3220         {
3221           \markdownWarning
3222           {
3223             The~jekyllDataString~renderer~has~been~deprecated,~
3224             to~be~removed~in~Markdown~4.0.0
3225           }
3226           \markdownRendererJekyllDataString
3227         }
3228       }
3229     {
3230       \cs_if_exist:NTF
3231         \markdownRendererJekyllDataStringPrototype
3232         {
3233           \@@_if_option:nTF
3234             { experimental }

```

```

3235         {
3236             \markdownError
3237             {
3238                 The~jekyllDataString~renderer~prototype~
3239                 has~been~deprecated,~
3240                 to~be~removed~in~Markdown~4.0.0
3241             }
3242         }
3243         {
3244             \markdownWarning
3245             {
3246                 The~jekyllDataString~renderer~prototype~
3247                 has~been~deprecated,~
3248                 to~be~removed~in~Markdown~4.0.0
3249             }
3250             \markdownRendererJekyllDataStringPrototype
3251         }
3252     }
3253     {
3254         \markdownRendererJekyllDataTypographicStringPrototype
3255     }
3256 }
3257 }
3258 \seq_gput_right:Nn
3259 \g_@@_renderers_seq
3260 { jekyllDataString }
3261 \prop_gput:Nnn
3262 \g_@@_renderer_arities_prop
3263 { jekyllDataString }
3264 { 2 }
3265 \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.

```

3266 \ExplSyntaxOn
3267 \cs_gset_protected:Npn
3268 \markdownRendererJekyllDataEmpty
3269 {
3270     \markdownRendererJekyllDataEmptyPrototype
3271 }
3272 \seq_gput_right:Nn
3273 \g_@@_renderers_seq
3274 { jekyllDataEmpty }

```

```

3275 \prop_gput:Nnn
3276   \g_@@_renderer_arities_prop
3277   { jekyllDataEmpty }
3278   { 1 }
3279 \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` and `unprotectedRenderers` keys. The value for these keys 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.

Whereas the key `renderers` defines protected functions, which are usually preferable for typesetting, the key `unprotectedRenderers` defines unprotected functions, which are easier to expand and may be preferable for programming.

```

3280 \ExplSyntaxOn
3281 \cs_new:Nn \@@_define_renderers:
3282 {
3283   \seq_map_inline:Nn
3284     \g_@@_renderers_seq
3285     {
3286       \@@_define_renderer:n
3287       { ##1 }
3288     }
3289 }
3290 \cs_new:Nn \@@_define_renderer:n
3291 {
3292   \@@_renderer_tl_to_csname:nN
3293   { #1 }
3294   \l_tmpa_tl
3295   \prop_get:NnN
3296     \g_@@_renderer_arities_prop
3297     { #1 }
3298     \l_tmpb_tl
3299   \@@_define_renderer:ncV
3300   { #1 }
3301   { \l_tmpa_tl }
3302   \l_tmpb_tl
3303 }
3304 \cs_new:Nn \@@_renderer_tl_to_csname:nN
3305 {
3306   \tl_set:Nn
3307     \l_tmpa_tl
3308     { \str_uppercase:n { #1 } }
3309   \tl_set:Nx

```

```

3310     #2
3311     {
3312         markdownRenderer
3313         \tl_head:f { \l_tmpa_tl }
3314         \tl_tail:n { #1 }
3315     }
3316 }
3317 \tl_new:N
3318 \l_@@_renderer_definition_tl
3319 \bool_new:N
3320 \g_@@_appending_renderer_bool
3321 \bool_new:N
3322 \g_@@_unprotected_renderer_bool
3323 \cs_new:Nn \@@_define_renderer:nNn
3324 {
3325     \keys_define:nn
3326     { markdown/options/renderers }
3327     {
3328         #1 .code:n = {
3329             \tl_set:Nn
3330             \l_@@_renderer_definition_tl
3331             { ##1 }
3332             \regex_replace_all:nnN
3333             { \cP\#0 }
3334             { #1 }
3335             \l_@@_renderer_definition_tl
3336             \bool_if:NT
3337             \g_@@_appending_renderer_bool
3338             {
3339                 \@@_tl_set_from_cs:NNn
3340                 \l_tmpa_tl
3341                 #2
3342                 { #3 }
3343                 \tl_put_left:NV
3344                 \l_@@_renderer_definition_tl
3345                 \l_tmpa_tl
3346             }
3347             \bool_if:NTF
3348             \g_@@_unprotected_renderer_bool
3349             {
3350                 \tl_set:Nn
3351                 \l_tmpa_tl
3352                 { \cs_set:Npn }
3353             }
3354             {
3355                 \tl_set:Nn
3356                 \l_tmpa_tl

```



```

3357         { \cs_set_protected:Npn }
3358     }
3359     \exp_last_unbraced:NNV
3360     \cs_generate_from_arg_count:NNnV
3361     #2
3362     \l_tmpa_tl
3363     { #3 }
3364     \l_@@_renderer_definition_tl
3365 },
3366 }

```

If the token renderer macro has been deprecated, we undefine it.

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

```

3367     \str_if_eq:nnT
3368     { #1 }
3369     { jekyllDataString }
3370     {
3371         \cs_undefine:N
3372         #2
3373     }
3374 }

```

We define the function `\@@_tl_set_from_cs:NNn` [12]. 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.

```

3375 \cs_new_protected:Nn
3376 \@@_tl_set_from_cs:NNn
3377 {
3378     \tl_set:Nn
3379     \l_tmpa_tl
3380     { #2 }
3381     \int_step_inline:nn
3382     { #3 }
3383     {
3384         \exp_args:NNc
3385         \tl_put_right:Nn
3386         \l_tmpa_tl
3387         { @@_tl_set_from_cs_parameter_ ##1 }
3388     }
3389     \exp_args:NNV
3390     \tl_set:No
3391     \l_tmpb_tl
3392     \l_tmpa_tl
3393     \regex_replace_all:nnN
3394     { \cP. }

```

```

3395     { \0\0 }
3396     \l_tmpb_tl
3397     \int_step_inline:nn
3398     { #3 }
3399     {
3400         \regex_replace_all:nnN
3401         { \c { @@_tl_set_from_cs_parameter_ ##1 } }
3402         { \cP\# ##1 }
3403         \l_tmpb_tl
3404     }
3405     \tl_set:NV
3406     #1
3407     \l_tmpb_tl
3408 }
3409 \cs_generate_variant:Nn
3410 \@@_define_renderer:nNn
3411 { ncV }
3412 \cs_generate_variant:Nn
3413 \cs_generate_from_arg_count:NNnn
3414 { NNnV }
3415 \cs_generate_variant:Nn
3416 \tl_put_left:Nn
3417 { Nv }
3418 \keys_define:nn
3419 { markdown/options }
3420 {
3421     renderers .code:n = {
3422         \bool_gset_false:N
3423         \g_@@_unprotected_renderer_bool
3424         \keys_set:nn
3425         { markdown/options/renderers }
3426         { #1 }
3427     },
3428     unprotectedRenderers .code:n = {
3429         \bool_gset_true:N
3430         \g_@@_unprotected_renderer_bool
3431         \keys_set:nn
3432         { markdown/options/renderers }
3433         { #1 }
3434     },
3435 }

```

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.
  }
}

```

```

3436 \tl_new:N
3437   \l_@@_renderer_glob_definition_tl
3438 \seq_new:N
3439   \l_@@_renderer_glob_results_seq
3440 \regex_const:Nn
3441   \c_@@_appending_key_regex
3442   { \s*+$ }
3443 \keys_define:nn
3444   { markdown/options/renderers }
3445   {
3446     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 += {...},
        },
      },
    },
  },
},

```

```
}
```

```
3447      % TODO: Use `\\regex_if_match` in TeX Live 2025.
3448      \\regex_match:NVTF % noqa: w202
3449      \\c_@@_appending_key_regex
3450      \\l_keys_key_str
3451      {
3452        \\bool_gset_true:N
3453        \\g_@@_appending_renderer_bool
3454        \\tl_set:NV
3455        \\l_tmpa_tl
3456        \\l_keys_key_str
3457        \\regex_replace_once:NnN
3458        \\c_@@_appending_key_regex
3459        { }
3460        \\l_tmpa_tl
3461        \\tl_set:Nx
3462        \\l_tmpb_tl
3463        { { \\l_tmpa_tl } = }
3464        \\tl_put_right:Nn
3465        \\l_tmpb_tl
3466        { { #1 } }
3467        \\keys_set:nV
3468        { markdown/options/renderers }
3469        \\l_tmpb_tl
3470        \\bool_gset_false:N
3471        \\g_@@_appending_renderer_bool
3472      }
```

In addition to exact token renderer names, we also support wildcards (\*) and enumerations (l) 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 = {
```

```

    *lItem(|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.
  }
}

```

```

3473   {
3474       \@@_glob_seq:VnN
3475       \l_keys_key_str
3476       { g_@@_renderers_seq }
3477       \l_@@_renderer_glob_results_seq
3478       \seq_if_empty:NTF
3479       \l_@@_renderer_glob_results_seq
3480       {
3481           \msg_error:nnV
3482           { markdown }
3483           { undefined-renderer }
3484           \l_keys_key_str
3485       }
3486       {
3487           \tl_set:Nn
3488           \l_@@_renderer_glob_definition_tl
3489           { \exp_not:n { #1 } }
3490           \seq_map_inline:Nn
3491           \l_@@_renderer_glob_results_seq
3492           {
3493               \tl_set:Nn
3494               \l_tmpa_tl
3495               { { ##1 } = }
3496               \tl_put_right:Nx
3497               \l_tmpa_tl
3498               { { \l_@@_renderer_glob_definition_tl } }
3499               \keys_set:nV
3500               { markdown/options/renderers }
3501               \l_tmpa_tl
3502           }
3503       }
3504   }
3505 },

```

```

3506 }
3507 \msg_new:nnn
3508 { markdown }
3509 { undefined-renderer }
3510 {
3511     Renderer~#1~is~undefined.
3512 }
3513 \cs_generate_variant:Nn
3514 \@@_glob_seq:nnN
3515 { VnN }
3516 \cs_generate_variant:Nn
3517 \cs_generate_from_arg_count:NNnn
3518 { cNvV }
3519 \cs_generate_variant:Nn
3520 \msg_error:nnn
3521 { nnV }
3522 \prg_generate_conditional_variant:Nnn
3523 % TODO: Use ``regex_if_match`` in TeX Live 2025.
3524 \regex_match:Nn % noqa: w202
3525 { NV }
3526 { TF }
3527 \prop_new:N
3528 \g_@@_glob_cache_prop
3529 \tl_new:N
3530 \l_@@_current_glob_tl
3531 \cs_new:Nn
3532 \@@_glob_seq:nnN
3533 {
3534     \tl_set:Nn
3535     \l_@@_current_glob_tl
3536     { ^ #1 $ }
3537     \prop_get:NeNTF
3538     \g_@@_glob_cache_prop
3539     { #2 / \l_@@_current_glob_tl }
3540     \l_tmpa_clist
3541     {
3542         \seq_set_from_clist:NN
3543         #3
3544         \l_tmpa_clist
3545     }
3546     {
3547         \seq_clear:N
3548         #3
3549         \regex_replace_all:nnN
3550         { \* }
3551         { .* }
3552         \l_@@_current_glob_tl

```

```

3553     \regex_set:NV
3554     \l_tmpa_regex
3555     \l_@@_current_glob_tl
3556     \seq_map_inline:cn
3557     { #2 }
3558     {
3559         % TODO: Use ``\regex_if_match`` in TeX Live 2025.
3560         \regex_match:NnT % noqa: w202
3561         \l_tmpa_regex
3562         { ##1 }
3563         {
3564             \seq_put_right:Nn
3565             #3
3566             { ##1 }
3567         }
3568     }
3569     \clist_set_from_seq:NN
3570     \l_tmpa_clist
3571     #3
3572     \prop_gput:NeV
3573     \g_@@_glob_cache_prop
3574     { #2 / \l_@@_current_glob_tl }
3575     \l_tmpa_clist
3576 }
3577 }
3578 \cs_generate_variant:Nn
3579 \regex_set:Nn
3580 { NV }
3581 \cs_generate_variant:Nn
3582 \prop_gput:Nnn
3583 { NeV }

```

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

```

3584 \str_if_eq:VVT
3585 \c_@@_top_layer_tl
3586 \c_@@_option_layer_plain_tex_tl
3587 {
3588     \@@_define_renderers:
3589 }
3590 \ExplSyntaxOff

```

## 2.2.6 Token Renderer Prototypes

### 2.2.6.1 YAML Metadata Renderer Prototypes

For simple YAML metadata, a simple high-level interface is provided that can be programmed by setting the expl3 key-values [2] for the module `markdown/jekyllData`.

```
3591 \ExplSyntaxOn
3592 \keys_define:nn
3593   { markdown/jekyllData }
3594   { }
3595 \ExplSyntaxOff
```

The option `jekyllDataRenderers=<key-values>` can be used to set the `<key-values>` for the module `markdown/jekyllData` without using the expl3 syntax.

```
3596 \ExplSyntaxOn
3597 \@@_with_various_cases:nn
3598   { jekyllDataRenderers }
3599   {
3600     \keys_define:nn
3601       { markdown/options }
3602       {
3603         #1 .code:n = {
3604           \tl_set:Nn
3605             \l_tmpa_tl
3606             { ##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.

```
3607         \tl_replace_all:NnV
3608         \l_tmpa_tl
3609         { / }
3610         \c_backslash_str
3611         \keys_set:nV
3612         { markdown/options/jekyll-data-renderers }
3613         \l_tmpa_tl
3614       },
3615     }
3616   }
3617 \keys_define:nn
3618   { markdown/options/jekyll-data-renderers }
3619   {
3620     unknown .code:n = {
3621       \tl_set_eq:NN
3622         \l_tmpa_tl
3623         \l_keys_key_str
3624       \tl_replace_all:NnVn
3625         \l_tmpa_tl
3626         \c_backslash_str

```



```

3627     { / }
3628     \tl_put_right:Nn
3629     \l_tmpa_tl
3630     {
3631         .code:n = { #1 }
3632     }
3633     \keys_define:nV
3634     { markdown/jekyllData }
3635     \l_tmpa_tl
3636 }
3637 }
3638 \ExplSyntaxOff

```

For complex YAML metadata, the option `jekyllDataKeyValue=<module>` [13] can be used to route the processing of all YAML metadata in the current  $\text{\TeX}$  group to the key-values from `<module>`.

### 2.2.6.2 Generating Plain $\text{\TeX}$ Token Renderer Prototype Macros and Key-Values

We define the command `\@@_define_renderer_prototypes:` that defines plain  $\text{\TeX}$  macros for token renderer prototypes. Furthermore, the `\markdownSetup` macro also accepts the `rendererPrototypes` and `unprotectedRendererPrototypes` keys. The value for these keys 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.

Whereas the key `rendererPrototypes` defines protected functions, which are usually preferable for typesetting, the key `unprotectedRendererPrototypes` defines unprotected functions, which are easier to expand and may be preferable for programming.

```

3639 \ExplSyntaxOn
3640 \cs_new:Nn \@@_define_renderer_prototypes:
3641 {
3642     \seq_map_inline:Nn
3643     \g_@@_renderers_seq
3644     {
3645         \@@_define_renderer_prototype:n
3646         { ##1 }
3647     }
3648 }
3649 \cs_new:Nn \@@_define_renderer_prototype:n
3650 {
3651     \@@_renderer_prototype_tl_to_csname:nN
3652     { #1 }
3653     \l_tmpa_tl
3654     \prop_get:NnN

```

```

3655     \g_@@_renderer_arities_prop
3656     { #1 }
3657     \l_tmpb_tl
3658     \@@_define_renderer_prototype:ncV
3659     { #1 }
3660     { \l_tmpa_tl }
3661     \l_tmpb_tl
3662   }
3663   \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
3664   {
3665     \tl_set:Nn
3666       \l_tmpa_tl
3667       { \str_uppercase:n { #1 } }
3668     \tl_set:Nx
3669       #2
3670       {
3671         markdownRenderer
3672         \tl_head:f { \l_tmpa_tl }
3673         \tl_tail:n { #1 }
3674         Prototype
3675       }
3676   }
3677   \tl_new:N
3678     \l_@@_renderer_prototype_definition_tl
3679   \bool_new:N
3680     \g_@@_appending_renderer_prototype_bool
3681   \bool_new:N
3682     \g_@@_unprotected_renderer_prototype_bool
3683   \cs_new:Nn \@@_define_renderer_prototype:nNn
3684   {
3685     \keys_define:nn
3686       { markdown/options/renderer-prototypes }
3687     {
3688       #1 .code:n = {
3689         \tl_set:Nn
3690           \l_@@_renderer_prototype_definition_tl
3691           { ##1 }
3692         \regex_replace_all:nnN
3693           { \cP\#0 }
3694           { #1 }
3695         \l_@@_renderer_prototype_definition_tl
3696         \bool_if:NT
3697           \g_@@_appending_renderer_prototype_bool
3698         {
3699           \@@_tl_set_from_cs:NNn
3700             \l_tmpa_tl
3701             #2

```

```

3702         { #3 }
3703         \tl_put_left:NV
3704         \l_@@_renderer_prototype_definition_tl
3705         \l_tmpa_tl
3706     }
3707     \bool_if:NTF
3708     \g_@@_unprotected_renderer_prototype_bool
3709     {
3710         \tl_set:Nn
3711         \l_tmpa_tl
3712         { \cs_set:Npn }
3713     }
3714     {
3715         \tl_set:Nn
3716         \l_tmpa_tl
3717         { \cs_set_protected:Npn }
3718     }
3719     \exp_last_unbraced:NNV
3720     \cs_generate_from_arg_count:NNnV
3721     #2
3722     \l_tmpa_tl
3723     { #3 }
3724     \l_@@_renderer_prototype_definition_tl
3725 },
3726 }

```

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

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

```

3727     \str_if_eq:nnF
3728     { #1 }
3729     { jekyllDataString }
3730     {
3731         \cs_if_free:NT
3732         #2
3733         {
3734             \cs_generate_from_arg_count:NNnn
3735             #2
3736             \cs_gset_protected:Npn
3737             { #3 }
3738             { }
3739         }
3740     }
3741 }
3742 \cs_generate_variant:Nn
3743 \@@_define_renderer_prototype:nNn

```

```
3744 { ncV }
```

The following example code showcases a possible configuration of the `\markdownRendererImageProto` 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.
  }
}
```

```
3745 \keys_define:nn
3746 { markdown/options/renderer-prototypes }
3747 {
3748   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 += {...},
        },
      },
    },
  },
}
```

```
}
```

```
3749      % TODO: Use `\\regex_if_match` in TeX Live 2025.
3750      \\regex_match:NVTF % noqa: w202
3751      \\c_@@_appending_key_regex
3752      \\l_keys_key_str
3753      {
3754          \\bool_gset_true:N
3755          \\g_@@_appending_renderer_prototype_bool
3756          \\tl_set:NV
3757          \\l_tmpa_tl
3758          \\l_keys_key_str
3759          \\regex_replace_once:NnN
3760          \\c_@@_appending_key_regex
3761          { }
3762          \\l_tmpa_tl
3763          \\tl_set:Nx
3764          \\l_tmpb_tl
3765          { { \\l_tmpa_tl } = }
3766          \\tl_put_right:Nn
3767          \\l_tmpb_tl
3768          { { #1 } }
3769          \\keys_set:nV
3770          { markdown/options/renderer-prototypes }
3771          \\l_tmpb_tl
3772          \\bool_gset_false:N
3773          \\g_@@_appending_renderer_prototype_bool
3774      }
```

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.
}

```

```

3775   {
3776     \@@_glob_seq:VnN
3777     \l_keys_key_str
3778     { g_@@_renderers_seq }
3779     \l_@@_renderer_glob_results_seq
3780   \seq_if_empty:NTF
3781     \l_@@_renderer_glob_results_seq
3782     {
3783       \msg_error:nnV
3784       { markdown }
3785       { undefined-renderer-prototype }
3786       \l_keys_key_str
3787     }
3788     {
3789       \tl_set:Nn
3790       \l_@@_renderer_glob_definition_tl
3791       { \exp_not:n { #1 } }
3792     \seq_map_inline:Nn
3793       \l_@@_renderer_glob_results_seq
3794       {
3795         \tl_set:Nn
3796         \l_tmpa_tl
3797         { { ##1 } = }
3798         \tl_put_right:Nx
3799         \l_tmpa_tl
3800         { { \l_@@_renderer_glob_definition_tl } }
3801       \keys_set:nV
3802         { markdown/options/renderer-prototypes }
3803         \l_tmpa_tl
3804       }
3805     }
3806   }

```

```

3807     },
3808   }
3809   \msg_new:nnn
3810   { markdown }
3811   { undefined-renderer-prototype }
3812   {
3813     Renderer~prototype~#1~is~undefined.
3814   }
3815   \@@_with_various_cases:nn
3816   { rendererPrototypes }
3817   {
3818     \keys_define:nn
3819     { markdown/options }
3820     {
3821       #1 .code:n = {
3822         \bool_gset_false:N
3823         \g_@@_unprotected_renderer_prototype_bool
3824         \keys_set:nn
3825         { markdown/options/renderer-prototypes }
3826         { ##1 }
3827       },
3828     }
3829   }
3830   \@@_with_various_cases:nn
3831   { unprotectedRendererPrototypes }
3832   {
3833     \keys_define:nn
3834     { markdown/options }
3835     {
3836       #1 .code:n = {
3837         \bool_gset_true:N
3838         \g_@@_unprotected_renderer_prototype_bool
3839         \keys_set:nn
3840         { markdown/options/renderer-prototypes }
3841         { ##1 }
3842       },
3843     }
3844   }

```

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

```

3845   \str_if_eq:VVT
3846   \c_@@_top_layer_tl
3847   \c_@@_option_layer_plain_tex_tl
3848   {
3849     \@@_define_renderer_prototypes:

```

```

3850 }
3851 \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 T<sub>E</sub>X engine that does not support direct Lua access is starting to buffer a text. The plain T<sub>E</sub>X implementation changes the category code of plain T<sub>E</sub>X special characters to other, but there may be other active characters that may break the output. This macro should temporarily change the category of these to *other*.

```

3852 \let\markdownMakeOther\relax

```

The `\markdownReadAndConvert` macro implements the `\markdownBegin` and `\yamlBegin` macros. The first argument specifies the token sequence that will terminate the markdown input when the plain T<sub>E</sub>X special characters have had their category changed to *other*: `\markdownEnd` for the `\markdownBegin` macro and `\yamlEnd` for the `\yamlBegin` macro. The second argument specifies the token sequence that will actually be inserted into the document, when the ending token sequence has been found.

```

3853 \let\markdownReadAndConvert\relax
3854 \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*.

```

3855 \catcode`\|=0\catcode`\=12%
3856 |gdef|markdownBegin{%
3857   |markdownReadAndConvert{\markdownEnd}%
3858                               {\|markdownEnd}}%
3859 |gdef|yamlBegin{%
3860   |begingroup
3861   |yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
3862   |markdownReadAndConvert{\yamlEnd}%
3863                               {\|yamlEnd}}%
3864 |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.

```

3865 \ExplSyntaxOn
3866 \keys_define:nn
3867   { markdown/options }
3868   {
3869     code .code:n = { #1 },
3870   }
3871 \ExplSyntaxOff

```

This can be especially useful in snippets.

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

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

To determine whether L<sup>A</sup>T<sub>E</sub>X is the top layer or if there are other layers above L<sup>A</sup>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 L<sup>A</sup>T<sub>E</sub>X is the top layer.

```

3872 \ExplSyntaxOn
3873 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
3874 \cs_generate_variant:Nn
3875   \tl_const:Nn
3876   { NV }
3877 \tl_if_exist:NF
3878   \c_@@_top_layer_tl
3879   {
3880     \tl_const:NV
3881       \c_@@_top_layer_tl
3882       \c_@@_option_layer_latex_tl
3883   }
3884 \ExplSyntaxOff
3885 \input markdown/markdown

```

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

```
\usepackage[⟨options⟩]{markdown}
```

where `⟨options⟩` are the L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.3). 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 2.3.4) 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<sub>ε</sub> parses package options.

### 2.3.1 Typesetting Markdown

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

#### 2.3.1.1 Typesetting Markdown and YAML directly

The `markdown` and `markdown*`  $\text{\LaTeX}$  environments are aliases for the macros `\markdownBegin` and `\markdownEnd` exposed by the plain  $\text{\TeX}$  interface.

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

```
3886 \newenvironment{markdown}\relax\relax
3887 \newenvironment{markdown*}[1]\relax\relax
```

Furthermore, both environments accept  $\text{\LaTeX}$  interface options (see Section 2.3.3) as the only argument. This argument is optional for the `markdown` environment and mandatory for the `markdown*` environment.

The `markdown` and `markdown*` environments are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

The following example  $\text{\LaTeX}$  code showcases the usage of the `markdown` and `markdown*` environments:

<code>\documentclass{article}</code>	<code>\documentclass{article}</code>
<code>\usepackage{markdown}</code>	<code>\usepackage{markdown}</code>
<code>\begin{document}</code>	<code>\begin{document}</code>
<code>\begin{markdown}[smartEllipses]</code>	<code>\begin{markdown*}{smartEllipses}</code>
<code>_Hello_ **world** ...</code>	<code>_Hello_ **world** ...</code>
<code>\end{markdown}</code>	<code>\end{markdown*}</code>
<code>\end{document}</code>	<code>\end{document}</code>

You can't directly extend the `markdown`  $\text{\LaTeX}$  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 before-text and must be followed by text that has not yet been ingested by TeX's input processor.

Furthermore, using the `\markdownEnd` macro in of after the replacement after-text 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 would be 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}
```

The `yaml` L<sup>A</sup>T<sub>E</sub>X environment is an alias for the macros `\yamlBegin` and `\yamlEnd` exposed by the plain TeX interface.

```
3888 \newenvironment{yaml}\relax\relax
```

Furthermore, the environment accepts L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.3) as the only optional argument.

The `yaml` environment is subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

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

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\begin{yaml}[smartEllipses]
title: _Hello_ **world** ...
author: John Doe
\end{yaml}
\end{document}
```

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

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\begin{markdown}[
  jekyllData,
  expectJekyllData,
  ensureJekyllData,
  smartEllipses,
]
title: _Hello_ **world** ...
author: John Doe
\end{markdown}
\end{document}
```

You can't directly extend the `yaml` L<sup>A</sup>T<sub>E</sub>X environment by using it in other environments. However, you can work around this limitation by using the `\yaml` and `\yamlEnd` macros directly in the definition, similarly to the `\markdown` and `\markdownEnd` macros described previously. Unlike with the `\markdown` and `\markdownEnd` macros, The `\yamlEnd` macro `_must_` be used in or after the replacement after-text.

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 T<sub>E</sub>X. Unlike the `\markinline` macro provided by the plain T<sub>E</sub>X interface, this macro also accepts L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.3) as its optional argument. These options will only influence this markdown content.

### 2.3.1.2 Typesetting Markdown and YAML from external documents

The `\markdownInput` macro accepts a single mandatory parameter containing the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain T<sub>E</sub>X. Unlike the `\markdownInput` macro provided by the plain T<sub>E</sub>X interface, this macro also accepts L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.3) as its optional argument. These options will only influence this markdown document.

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

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

```
\end{document}
```

The `\yamlInput` macro accepts a single mandatory parameter containing the filename of a YAML document and expands to the result of the conversion of the input YAML document to plain  $\text{\TeX}$ . Unlike the `\yamlInput` macro provided by the plain  $\text{\TeX}$  interface, this macro also accepts  $\text{\LaTeX}$  interface options (see Section 2.3.3) as its optional argument. These options will only influence this YAML document.

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

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\yamlInput[smartEllipses]{hello.yml}
\end{document}
```

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

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\markdownInput[
  jekyllData,
  expectJekyllData,
  ensureJekyllData,
  smartEllipses,
]{hello.yml}
\end{document}
```

### 2.3.2 Using $\text{\LaTeX}$ hooks with the Markdown package

$\text{\LaTeX}$  provides an intricate hook management system that allows users to insert extra material before and after certain  $\text{\TeX}$  macros and  $\text{\LaTeX}$  environments, among other things. [14, Section 3.1.2]

The Markdown package is compatible with hooks and allows the use of hooks to insert extra material before  $\text{\TeX}$  commands and before/after  $\text{\LaTeX}$  environments without restriction:

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\AddToHook{cmd/markdownRendererEmphasis/before}{emphasis: }
```

```

\AddToHook{env/markdown/before}{<markdown>}
\AddToHook{env/markdown/after}{</markdown>}
\begin{markdown}
foo _bar_ baz!
\end{markdown}
\end{document}

```

Processing the above example with L<sup>A</sup>T<sub>E</sub>X will produce the text “[markdown](#)foo emphasis: [\\_bar\\_ baz!](#)”, as expected.

However, using hooks to insert extra material after T<sub>E</sub>X commands only works for commands with a fixed number of parameters that don’t use currying.

If, in the above example, you explicitly defined the renderer for emphasis using `\markdownSetup` or another method that does not use currying, then you would be able to insert extra material even after the renderer:

```

\documentclass{article}
\usepackage{markdown}
\markdownSetup{renderers={emphasis={\emph{#1}}}}
\begin{document}
\AddToHook{cmd/markdownRendererEmphasis/before}{<emphasis>}
\AddToHook{cmd/markdownRendererEmphasis/after}{</emphasis>}
\AddToHook{env/markdown/before}{<markdown>}
\AddToHook{env/markdown/after}{</markdown>}
\begin{markdown}
foo _bar_ baz!
\end{markdown}
\end{document}

```

Processing the above example with L<sup>A</sup>T<sub>E</sub>X will produce the text “[markdown](#)foo [emphasis](#) [\\_bar\\_](#) [/emphasis](#) baz!”, as expected.

However, the default renderer for emphasis uses currying and calls the renderer prototype in a way that prevents the use of hooks to insert extra material after the renderer, see Section 2.2.5.12. In such a case, you would need to redefine the renderer in a way that does not use currying before you would be able to use hooks to insert extra material after it.

Hooks also cannot be used to insert extra material after renderers with a variable number of parameters such as the renderer for tables, see Section 2.2.5.39.

### 2.3.3 Options

The L<sup>A</sup>T<sub>E</sub>X options are represented by a comma-delimited list of  $\langle key \rangle = \langle value \rangle$  pairs. For boolean options, the  $= \langle value \rangle$  part is optional, and  $\langle key \rangle$  will be interpreted as  $\langle key \rangle = \text{true}$  if the  $= \langle value \rangle$  part has been omitted.

$\text{\LaTeX}$  options map directly to the options recognized by the plain  $\text{\TeX}$  interface (see Section 2.2.2) and to the markdown token renderers and their prototypes recognized by the plain  $\text{\TeX}$  interface (see Sections 2.2.5 and 2.2.6).

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

### 2.3.3.1 Finalizing and Freezing the Cache

To ensure compatibility with the `minted` package [15, Section 5.1], which supports the `finalizcache` and `frozenscache` package options with similar semantics to the `finalizeCache` and `frozenCache` plain  $\text{\TeX}$  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  $\text{\LaTeX}$  document sources for distribution.

```
3889 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
3890 \DeclareOption{frozenscache}{\markdownSetup{frozenCache}}
```

### 2.3.3.2 Generating Plain $\text{\TeX}$ Option, Token Renderer, and Token Renderer Prototype Macros and Key-Values

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

```
3891 \ExplSyntaxOn
3892 \str_if_eq:VVT
3893   \c_@@_top_layer_tl
3894   \c_@@_option_layer_latex_tl
3895   {
3896     \@@define_option_commands_and_keyvals:
3897     \@@define_renderers:
3898     \@@define_renderer_prototypes:
3899   }
3900 \ExplSyntaxOff
```

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

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

### 2.3.4 Themes

In Section 2.2.3, we described the concept of themes. In  $\text{\LaTeX}$ , we expand on the concept of themes by allowing a theme to be a full-blown  $\text{\LaTeX}$  package. Specifically, the key-values `theme=<theme name>` and `import=<theme name>` load a  $\text{\LaTeX}$  package named `markdowntheme<munged theme name>.sty` if it exists and a  $\text{\TeX}$  document named `markdowntheme<munged theme name>.tex` otherwise.

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

If the  $\text{\LaTeX}$  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  $\text{\LaTeX}$  package has been loaded. Otherwise, the theme(s) will be loaded immediately. For example, the following code would first load the Markdown package, then the theme `witiko/example/foo`, and finally the theme `witiko/example/bar`:

```
\usepackage[
  import=witiko/example/foo,
  import=witiko/example/bar,
]{markdown}
```

```
3901 \newif\ifmarkdownLaTeXLoaded
3902 \markdownLaTeXLoadedfalse
```

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

We also define the prop `\g_@@_latex_built_in_themes_prop` that contains the code of built-in themes. This is a packaging optimization, so that built-in themes does not need to be distributed in many small files.



```

3903 \ExplSyntaxOn
3904 \prop_new:N
3905   \g_@@_latex_built_in_themes_prop
3906 \ExplSyntaxOff

```

Built-in  $\text{\LaTeX}$  themes provided with the Markdown package include:

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

```

3907 \AtEndOfPackage{\markdownLaTeXLoadedtrue}

```

At the end of the  $\text{\LaTeX}$  module, we load the **witiko/markdown/defaults**  $\text{\LaTeX}$  theme (see Section 2.2.3) with the default definitions for token renderer prototypes unless the option **noDefaults** has been enabled (see Section 2.2.2.3).

```

3908 \ExplSyntaxOn
3909 \str_if_eq:VVT
3910   \c_@@_top_layer_tl
3911   \c_@@_option_layer_latex_tl
3912   {
3913     \use:c
3914       { ExplSyntaxOff }
3915     \AtEndOfPackage
3916       {
3917         \@@_if_option:nF
3918           { noDefaults }
3919           {
3920             \@@_if_option:nTF
3921               { experimental }
3922               {
3923                 \@@_setup:n
3924                   { theme = witiko/markdown/defaults@experimental }
3925               }
3926               {
3927                 \@@_setup:n
3928                   { theme = witiko/markdown/defaults }
3929               }
3930           }
3931       }
3932     \use:c
3933       { ExplSyntaxOn }
3934   }
3935 \ExplSyntaxOff

```

Please, see Section 3.3.2 for implementation details of the built-in  $\text{\LaTeX}$  themes.

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

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

```
3936 \ExplSyntaxOn
3937 \tl_const:Nn \c_@@_option_layer_context_tl { context }
3938 \cs_generate_variant:Nn
3939   \tl_const:Nn
3940   { NV }
3941 \tl_if_exist:NF
3942   \c_@@_top_layer_tl
3943   {
3944     \tl_const:NV
3945       \c_@@_top_layer_tl
3946       \c_@@_option_layer_context_tl
3947   }
3948 \ExplSyntaxOff
```

The ConT<sub>E</sub>Xt interface provides a start-stop macro pair for the typesetting of markdown input from within ConT<sub>E</sub>Xt and facilities for setting Lua, plain T<sub>E</sub>X, and ConT<sub>E</sub>Xt options used during the conversion from markdown to plain T<sub>E</sub>X. The rest of the interface is inherited from the plain T<sub>E</sub>X interface (see Section 2.2).

```
3949 \writestatus{loading}{ConTEXt User Module / markdown}%
3950 \startmodule[markdown]
3951 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
3952   \do\#\do\^\do\_do\%do\~}%
3953 \input markdown/markdown
```

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

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

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

### 2.4.1 Typesetting Markdown and YAML

The interface exposes the `\startmarkdown`, `\stopmarkdown`, `\startyaml`, `\stopyaml`, `\inputmarkdown`, and `\inputyaml` macros.

#### 2.4.1.1 Typesetting Markdown and YAML directly

The `\startmarkdown` and `\stopmarkdown` macros are aliases for the macros `\markdownBegin` and `\markdownEnd` exposed by the plain T<sub>E</sub>X interface.

```
3954 \let\startmarkdown\relax
3955 \let\stopmarkdown\relax
```

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

The macros `\startmarkdown` and `\stopmarkdown` are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

The following example ConTeXt code showcases the usage of the `\startmarkdown` and `\stopmarkdown` macros:

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

The `\startyaml` and `\stopyaml` macros are aliases for the macros `\yamlBegin` and `\yamlEnd` exposed by the plain TeX interface.

```
3956 \let\startyaml\relax
3957 \let\stopyaml\relax
```

You may prepend your own code to the `\startyaml` macro and append your own code to the `\stopyaml` macro to produce special effects before and after the YAML document.

The macros `\startyaml` and `\stopyaml` are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

The following example ConTeXt code showcases the usage of the `\startyaml` and `\stopyaml` macros:

```
\usemodule[t] [markdown]
\starttext
\startyaml
title: _Hello_ **world** ...
author: John Doe
\stopyaml
\stoptext
```

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

```
\usemodule[t] [markdown]
\starttext
\setupyaml[jekyllData, expectJekyllData, ensureJekyllData]
\startyaml
```

```
title: _Hello_ **world** ...
author: John Doe
\stopyaml
\stoptext
```

#### 2.4.1.2 Typesetting Markdown and YAML from external documents

The `\inputmarkdown` macro aliases the macro `\markdownInput` exposed by the plain  $\text{\TeX}$  interface.

```
3958 \let\inputmarkdown\relax
```

Furthermore, the `\inputmarkdown` macro also accepts Con $\text{\TeX}$ t interface options (see Section 2.4.2) as its optional argument. These options will only influence this markdown document.

The following example Con $\text{\TeX}$ t code showcases the usage of the `\inputmarkdown` macro:

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

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

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

The `\inputyaml` macro aliases the macro `\yamlInput` exposed by the plain  $\text{\TeX}$  interface.

```
3959 \let\inputyaml\relax
```

Furthermore, the `\inputyaml` macro also accepts Con $\text{\TeX}$ t interface options (see Section 2.4.2) as its optional argument. These options will only influence this YAML document.

The following example Con $\text{\TeX}$ t code showcases the usage of the `\inputyaml` macro:

```
\usemodule[t][markdown]
\starttext
\inputyaml[smartEllipses]{hello.yml}
\stoptext
```

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

```
\usemodule[t] [markdown]
\starttext
\setupyaml[smartEllipses]
\inputyaml{hello.yaml}
\stoptext
```

## 2.4.2 Options

The ConT<sub>E</sub>Xt 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.

ConT<sub>E</sub>Xt options map directly to the options recognized by the plain T<sub>E</sub>X interface (see Section 2.2.2).

The ConT<sub>E</sub>Xt 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}`.

```
3960 \ExplSyntaxOn
3961 \cs_new:Npn
3962   \setupmarkdown
3963   [ #1 ]
3964   {
3965     \@@_setup:n
3966     { #1 }
3967   }
```

The command `\setupyaml` is also available as an alias for the command `\setupmarkdown`.

```
3968 \cs_gset_eq:NN
3969   \setupyaml
3970   \setupmarkdown
```

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

Unlike plain T<sub>E</sub>X, we also accept caseless variants of options in line with the style of ConT<sub>E</sub>Xt.

```
3971 \cs_new:Nn \@@_caseless:N % noqa: w401
3972   {
3973     \regex_replace_all:nnN
3974     { ([a-z])([A-Z]) }
3975     { \1 \c { str_lowercase:n } \cB\{ \2 \cE\} }
3976     #1
3977     \tl_set:Nx
3978     #1
```

```

3979      { #1 }
3980    }
3981    \seq_gput_right:Nn \g_@@_cases_seq { @@_caseless:N }

```

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

```

3982 \str_if_eq:VVT
3983   \c_@@_top_layer_tl
3984   \c_@@_option_layer_context_tl
3985   {
3986     \@@_define_option_commands_and_keyvals:
3987     \@@_define_renderers:
3988     \@@_define_renderer_prototypes:
3989   }

```

### 2.4.3 Themes

In Section 2.2.3, we described the concept of themes. In ConT<sub>E</sub>Xt, we expand on the concept of themes by allowing a theme to be a full-blown ConT<sub>E</sub>Xt module. Specifically, the key-values `theme=<theme name>` and `import=<theme name>` load a ConT<sub>E</sub>Xt module named `t-markdowntheme<munged theme name>.tex` 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 ConT<sub>E</sub>Xt-specific `t-*.tex` 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 `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]

```

We also define the prop `\g_@@_context_built_in_themes_prop` that contains the code of built-in themes. This is a packaging optimization, so that built-in themes does not need to be distributed in many small files.

```

3990 \prop_new:N
3991   \g_@@_context_built_in_themes_prop
3992 \ExplSyntaxOff

```

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.

```
3993 \startmodule[markdownthemewitiko_markdown_defaults]
3994 \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).

```
3995 local upper, format, length =
3996   string.upper, string.format, string.len
3997 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, Cp, any =
3998   lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
3999   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.

```
4000 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.

```
4001 function util.err(msg, exit_code)
4002   io.stderr:write("markdown.lua: " .. msg .. "\n")
4003   os.exit(exit_code or 1)
4004 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 and the result of `transform(string)` is returned as the second return value in case it's useful to the caller. Regardless, the pathname is always returned as the first return value.

```
4005 function util.cache(dir, string, salt, transform, suffix)
4006   local digest = md5.sumhexa(string .. (salt or ""))
4007   local name = util.pathname(dir, digest .. suffix)
4008   local file = io.open(name, "r")
4009   local result = nil
4010   if file == nil then -- If no cache entry exists, create a new one.
4011     file = assert(io.open(name, "w"),
4012       [[Could not open file ]] .. name .. [[ for writing]])
4013     result = string
4014     if transform ~= nil then
4015       result = transform(result)
4016     end
4017     assert(file:write(result))
4018     assert(file:close())
4019   end
4020   return name, result
4021 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.

```
4022 function util.cache_verbatim(dir, string)
4023   local name = util.cache(dir, string, nil, nil, ".verbatim")
4024   return name
4025 end
```

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

```
4026 function util.table_copy(t)
4027   local u = { }
4028   for k, v in pairs(t) do u[k] = v end
4029   return setmetatable(u, getmetatable(t))
4030 end
```

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

```
4031 function util.encode_json_string(s)
4032   s = s:gsub([[\\]], [[\\]])
```



```

4033  s = s:gsub([""], [{"\"}])
4034  return [{""] .. s .. [{""]}
4035 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 [16, Chapter 21].

```

4036 function util.expand_tabs_in_line(s, tabstop)
4037   local tab = tabstop or 4
4038   local corr = 0
4039   return (s:gsub(")\t", function(p)
4040     local sp = tab - (p - 1 + corr) % tab
4041     corr = corr - 1 + sp
4042     return string.rep(" ", sp)
4043   end))
4044 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.

```

4045 function util.walk(t, f)
4046   local typ = type(t)
4047   if typ == "string" then
4048     f(t)
4049   elseif typ == "table" then
4050     local i = 1
4051     local n
4052     n = t[i]
4053     while n do
4054       util.walk(n, f)
4055       i = i + 1
4056       n = t[i]
4057     end
4058   elseif typ == "function" then
4059     local ok, val = pcall(t)
4060     if ok then
4061       util.walk(val, f)
4062     end
4063   else
4064     f(tostring(t))
4065   end
4066 end

```

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

```

4067 function util.flatten(ary)
4068   local new = {}
4069   for _,v in ipairs(ary) do
4070     if type(v) == "table" then
4071       for _,w in ipairs(util.flatten(v)) do
4072         new[#new + 1] = w
4073       end
4074     else
4075       new[#new + 1] = v
4076     end
4077   end
4078   return new
4079 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.

```

4080 function util.rope_to_string(rope)
4081   local buffer = {}
4082   util.walk(rope, function(x) buffer[#buffer + 1] = x end)
4083   return table.concat(buffer)
4084 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.

```

4085 function util.rope_last(rope)
4086   if #rope == 0 then
4087     return nil
4088   else
4089     local l = rope[#rope]
4090     if type(l) == "table" then
4091       return util.rope_last(l)
4092     else
4093       return l
4094     end
4095   end
4096 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  $1 \leq i \leq \#ary$ .

```

4097 function util.intersperse(ary, x)
4098   local new = {}
4099   local l = #ary
4100   for i,v in ipairs(ary) do
4101     local n = #new
4102     new[n + 1] = v
4103     if i ~= l then
4104       new[n + 2] = x

```

```

4105     end
4106   end
4107   return new
4108 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  $1 \leq i \leq \#ary$ .

```

4109 function util.map(ary, f)
4110   local new = {}
4111   for i,v in ipairs(ary) do
4112     new[i] = f(v)
4113   end
4114   return new
4115 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.

```

4116 function util.escaper(char_escapes, string_escapes)

```

Build a string of escapable characters.

```

4117   local char_escapes_list = ""
4118   for i,_ in pairs(char_escapes) do
4119     char_escapes_list = char_escapes_list .. i
4120   end

```

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

```

4121   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.

```

4122   if string_escapes then
4123     for k,v in pairs(string_escapes) do
4124       escapable = P(k) / v + escapable
4125     end
4126   end

```

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

```
4127 local escape_string = Cs((escapable + any)^0)
```

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

```
4128 return function(s)
4129     return lpeg.match(escape_string, s)
4130 end
4131 end
```

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

```
4132 function util.pathname(dir, file)
4133     if #dir == 0 then
4134         return file
4135     else
4136         return dir .. "/" .. file
4137     end
4138 end
```

The `util.salt` method produces cryptographic salt out of a table of options `options`.

```
4139 function util.salt(options)
4140     local opt_string = {}
4141     for k, _ in pairs(defaultOptions) do
4142         local v = options[k]
4143         if type(v) == "table" then
4144             for _, i in ipairs(v) do
4145                 opt_string[#opt_string+1] = k .. "=" .. tostring(i)
4146             end
4147         end
4148     end
```

The `cacheDir` option is disregarded.

```
4147     elseif k ~= "cacheDir" then
4148         opt_string[#opt_string+1] = k .. "=" .. tostring(v)
4149     end
4150 end
4151 table.sort(opt_string)
4152 local salt = table.concat(opt_string, ",")
4153 .. "," .. metadata.version
4154 return salt
4155 end
```

The `util.warning` method produces a warning `s` that is unrelated to any specific markdown text being processed. For warnings that are specific to a markdown text, use `writer->warning` function.

```
4156 function util.warning(s)
4157     io.stderr:write("Warning: " .. s .. "\n")
4158 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.

```
4159 local entities = {}
4160
4161 local character_entities = {
4162     ["Tab"] = 9,
4163     ["NewLine"] = 10,
4164     ["excl"] = 33,
4165     ["QUOT"] = 34,
4166     ["quot"] = 34,
4167     ["num"] = 35,
4168     ["dollar"] = 36,
4169     ["percent"] = 37,
4170     ["AMP"] = 38,
4171     ["amp"] = 38,
4172     ["apos"] = 39,
4173     ["lpar"] = 40,
4174     ["rpar"] = 41,
4175     ["ast"] = 42,
4176     ["midast"] = 42,
4177     ["plus"] = 43,
4178     ["comma"] = 44,
4179     ["period"] = 46,
4180     ["sol"] = 47,
4181     ["colon"] = 58,
4182     ["semi"] = 59,
4183     ["LT"] = 60,
4184     ["lt"] = 60,
4185     ["nvlt"] = {60, 8402},
4186     ["bne"] = {61, 8421},
4187     ["equals"] = 61,
4188     ["GT"] = 62,
4189     ["gt"] = 62,
4190     ["nvgt"] = {62, 8402},
4191     ["quest"] = 63,
4192     ["commat"] = 64,
4193     ["lbrack"] = 91,
4194     ["lsqb"] = 91,
4195     ["bsol"] = 92,
4196     ["rbrack"] = 93,
4197     ["rsqb"] = 93,
4198     ["Hat"] = 94,
4199     ["UnderBar"] = 95,
4200     ["lowbar"] = 95,
```

```

4201 ["DiacriticalGrave"] = 96,
4202 ["grave"] = 96,
4203 ["fjlig"] = {102, 106},
4204 ["lbrace"] = 123,
4205 ["lcub"] = 123,
4206 ["VerticalLine"] = 124,
4207 ["verbar"] = 124,
4208 ["vert"] = 124,
4209 ["rbrace"] = 125,
4210 ["rcub"] = 125,
4211 ["NonBreakingSpace"] = 160,
4212 ["nbsp"] = 160,
4213 ["iexcl"] = 161,
4214 ["cent"] = 162,
4215 ["pound"] = 163,
4216 ["curren"] = 164,
4217 ["yen"] = 165,
4218 ["brvbar"] = 166,
4219 ["sect"] = 167,
4220 ["Dot"] = 168,
4221 ["DoubleDot"] = 168,
4222 ["die"] = 168,
4223 ["uml"] = 168,
4224 ["COPY"] = 169,
4225 ["copy"] = 169,
4226 ["ordf"] = 170,
4227 ["laquo"] = 171,
4228 ["not"] = 172,
4229 ["shy"] = 173,
4230 ["REG"] = 174,
4231 ["circledR"] = 174,
4232 ["reg"] = 174,
4233 ["macr"] = 175,
4234 ["strns"] = 175,
4235 ["deg"] = 176,
4236 ["PlusMinus"] = 177,
4237 ["plusmn"] = 177,
4238 ["pm"] = 177,
4239 ["sup2"] = 178,
4240 ["sup3"] = 179,
4241 ["DiacriticalAcute"] = 180,
4242 ["acute"] = 180,
4243 ["micro"] = 181,
4244 ["para"] = 182,
4245 ["CenterDot"] = 183,
4246 ["centerdot"] = 183,
4247 ["middot"] = 183,

```

4248 ["Cedilla"] = 184,  
 4249 ["cedil"] = 184,  
 4250 ["sup1"] = 185,  
 4251 ["ordm"] = 186,  
 4252 ["raquo"] = 187,  
 4253 ["frac14"] = 188,  
 4254 ["frac12"] = 189,  
 4255 ["half"] = 189,  
 4256 ["frac34"] = 190,  
 4257 ["iquest"] = 191,  
 4258 ["Agrave"] = 192,  
 4259 ["Aacute"] = 193,  
 4260 ["Acirc"] = 194,  
 4261 ["Atilde"] = 195,  
 4262 ["Auml"] = 196,  
 4263 ["Aring"] = 197,  
 4264 ["angst"] = 197,  
 4265 ["AElig"] = 198,  
 4266 ["Ccedil"] = 199,  
 4267 ["Egrave"] = 200,  
 4268 ["Eacute"] = 201,  
 4269 ["Ecirc"] = 202,  
 4270 ["Euml"] = 203,  
 4271 ["Igrave"] = 204,  
 4272 ["Iacute"] = 205,  
 4273 ["Icirc"] = 206,  
 4274 ["Iuml"] = 207,  
 4275 ["ETH"] = 208,  
 4276 ["Ntilde"] = 209,  
 4277 ["Ograve"] = 210,  
 4278 ["Oacute"] = 211,  
 4279 ["Ocirc"] = 212,  
 4280 ["Otilde"] = 213,  
 4281 ["Ouml"] = 214,  
 4282 ["times"] = 215,  
 4283 ["Oslash"] = 216,  
 4284 ["Ugrave"] = 217,  
 4285 ["Uacute"] = 218,  
 4286 ["Ucirc"] = 219,  
 4287 ["Uuml"] = 220,  
 4288 ["Yacute"] = 221,  
 4289 ["THORN"] = 222,  
 4290 ["szlig"] = 223,  
 4291 ["agrave"] = 224,  
 4292 ["aacute"] = 225,  
 4293 ["acirc"] = 226,  
 4294 ["atilde"] = 227,

```

4295 ["auml"] = 228,
4296 ["aring"] = 229,
4297 ["aelig"] = 230,
4298 ["ccedil"] = 231,
4299 ["egrave"] = 232,
4300 ["eacute"] = 233,
4301 ["ecirc"] = 234,
4302 ["euml"] = 235,
4303 ["igrave"] = 236,
4304 ["iacute"] = 237,
4305 ["icirc"] = 238,
4306 ["iuml"] = 239,
4307 ["eth"] = 240,
4308 ["ntilde"] = 241,
4309 ["ograve"] = 242,
4310 ["oacute"] = 243,
4311 ["ocirc"] = 244,
4312 ["otilde"] = 245,
4313 ["ouml"] = 246,
4314 ["div"] = 247,
4315 ["divide"] = 247,
4316 ["oslash"] = 248,
4317 ["ugrave"] = 249,
4318 ["uacute"] = 250,
4319 ["ucirc"] = 251,
4320 ["uuml"] = 252,
4321 ["yacute"] = 253,
4322 ["thorn"] = 254,
4323 ["yuml"] = 255,
4324 ["Amacr"] = 256,
4325 ["amacr"] = 257,
4326 ["Abreve"] = 258,
4327 ["abreve"] = 259,
4328 ["Aogon"] = 260,
4329 ["aogon"] = 261,
4330 ["Cacute"] = 262,
4331 ["cacute"] = 263,
4332 ["Ccirc"] = 264,
4333 ["ccirc"] = 265,
4334 ["Cdot"] = 266,
4335 ["cdot"] = 267,
4336 ["Ccaron"] = 268,
4337 ["ccaron"] = 269,
4338 ["Dcaron"] = 270,
4339 ["dcaron"] = 271,
4340 ["Dstrok"] = 272,
4341 ["dstrok"] = 273,

```



```

4342 ["Emacr"] = 274,
4343 ["emacr"] = 275,
4344 ["Edot"] = 278,
4345 ["edot"] = 279,
4346 ["Eogon"] = 280,
4347 ["eogon"] = 281,
4348 ["Ecaron"] = 282,
4349 ["ecaron"] = 283,
4350 ["Gcirc"] = 284,
4351 ["gcirc"] = 285,
4352 ["Gbreve"] = 286,
4353 ["gbreve"] = 287,
4354 ["Gdot"] = 288,
4355 ["gdot"] = 289,
4356 ["Gcedil"] = 290,
4357 ["Hcirc"] = 292,
4358 ["hcirc"] = 293,
4359 ["Hstrokr"] = 294,
4360 ["hstrokr"] = 295,
4361 ["Itilde"] = 296,
4362 ["itilde"] = 297,
4363 ["Imacr"] = 298,
4364 ["imacr"] = 299,
4365 ["Iogon"] = 302,
4366 ["iogon"] = 303,
4367 ["Idot"] = 304,
4368 ["imath"] = 305,
4369 ["inodot"] = 305,
4370 ["IJlig"] = 306,
4371 ["ijlig"] = 307,
4372 ["Jcirc"] = 308,
4373 ["jcirc"] = 309,
4374 ["Kcedil"] = 310,
4375 ["kcedil"] = 311,
4376 ["kgreen"] = 312,
4377 ["Lacute"] = 313,
4378 ["lacute"] = 314,
4379 ["Lcedil"] = 315,
4380 ["lcedil"] = 316,
4381 ["Lcaron"] = 317,
4382 ["lcaron"] = 318,
4383 ["Lmidot"] = 319,
4384 ["lmidot"] = 320,
4385 ["Lstrokr"] = 321,
4386 ["lstrokr"] = 322,
4387 ["Nacute"] = 323,
4388 ["nacute"] = 324,

```

4389 ["Ncedil"] = 325,  
 4390 ["ncedil"] = 326,  
 4391 ["Ncaron"] = 327,  
 4392 ["ncaron"] = 328,  
 4393 ["napos"] = 329,  
 4394 ["ENG"] = 330,  
 4395 ["eng"] = 331,  
 4396 ["Omacr"] = 332,  
 4397 ["omacr"] = 333,  
 4398 ["Odblac"] = 336,  
 4399 ["odblac"] = 337,  
 4400 ["OElig"] = 338,  
 4401 ["oelig"] = 339,  
 4402 ["Racute"] = 340,  
 4403 ["racute"] = 341,  
 4404 ["Rcedil"] = 342,  
 4405 ["rcedil"] = 343,  
 4406 ["Rcaron"] = 344,  
 4407 ["rcaron"] = 345,  
 4408 ["Sacute"] = 346,  
 4409 ["sacute"] = 347,  
 4410 ["Scirc"] = 348,  
 4411 ["scirc"] = 349,  
 4412 ["Scedil"] = 350,  
 4413 ["scedil"] = 351,  
 4414 ["Scaron"] = 352,  
 4415 ["scaron"] = 353,  
 4416 ["Tcedil"] = 354,  
 4417 ["tcedil"] = 355,  
 4418 ["Tcaron"] = 356,  
 4419 ["tcaron"] = 357,  
 4420 ["Tstrok"] = 358,  
 4421 ["tstrok"] = 359,  
 4422 ["Utilde"] = 360,  
 4423 ["utilde"] = 361,  
 4424 ["Umacr"] = 362,  
 4425 ["umacr"] = 363,  
 4426 ["Ubreve"] = 364,  
 4427 ["ubreve"] = 365,  
 4428 ["Uring"] = 366,  
 4429 ["uring"] = 367,  
 4430 ["Udblac"] = 368,  
 4431 ["udblac"] = 369,  
 4432 ["Uogon"] = 370,  
 4433 ["uogon"] = 371,  
 4434 ["Wcirc"] = 372,  
 4435 ["wcirc"] = 373,

4436 ["Ycirc"] = 374,  
 4437 ["ycirc"] = 375,  
 4438 ["Yuml"] = 376,  
 4439 ["Zacute"] = 377,  
 4440 ["zacute"] = 378,  
 4441 ["Zdot"] = 379,  
 4442 ["zdot"] = 380,  
 4443 ["Zcaron"] = 381,  
 4444 ["zcaron"] = 382,  
 4445 ["fnof"] = 402,  
 4446 ["imped"] = 437,  
 4447 ["gacute"] = 501,  
 4448 ["jmath"] = 567,  
 4449 ["circ"] = 710,  
 4450 ["Hacek"] = 711,  
 4451 ["caron"] = 711,  
 4452 ["Breve"] = 728,  
 4453 ["breve"] = 728,  
 4454 ["DiacriticalDot"] = 729,  
 4455 ["dot"] = 729,  
 4456 ["ring"] = 730,  
 4457 ["ogon"] = 731,  
 4458 ["DiacriticalTilde"] = 732,  
 4459 ["tilde"] = 732,  
 4460 ["DiacriticalDoubleAcute"] = 733,  
 4461 ["dblac"] = 733,  
 4462 ["DownBreve"] = 785,  
 4463 ["Alpha"] = 913,  
 4464 ["Beta"] = 914,  
 4465 ["Gamma"] = 915,  
 4466 ["Delta"] = 916,  
 4467 ["Epsilon"] = 917,  
 4468 ["Zeta"] = 918,  
 4469 ["Eta"] = 919,  
 4470 ["Theta"] = 920,  
 4471 ["Iota"] = 921,  
 4472 ["Kappa"] = 922,  
 4473 ["Lambda"] = 923,  
 4474 ["Mu"] = 924,  
 4475 ["Nu"] = 925,  
 4476 ["Xi"] = 926,  
 4477 ["Omicron"] = 927,  
 4478 ["Pi"] = 928,  
 4479 ["Rho"] = 929,  
 4480 ["Sigma"] = 931,  
 4481 ["Tau"] = 932,  
 4482 ["Upsilon"] = 933,

```

4483 ["Phi"] = 934,
4484 ["Chi"] = 935,
4485 ["Psi"] = 936,
4486 ["Omega"] = 937,
4487 ["ohm"] = 937,
4488 ["alpha"] = 945,
4489 ["beta"] = 946,
4490 ["gamma"] = 947,
4491 ["delta"] = 948,
4492 ["epsi"] = 949,
4493 ["epsilon"] = 949,
4494 ["zeta"] = 950,
4495 ["eta"] = 951,
4496 ["theta"] = 952,
4497 ["iota"] = 953,
4498 ["kappa"] = 954,
4499 ["lambda"] = 955,
4500 ["mu"] = 956,
4501 ["nu"] = 957,
4502 ["xi"] = 958,
4503 ["omicron"] = 959,
4504 ["pi"] = 960,
4505 ["rho"] = 961,
4506 ["sigmaf"] = 962,
4507 ["sigmav"] = 962,
4508 ["varsigma"] = 962,
4509 ["sigma"] = 963,
4510 ["tau"] = 964,
4511 ["upsi"] = 965,
4512 ["upsilon"] = 965,
4513 ["phi"] = 966,
4514 ["chi"] = 967,
4515 ["psi"] = 968,
4516 ["omega"] = 969,
4517 ["thetasym"] = 977,
4518 ["thetav"] = 977,
4519 ["vartheta"] = 977,
4520 ["Upsi"] = 978,
4521 ["upsih"] = 978,
4522 ["phiv"] = 981,
4523 ["straightphi"] = 981,
4524 ["varphi"] = 981,
4525 ["piv"] = 982,
4526 ["varpi"] = 982,
4527 ["Gammad"] = 988,
4528 ["digamma"] = 989,
4529 ["gammad"] = 989,

```

```

4530 ["kappav"] = 1008,
4531 ["varkappa"] = 1008,
4532 ["rhov"] = 1009,
4533 ["varrho"] = 1009,
4534 ["epsiv"] = 1013,
4535 ["straightepsilon"] = 1013,
4536 ["varepsilon"] = 1013,
4537 ["backepsilon"] = 1014,
4538 ["bepsi"] = 1014,
4539 ["IOcy"] = 1025,
4540 ["DJcy"] = 1026,
4541 ["GJcy"] = 1027,
4542 ["Jukcy"] = 1028,
4543 ["DScy"] = 1029,
4544 ["Iukcy"] = 1030,
4545 ["YIcy"] = 1031,
4546 ["Jsercy"] = 1032,
4547 ["LJcy"] = 1033,
4548 ["NJcy"] = 1034,
4549 ["TSHcy"] = 1035,
4550 ["KJcy"] = 1036,
4551 ["Ubrcy"] = 1038,
4552 ["DZcy"] = 1039,
4553 ["Acy"] = 1040,
4554 ["Bcy"] = 1041,
4555 ["Vcy"] = 1042,
4556 ["Gcy"] = 1043,
4557 ["Dcy"] = 1044,
4558 ["IEcy"] = 1045,
4559 ["ZHcy"] = 1046,
4560 ["Zcy"] = 1047,
4561 ["Icy"] = 1048,
4562 ["Jcy"] = 1049,
4563 ["Kcy"] = 1050,
4564 ["Lcy"] = 1051,
4565 ["Mcy"] = 1052,
4566 ["Ncy"] = 1053,
4567 ["Ocy"] = 1054,
4568 ["Pcy"] = 1055,
4569 ["Rcy"] = 1056,
4570 ["Scy"] = 1057,
4571 ["Tcy"] = 1058,
4572 ["Ucy"] = 1059,
4573 ["Fcy"] = 1060,
4574 ["KHcy"] = 1061,
4575 ["TScy"] = 1062,
4576 ["CHcy"] = 1063,

```

```

4577 ["SHcy"] = 1064,
4578 ["SHCHcy"] = 1065,
4579 ["HARDcy"] = 1066,
4580 ["Ycy"] = 1067,
4581 ["SOFTcy"] = 1068,
4582 ["Ecy"] = 1069,
4583 ["YUcy"] = 1070,
4584 ["YAcy"] = 1071,
4585 ["acy"] = 1072,
4586 ["bcy"] = 1073,
4587 ["vcy"] = 1074,
4588 ["gcy"] = 1075,
4589 ["dcy"] = 1076,
4590 ["iecy"] = 1077,
4591 ["zhcy"] = 1078,
4592 ["zcy"] = 1079,
4593 ["icy"] = 1080,
4594 ["jcy"] = 1081,
4595 ["kcy"] = 1082,
4596 ["lcy"] = 1083,
4597 ["mcy"] = 1084,
4598 ["ncy"] = 1085,
4599 ["ocy"] = 1086,
4600 ["pcy"] = 1087,
4601 ["rcy"] = 1088,
4602 ["scy"] = 1089,
4603 ["tcy"] = 1090,
4604 ["ucy"] = 1091,
4605 ["fcy"] = 1092,
4606 ["khcy"] = 1093,
4607 ["tscy"] = 1094,
4608 ["chcy"] = 1095,
4609 ["shcy"] = 1096,
4610 ["shchcy"] = 1097,
4611 ["hardcy"] = 1098,
4612 ["ycy"] = 1099,
4613 ["softcy"] = 1100,
4614 ["ecy"] = 1101,
4615 ["yucy"] = 1102,
4616 ["yacy"] = 1103,
4617 ["iocy"] = 1105,
4618 ["djcy"] = 1106,
4619 ["gjcy"] = 1107,
4620 ["jukcy"] = 1108,
4621 ["dscy"] = 1109,
4622 ["iukcy"] = 1110,
4623 ["yicy"] = 1111,

```

```

4624 ["jsercy"] = 1112,
4625 ["ljcy"] = 1113,
4626 ["njcy"] = 1114,
4627 ["tshcy"] = 1115,
4628 ["kjcy"] = 1116,
4629 ["ubrcy"] = 1118,
4630 ["dzcy"] = 1119,
4631 ["ensp"] = 8194,
4632 ["emsp"] = 8195,
4633 ["emsp13"] = 8196,
4634 ["emsp14"] = 8197,
4635 ["numsp"] = 8199,
4636 ["puncsp"] = 8200,
4637 ["ThinSpace"] = 8201,
4638 ["thinsp"] = 8201,
4639 ["VeryThinSpace"] = 8202,
4640 ["hairsp"] = 8202,
4641 ["NegativeMediumSpace"] = 8203,
4642 ["NegativeThickSpace"] = 8203,
4643 ["NegativeThinSpace"] = 8203,
4644 ["NegativeVeryThinSpace"] = 8203,
4645 ["ZeroWidthSpace"] = 8203,
4646 ["zwnj"] = 8204,
4647 ["zwj"] = 8205,
4648 ["lrm"] = 8206,
4649 ["rlm"] = 8207,
4650 ["dash"] = 8208,
4651 ["hyphen"] = 8208,
4652 ["ndash"] = 8211,
4653 ["mdash"] = 8212,
4654 ["horbar"] = 8213,
4655 ["Verbar"] = 8214,
4656 ["Vert"] = 8214,
4657 ["OpenCurlyQuote"] = 8216,
4658 ["lsquo"] = 8216,
4659 ["CloseCurlyQuote"] = 8217,
4660 ["rsquo"] = 8217,
4661 ["rsquor"] = 8217,
4662 ["lsquor"] = 8218,
4663 ["sbquo"] = 8218,
4664 ["OpenCurlyDoubleQuote"] = 8220,
4665 ["ldquo"] = 8220,
4666 ["CloseCurlyDoubleQuote"] = 8221,
4667 ["rdquo"] = 8221,
4668 ["rdquor"] = 8221,
4669 ["bdquo"] = 8222,
4670 ["ldquor"] = 8222,

```

```

4671 ["dagger"] = 8224,
4672 ["Dagger"] = 8225,
4673 ["ddagger"] = 8225,
4674 ["bull"] = 8226,
4675 ["bullet"] = 8226,
4676 ["nldr"] = 8229,
4677 ["hellip"] = 8230,
4678 ["mldr"] = 8230,
4679 ["permil"] = 8240,
4680 ["pertenk"] = 8241,
4681 ["prime"] = 8242,
4682 ["Prime"] = 8243,
4683 ["tprime"] = 8244,
4684 ["backprime"] = 8245,
4685 ["bprime"] = 8245,
4686 ["lsaquo"] = 8249,
4687 ["rsaquo"] = 8250,
4688 ["OverBar"] = 8254,
4689 ["oline"] = 8254,
4690 ["caret"] = 8257,
4691 ["hybull"] = 8259,
4692 ["frasl"] = 8260,
4693 ["bsemi"] = 8271,
4694 ["qprime"] = 8279,
4695 ["MediumSpace"] = 8287,
4696 ["ThickSpace"] = {8287, 8202},
4697 ["NoBreak"] = 8288,
4698 ["ApplyFunction"] = 8289,
4699 ["af"] = 8289,
4700 ["InvisibleTimes"] = 8290,
4701 ["it"] = 8290,
4702 ["InvisibleComma"] = 8291,
4703 ["ic"] = 8291,
4704 ["euro"] = 8364,
4705 ["TripleDot"] = 8411,
4706 ["tdot"] = 8411,
4707 ["DotDot"] = 8412,
4708 ["Copf"] = 8450,
4709 ["complexes"] = 8450,
4710 ["incare"] = 8453,
4711 ["gscr"] = 8458,
4712 ["HilbertSpace"] = 8459,
4713 ["Hscr"] = 8459,
4714 ["hamilt"] = 8459,
4715 ["Hfr"] = 8460,
4716 ["Poincareplane"] = 8460,
4717 ["Hopf"] = 8461,

```



```

4718 ["quaternions"] = 8461,
4719 ["planckh"] = 8462,
4720 ["hbar"] = 8463,
4721 ["hslash"] = 8463,
4722 ["planck"] = 8463,
4723 ["plankv"] = 8463,
4724 ["Iscr"] = 8464,
4725 ["imagline"] = 8464,
4726 ["Ifr"] = 8465,
4727 ["Im"] = 8465,
4728 ["image"] = 8465,
4729 ["imagpart"] = 8465,
4730 ["Laplacetrfr"] = 8466,
4731 ["Lscr"] = 8466,
4732 ["lagran"] = 8466,
4733 ["ell"] = 8467,
4734 ["Nopf"] = 8469,
4735 ["naturals"] = 8469,
4736 ["numero"] = 8470,
4737 ["copysr"] = 8471,
4738 ["weierp"] = 8472,
4739 ["wp"] = 8472,
4740 ["Popf"] = 8473,
4741 ["primes"] = 8473,
4742 ["Qopf"] = 8474,
4743 ["rationals"] = 8474,
4744 ["Rscr"] = 8475,
4745 ["realine"] = 8475,
4746 ["Re"] = 8476,
4747 ["Rfr"] = 8476,
4748 ["real"] = 8476,
4749 ["realpart"] = 8476,
4750 ["Ropf"] = 8477,
4751 ["reals"] = 8477,
4752 ["rx"] = 8478,
4753 ["TRADE"] = 8482,
4754 ["trade"] = 8482,
4755 ["Zopf"] = 8484,
4756 ["integers"] = 8484,
4757 ["mho"] = 8487,
4758 ["Zfr"] = 8488,
4759 ["zeetrf"] = 8488,
4760 ["iiota"] = 8489,
4761 ["Bernoullis"] = 8492,
4762 ["Bscr"] = 8492,
4763 ["bernou"] = 8492,
4764 ["Cayleys"] = 8493,

```

```

4765 ["Cfr"] = 8493,
4766 ["escr"] = 8495,
4767 ["Escr"] = 8496,
4768 ["expectation"] = 8496,
4769 ["Fouriertrf"] = 8497,
4770 ["Fscr"] = 8497,
4771 ["Mellintrf"] = 8499,
4772 ["Mscr"] = 8499,
4773 ["phmmat"] = 8499,
4774 ["order"] = 8500,
4775 ["orderof"] = 8500,
4776 ["oscr"] = 8500,
4777 ["alefsym"] = 8501,
4778 ["aleph"] = 8501,
4779 ["beth"] = 8502,
4780 ["gimel"] = 8503,
4781 ["daleth"] = 8504,
4782 ["CapitalDifferentialD"] = 8517,
4783 ["DD"] = 8517,
4784 ["DifferentialD"] = 8518,
4785 ["dd"] = 8518,
4786 ["ExponentialE"] = 8519,
4787 ["ee"] = 8519,
4788 ["exponentiale"] = 8519,
4789 ["ImaginaryI"] = 8520,
4790 ["ii"] = 8520,
4791 ["frac13"] = 8531,
4792 ["frac23"] = 8532,
4793 ["frac15"] = 8533,
4794 ["frac25"] = 8534,
4795 ["frac35"] = 8535,
4796 ["frac45"] = 8536,
4797 ["frac16"] = 8537,
4798 ["frac56"] = 8538,
4799 ["frac18"] = 8539,
4800 ["frac38"] = 8540,
4801 ["frac58"] = 8541,
4802 ["frac78"] = 8542,
4803 ["LeftArrow"] = 8592,
4804 ["ShortLeftArrow"] = 8592,
4805 ["larr"] = 8592,
4806 ["leftarrow"] = 8592,
4807 ["slarr"] = 8592,
4808 ["ShortUpArrow"] = 8593,
4809 ["UpArrow"] = 8593,
4810 ["uarr"] = 8593,
4811 ["uparrow"] = 8593,

```

```

4812 ["RightArrow"] = 8594,
4813 ["ShortRightArrow"] = 8594,
4814 ["rarr"] = 8594,
4815 ["rightarrow"] = 8594,
4816 ["srarr"] = 8594,
4817 ["DownArrow"] = 8595,
4818 ["ShortDownArrow"] = 8595,
4819 ["darr"] = 8595,
4820 ["downarrow"] = 8595,
4821 ["LeftRightArrow"] = 8596,
4822 ["harr"] = 8596,
4823 ["leftrightarrow"] = 8596,
4824 ["UpDownArrow"] = 8597,
4825 ["updownarrow"] = 8597,
4826 ["varr"] = 8597,
4827 ["UpperLeftArrow"] = 8598,
4828 ["nwarr"] = 8598,
4829 ["nwarrow"] = 8598,
4830 ["UpperRightArrow"] = 8599,
4831 ["nearr"] = 8599,
4832 ["nearrow"] = 8599,
4833 ["LowerRightArrow"] = 8600,
4834 ["searr"] = 8600,
4835 ["searrow"] = 8600,
4836 ["LowerLeftArrow"] = 8601,
4837 ["swarr"] = 8601,
4838 ["swarrow"] = 8601,
4839 ["nlarr"] = 8602,
4840 ["nleftarrow"] = 8602,
4841 ["nrarr"] = 8603,
4842 ["nrightarrow"] = 8603,
4843 ["nrarrw"] = {8605, 824},
4844 ["rarrw"] = 8605,
4845 ["rightsquigarrow"] = 8605,
4846 ["Larr"] = 8606,
4847 ["twoheadleftarrow"] = 8606,
4848 ["Uarr"] = 8607,
4849 ["Rarr"] = 8608,
4850 ["twoheadrightarrow"] = 8608,
4851 ["Darr"] = 8609,
4852 ["larrtl"] = 8610,
4853 ["leftarrowtail"] = 8610,
4854 ["rarrtl"] = 8611,
4855 ["rightarrowtail"] = 8611,
4856 ["LeftTeeArrow"] = 8612,
4857 ["mapstoleft"] = 8612,
4858 ["UpTeeArrow"] = 8613,

```

```

4859 ["mapstoup"] = 8613,
4860 ["RightTeeArrow"] = 8614,
4861 ["map"] = 8614,
4862 ["mapsto"] = 8614,
4863 ["DownTeeArrow"] = 8615,
4864 ["mapstodown"] = 8615,
4865 ["hookleftarrow"] = 8617,
4866 ["larrhk"] = 8617,
4867 ["hookrightarrow"] = 8618,
4868 ["rarrhk"] = 8618,
4869 ["larrlp"] = 8619,
4870 ["looparrowleft"] = 8619,
4871 ["looparrowright"] = 8620,
4872 ["rarrlp"] = 8620,
4873 ["harrw"] = 8621,
4874 ["leftrightsquigarrow"] = 8621,
4875 ["nharr"] = 8622,
4876 ["nletrightarrow"] = 8622,
4877 ["Lsh"] = 8624,
4878 ["lsh"] = 8624,
4879 ["Rsh"] = 8625,
4880 ["rsh"] = 8625,
4881 ["ldsh"] = 8626,
4882 ["rdsh"] = 8627,
4883 ["crarr"] = 8629,
4884 ["cularr"] = 8630,
4885 ["curvearrowleft"] = 8630,
4886 ["curarr"] = 8631,
4887 ["curvearrowright"] = 8631,
4888 ["circlearrowleft"] = 8634,
4889 ["olarr"] = 8634,
4890 ["circlearrowright"] = 8635,
4891 ["orarr"] = 8635,
4892 ["LeftVector"] = 8636,
4893 ["leftharpoonup"] = 8636,
4894 ["lharu"] = 8636,
4895 ["DownLeftVector"] = 8637,
4896 ["leftharpoondown"] = 8637,
4897 ["lhard"] = 8637,
4898 ["RightUpVector"] = 8638,
4899 ["uharr"] = 8638,
4900 ["upharpoonright"] = 8638,
4901 ["LeftUpVector"] = 8639,
4902 ["uharl"] = 8639,
4903 ["upharpoonleft"] = 8639,
4904 ["RightVector"] = 8640,
4905 ["rharu"] = 8640,

```

```

4906 ["rightharpoonup"] = 8640,
4907 ["DownRightVector"] = 8641,
4908 ["rhard"] = 8641,
4909 ["rightharpoondown"] = 8641,
4910 ["RightDownVector"] = 8642,
4911 ["dharr"] = 8642,
4912 ["downharpoonright"] = 8642,
4913 ["LeftDownVector"] = 8643,
4914 ["dharl"] = 8643,
4915 ["downharpoonleft"] = 8643,
4916 ["RightArrowLeftArrow"] = 8644,
4917 ["rightleftarrows"] = 8644,
4918 ["rlarr"] = 8644,
4919 ["UpArrowDownArrow"] = 8645,
4920 ["udarr"] = 8645,
4921 ["LeftArrowRightArrow"] = 8646,
4922 ["leftrightarrows"] = 8646,
4923 ["lrarr"] = 8646,
4924 ["leftleftarrows"] = 8647,
4925 ["llarr"] = 8647,
4926 ["upuparrows"] = 8648,
4927 ["uuarr"] = 8648,
4928 ["rightrightarrows"] = 8649,
4929 ["rrarr"] = 8649,
4930 ["ddarr"] = 8650,
4931 ["downdownarrows"] = 8650,
4932 ["ReverseEquilibrium"] = 8651,
4933 ["leftrightharpoons"] = 8651,
4934 ["lrhar"] = 8651,
4935 ["Equilibrium"] = 8652,
4936 ["rightleftharpoons"] = 8652,
4937 ["rlhar"] = 8652,
4938 ["nLeftarrow"] = 8653,
4939 ["nLArr"] = 8653,
4940 ["nLeftrightarrow"] = 8654,
4941 ["nhArr"] = 8654,
4942 ["nRightarrow"] = 8655,
4943 ["nrArr"] = 8655,
4944 ["DoubleLeftArrow"] = 8656,
4945 ["Leftarrow"] = 8656,
4946 ["lArr"] = 8656,
4947 ["DoubleUpArrow"] = 8657,
4948 ["Uparrow"] = 8657,
4949 ["uArr"] = 8657,
4950 ["DoubleRightArrow"] = 8658,
4951 ["Implies"] = 8658,
4952 ["Rightarrow"] = 8658,

```

```

4953 ["rArr"] = 8658,
4954 ["DoubleDownArrow"] = 8659,
4955 ["Downarrow"] = 8659,
4956 ["dArr"] = 8659,
4957 ["DoubleLeftRightArrow"] = 8660,
4958 ["Leftrightarrow"] = 8660,
4959 ["hArr"] = 8660,
4960 ["iff"] = 8660,
4961 ["DoubleUpDownArrow"] = 8661,
4962 ["Updownarrow"] = 8661,
4963 ["vArr"] = 8661,
4964 ["nwArr"] = 8662,
4965 ["neArr"] = 8663,
4966 ["seArr"] = 8664,
4967 ["swArr"] = 8665,
4968 ["Lleftarrow"] = 8666,
4969 ["lAarr"] = 8666,
4970 ["Rrightarrow"] = 8667,
4971 ["rAarr"] = 8667,
4972 ["zigrarr"] = 8669,
4973 ["LeftArrowBar"] = 8676,
4974 ["larrb"] = 8676,
4975 ["RightArrowBar"] = 8677,
4976 ["rarrb"] = 8677,
4977 ["DownArrowUpArrow"] = 8693,
4978 ["duarr"] = 8693,
4979 ["loarr"] = 8701,
4980 ["roarr"] = 8702,
4981 ["hoarr"] = 8703,
4982 ["ForAll"] = 8704,
4983 ["forall"] = 8704,
4984 ["comp"] = 8705,
4985 ["complement"] = 8705,
4986 ["PartialD"] = 8706,
4987 ["npart"] = {8706, 824},
4988 ["part"] = 8706,
4989 ["Exists"] = 8707,
4990 ["exist"] = 8707,
4991 ["NotExists"] = 8708,
4992 ["nexist"] = 8708,
4993 ["nexists"] = 8708,
4994 ["empty"] = 8709,
4995 ["emptyset"] = 8709,
4996 ["emptyv"] = 8709,
4997 ["varnothing"] = 8709,
4998 ["Del"] = 8711,
4999 ["nabla"] = 8711,

```

```

5000 ["Element"] = 8712,
5001 ["in"] = 8712,
5002 ["isin"] = 8712,
5003 ["isinv"] = 8712,
5004 ["NotElement"] = 8713,
5005 ["notin"] = 8713,
5006 ["notinva"] = 8713,
5007 ["ReverseElement"] = 8715,
5008 ["SuchThat"] = 8715,
5009 ["ni"] = 8715,
5010 ["niv"] = 8715,
5011 ["NotReverseElement"] = 8716,
5012 ["notni"] = 8716,
5013 ["notniva"] = 8716,
5014 ["Product"] = 8719,
5015 ["prod"] = 8719,
5016 ["Coproduct"] = 8720,
5017 ["coprod"] = 8720,
5018 ["Sum"] = 8721,
5019 ["sum"] = 8721,
5020 ["minus"] = 8722,
5021 ["MinusPlus"] = 8723,
5022 ["mnplus"] = 8723,
5023 ["mp"] = 8723,
5024 ["dotplus"] = 8724,
5025 ["plusdo"] = 8724,
5026 ["Backslash"] = 8726,
5027 ["setminus"] = 8726,
5028 ["setmn"] = 8726,
5029 ["smallsetminus"] = 8726,
5030 ["ssetmn"] = 8726,
5031 ["lowast"] = 8727,
5032 ["SmallCircle"] = 8728,
5033 ["compfn"] = 8728,
5034 ["Sqrt"] = 8730,
5035 ["radic"] = 8730,
5036 ["Proportional"] = 8733,
5037 ["prop"] = 8733,
5038 ["propto"] = 8733,
5039 ["varpropto"] = 8733,
5040 ["vprop"] = 8733,
5041 ["infin"] = 8734,
5042 ["angrt"] = 8735,
5043 ["ang"] = 8736,
5044 ["angle"] = 8736,
5045 ["nang"] = {8736, 8402},
5046 ["angmsd"] = 8737,

```

```

5047 ["measuredangle"] = 8737,
5048 ["angsph"] = 8738,
5049 ["VerticalBar"] = 8739,
5050 ["mid"] = 8739,
5051 ["shortmid"] = 8739,
5052 ["smid"] = 8739,
5053 ["NotVerticalBar"] = 8740,
5054 ["nmid"] = 8740,
5055 ["nshortmid"] = 8740,
5056 ["nsmid"] = 8740,
5057 ["DoubleVerticalBar"] = 8741,
5058 ["par"] = 8741,
5059 ["parallel"] = 8741,
5060 ["shortparallel"] = 8741,
5061 ["spar"] = 8741,
5062 ["NotDoubleVerticalBar"] = 8742,
5063 ["npar"] = 8742,
5064 ["nparallel"] = 8742,
5065 ["nshortparallel"] = 8742,
5066 ["nspar"] = 8742,
5067 ["and"] = 8743,
5068 ["wedge"] = 8743,
5069 ["or"] = 8744,
5070 ["vee"] = 8744,
5071 ["cap"] = 8745,
5072 ["caps"] = {8745, 65024},
5073 ["cup"] = 8746,
5074 ["cups"] = {8746, 65024},
5075 ["Integral"] = 8747,
5076 ["int"] = 8747,
5077 ["Int"] = 8748,
5078 ["iiint"] = 8749,
5079 ["tint"] = 8749,
5080 ["ContourIntegral"] = 8750,
5081 ["conint"] = 8750,
5082 ["oint"] = 8750,
5083 ["Conint"] = 8751,
5084 ["DoubleContourIntegral"] = 8751,
5085 ["Cconint"] = 8752,
5086 ["cwint"] = 8753,
5087 ["ClockwiseContourIntegral"] = 8754,
5088 ["cwconint"] = 8754,
5089 ["CounterClockwiseContourIntegral"] = 8755,
5090 ["awconint"] = 8755,
5091 ["Therefore"] = 8756,
5092 ["there4"] = 8756,
5093 ["therefore"] = 8756,

```



```

5094 ["Because"] = 8757,
5095 ["becaus"] = 8757,
5096 ["because"] = 8757,
5097 ["ratio"] = 8758,
5098 ["Colon"] = 8759,
5099 ["Proportion"] = 8759,
5100 ["dotminus"] = 8760,
5101 ["minusd"] = 8760,
5102 ["mDDot"] = 8762,
5103 ["homtht"] = 8763,
5104 ["Tilde"] = 8764,
5105 ["nvsim"] = {8764, 8402},
5106 ["sim"] = 8764,
5107 ["thicksim"] = 8764,
5108 ["thksim"] = 8764,
5109 ["backsim"] = 8765,
5110 ["bsim"] = 8765,
5111 ["race"] = {8765, 817},
5112 ["ac"] = 8766,
5113 ["acE"] = {8766, 819},
5114 ["mstpos"] = 8766,
5115 ["acd"] = 8767,
5116 ["VerticalTilde"] = 8768,
5117 ["wr"] = 8768,
5118 ["wreath"] = 8768,
5119 ["NotTilde"] = 8769,
5120 ["nsim"] = 8769,
5121 ["EqualTilde"] = 8770,
5122 ["NotEqualTilde"] = {8770, 824},
5123 ["eqsim"] = 8770,
5124 ["esim"] = 8770,
5125 ["nesim"] = {8770, 824},
5126 ["TildeEqual"] = 8771,
5127 ["sime"] = 8771,
5128 ["simeq"] = 8771,
5129 ["NotTildeEqual"] = 8772,
5130 ["nsime"] = 8772,
5131 ["nsimeq"] = 8772,
5132 ["TildeFullEqual"] = 8773,
5133 ["cong"] = 8773,
5134 ["simne"] = 8774,
5135 ["NotTildeFullEqual"] = 8775,
5136 ["ncong"] = 8775,
5137 ["TildeTilde"] = 8776,
5138 ["ap"] = 8776,
5139 ["approx"] = 8776,
5140 ["asymp"] = 8776,

```

```

5141 ["thickapprox"] = 8776,
5142 ["thkap"] = 8776,
5143 ["NotTildeTilde"] = 8777,
5144 ["nap"] = 8777,
5145 ["napprox"] = 8777,
5146 ["ape"] = 8778,
5147 ["approxeq"] = 8778,
5148 ["apid"] = 8779,
5149 ["napid"] = {8779, 824},
5150 ["backcong"] = 8780,
5151 ["bcong"] = 8780,
5152 ["CupCap"] = 8781,
5153 ["asympeq"] = 8781,
5154 ["nvap"] = {8781, 8402},
5155 ["Bumpeq"] = 8782,
5156 ["HumpDownHump"] = 8782,
5157 ["NotHumpDownHump"] = {8782, 824},
5158 ["bump"] = 8782,
5159 ["nbump"] = {8782, 824},
5160 ["HumpEqual"] = 8783,
5161 ["NotHumpEqual"] = {8783, 824},
5162 ["bumpe"] = 8783,
5163 ["bumpeq"] = 8783,
5164 ["nbumpe"] = {8783, 824},
5165 ["DotEqual"] = 8784,
5166 ["doteq"] = 8784,
5167 ["esdot"] = 8784,
5168 ["nedot"] = {8784, 824},
5169 ["doteqdot"] = 8785,
5170 ["eDot"] = 8785,
5171 ["efDot"] = 8786,
5172 ["fallingdotseq"] = 8786,
5173 ["erDot"] = 8787,
5174 ["risingdotseq"] = 8787,
5175 ["Assign"] = 8788,
5176 ["colone"] = 8788,
5177 ["coloneq"] = 8788,
5178 ["ecolon"] = 8789,
5179 ["eqcolon"] = 8789,
5180 ["ecir"] = 8790,
5181 ["eqcirc"] = 8790,
5182 ["circeq"] = 8791,
5183 ["cire"] = 8791,
5184 ["wedgeq"] = 8793,
5185 ["veeeq"] = 8794,
5186 ["triangleq"] = 8796,
5187 ["trie"] = 8796,

```

```

5188 ["equest"] = 8799,
5189 ["questeq"] = 8799,
5190 ["NotEqual"] = 8800,
5191 ["ne"] = 8800,
5192 ["Congruent"] = 8801,
5193 ["bnequiv"] = {8801, 8421},
5194 ["equiv"] = 8801,
5195 ["NotCongruent"] = 8802,
5196 ["nequiv"] = 8802,
5197 ["le"] = 8804,
5198 ["leq"] = 8804,
5199 ["nvle"] = {8804, 8402},
5200 ["GreaterEqual"] = 8805,
5201 ["ge"] = 8805,
5202 ["geq"] = 8805,
5203 ["nvge"] = {8805, 8402},
5204 ["LessFullEqual"] = 8806,
5205 ["lE"] = 8806,
5206 ["leqq"] = 8806,
5207 ["nlE"] = {8806, 824},
5208 ["nleqq"] = {8806, 824},
5209 ["GreaterFullEqual"] = 8807,
5210 ["NotGreaterFullEqual"] = {8807, 824},
5211 ["gE"] = 8807,
5212 ["geqq"] = 8807,
5213 ["ngE"] = {8807, 824},
5214 ["ngeqq"] = {8807, 824},
5215 ["lnE"] = 8808,
5216 ["lneqq"] = 8808,
5217 ["lvertneqq"] = {8808, 65024},
5218 ["lvnE"] = {8808, 65024},
5219 ["gnE"] = 8809,
5220 ["gneqq"] = 8809,
5221 ["gvertneqq"] = {8809, 65024},
5222 ["gvnE"] = {8809, 65024},
5223 ["Lt"] = 8810,
5224 ["NestedLessLess"] = 8810,
5225 ["NotLessLess"] = {8810, 824},
5226 ["ll"] = 8810,
5227 ["nLt"] = {8810, 8402},
5228 ["nLtv"] = {8810, 824},
5229 ["Gt"] = 8811,
5230 ["NestedGreaterGreater"] = 8811,
5231 ["NotGreaterGreater"] = {8811, 824},
5232 ["gg"] = 8811,
5233 ["nGt"] = {8811, 8402},
5234 ["nGtv"] = {8811, 824},

```

```

5235 ["between"] = 8812,
5236 ["twixt"] = 8812,
5237 ["NotCupCap"] = 8813,
5238 ["NotLess"] = 8814,
5239 ["nless"] = 8814,
5240 ["nlt"] = 8814,
5241 ["NotGreater"] = 8815,
5242 ["ngt"] = 8815,
5243 ["ngtr"] = 8815,
5244 ["NotLessEqual"] = 8816,
5245 ["nle"] = 8816,
5246 ["nleq"] = 8816,
5247 ["NotGreaterEqual"] = 8817,
5248 ["nge"] = 8817,
5249 ["ngeq"] = 8817,
5250 ["LessTilde"] = 8818,
5251 ["lesssim"] = 8818,
5252 ["lsim"] = 8818,
5253 ["GreaterTilde"] = 8819,
5254 ["gsim"] = 8819,
5255 ["gtrsim"] = 8819,
5256 ["NotLessTilde"] = 8820,
5257 ["nlsim"] = 8820,
5258 ["NotGreaterTilde"] = 8821,
5259 ["ngsim"] = 8821,
5260 ["LessGreater"] = 8822,
5261 ["lessgtr"] = 8822,
5262 ["lg"] = 8822,
5263 ["GreaterLess"] = 8823,
5264 ["gl"] = 8823,
5265 ["gtrless"] = 8823,
5266 ["NotLessGreater"] = 8824,
5267 ["ntlg"] = 8824,
5268 ["NotGreaterLess"] = 8825,
5269 ["ntgl"] = 8825,
5270 ["Precedes"] = 8826,
5271 ["pr"] = 8826,
5272 ["prec"] = 8826,
5273 ["Succeeds"] = 8827,
5274 ["sc"] = 8827,
5275 ["succ"] = 8827,
5276 ["PrecedesSlantEqual"] = 8828,
5277 ["prcue"] = 8828,
5278 ["preccurlyeq"] = 8828,
5279 ["SucceedsSlantEqual"] = 8829,
5280 ["sccue"] = 8829,
5281 ["succcurlyeq"] = 8829,

```

```

5282 ["PrecedesTilde"] = 8830,
5283 ["precsim"] = 8830,
5284 ["prsim"] = 8830,
5285 ["NotSucceedsTilde"] = {8831, 824},
5286 ["SucceedsTilde"] = 8831,
5287 ["scsim"] = 8831,
5288 ["succsim"] = 8831,
5289 ["NotPrecedes"] = 8832,
5290 ["npr"] = 8832,
5291 ["nprec"] = 8832,
5292 ["NotSucceeds"] = 8833,
5293 ["nsc"] = 8833,
5294 ["nsucc"] = 8833,
5295 ["NotSubset"] = {8834, 8402},
5296 ["nsubset"] = {8834, 8402},
5297 ["sub"] = 8834,
5298 ["subset"] = 8834,
5299 ["vnsup"] = {8834, 8402},
5300 ["NotSuperset"] = {8835, 8402},
5301 ["Superset"] = 8835,
5302 ["nsupset"] = {8835, 8402},
5303 ["sup"] = 8835,
5304 ["supset"] = 8835,
5305 ["vnsup"] = {8835, 8402},
5306 ["nsub"] = 8836,
5307 ["nsup"] = 8837,
5308 ["SubsetEqual"] = 8838,
5309 ["sube"] = 8838,
5310 ["subseteq"] = 8838,
5311 ["SupersetEqual"] = 8839,
5312 ["supe"] = 8839,
5313 ["supseteq"] = 8839,
5314 ["NotSubsetEqual"] = 8840,
5315 ["nsube"] = 8840,
5316 ["nsubseteq"] = 8840,
5317 ["NotSupersetEqual"] = 8841,
5318 ["nsupe"] = 8841,
5319 ["nsupseteq"] = 8841,
5320 ["subne"] = 8842,
5321 ["subsetneq"] = 8842,
5322 ["varsubsetneq"] = {8842, 65024},
5323 ["vsubne"] = {8842, 65024},
5324 ["supne"] = 8843,
5325 ["supsetneq"] = 8843,
5326 ["varsupsetneq"] = {8843, 65024},
5327 ["vsupne"] = {8843, 65024},
5328 ["cupdot"] = 8845,

```

```

5329 ["UnionPlus"] = 8846,
5330 ["uplus"] = 8846,
5331 ["NotSquareSubset"] = {8847, 824},
5332 ["SquareSubset"] = 8847,
5333 ["sqsub"] = 8847,
5334 ["sqsubset"] = 8847,
5335 ["NotSquareSuperset"] = {8848, 824},
5336 ["SquareSuperset"] = 8848,
5337 ["sqsup"] = 8848,
5338 ["sqsupset"] = 8848,
5339 ["SquareSubsetEqual"] = 8849,
5340 ["sqsube"] = 8849,
5341 ["sqsubseteq"] = 8849,
5342 ["SquareSupersetEqual"] = 8850,
5343 ["sqsupe"] = 8850,
5344 ["sqsupseteq"] = 8850,
5345 ["SquareIntersection"] = 8851,
5346 ["sqcap"] = 8851,
5347 ["sqcaps"] = {8851, 65024},
5348 ["SquareUnion"] = 8852,
5349 ["sqcup"] = 8852,
5350 ["sqcups"] = {8852, 65024},
5351 ["CirclePlus"] = 8853,
5352 ["oplus"] = 8853,
5353 ["CircleMinus"] = 8854,
5354 ["ominus"] = 8854,
5355 ["CircleTimes"] = 8855,
5356 ["otimes"] = 8855,
5357 ["osol"] = 8856,
5358 ["CircleDot"] = 8857,
5359 ["odot"] = 8857,
5360 ["circledcirc"] = 8858,
5361 ["ocir"] = 8858,
5362 ["circledast"] = 8859,
5363 ["oast"] = 8859,
5364 ["circleddash"] = 8861,
5365 ["odash"] = 8861,
5366 ["boxplus"] = 8862,
5367 ["plusb"] = 8862,
5368 ["boxminus"] = 8863,
5369 ["minusb"] = 8863,
5370 ["boxtimes"] = 8864,
5371 ["timesb"] = 8864,
5372 ["dotsquare"] = 8865,
5373 ["sdotb"] = 8865,
5374 ["RightTee"] = 8866,
5375 ["vdash"] = 8866,

```

```

5376 ["LeftTee"] = 8867,
5377 ["dashv"] = 8867,
5378 ["DownTee"] = 8868,
5379 ["top"] = 8868,
5380 ["UpTee"] = 8869,
5381 ["bot"] = 8869,
5382 ["bottom"] = 8869,
5383 ["perp"] = 8869,
5384 ["models"] = 8871,
5385 ["DoubleRightTee"] = 8872,
5386 ["vDash"] = 8872,
5387 ["Vdash"] = 8873,
5388 ["Vvdash"] = 8874,
5389 ["VDash"] = 8875,
5390 ["nvdash"] = 8876,
5391 ["nvDash"] = 8877,
5392 ["nVdash"] = 8878,
5393 ["nVDash"] = 8879,
5394 ["prurel"] = 8880,
5395 ["LeftTriangle"] = 8882,
5396 ["vartriangleleft"] = 8882,
5397 ["vltri"] = 8882,
5398 ["RightTriangle"] = 8883,
5399 ["vartriangleright"] = 8883,
5400 ["vrtri"] = 8883,
5401 ["LeftTriangleEqual"] = 8884,
5402 ["ltrie"] = 8884,
5403 ["nvltrie"] = {8884, 8402},
5404 ["trianglelefteq"] = 8884,
5405 ["RightTriangleEqual"] = 8885,
5406 ["nvrtrie"] = {8885, 8402},
5407 ["rtrie"] = 8885,
5408 ["trianglerighteq"] = 8885,
5409 ["origof"] = 8886,
5410 ["imof"] = 8887,
5411 ["multimap"] = 8888,
5412 ["mumap"] = 8888,
5413 ["hercon"] = 8889,
5414 ["intcal"] = 8890,
5415 ["intercal"] = 8890,
5416 ["veebar"] = 8891,
5417 ["barvee"] = 8893,
5418 ["angrtvb"] = 8894,
5419 ["lrtri"] = 8895,
5420 ["Wedge"] = 8896,
5421 ["bigwedge"] = 8896,
5422 ["xwedge"] = 8896,

```

```

5423 ["Vee"] = 8897,
5424 ["bigvee"] = 8897,
5425 ["xvee"] = 8897,
5426 ["Intersection"] = 8898,
5427 ["bigcap"] = 8898,
5428 ["xcap"] = 8898,
5429 ["Union"] = 8899,
5430 ["bigcup"] = 8899,
5431 ["xcup"] = 8899,
5432 ["Diamond"] = 8900,
5433 ["diam"] = 8900,
5434 ["diamond"] = 8900,
5435 ["sdot"] = 8901,
5436 ["Star"] = 8902,
5437 ["sstarf"] = 8902,
5438 ["divideontimes"] = 8903,
5439 ["divonx"] = 8903,
5440 ["bowtie"] = 8904,
5441 ["ltimes"] = 8905,
5442 ["rtimes"] = 8906,
5443 ["leftthreetimes"] = 8907,
5444 ["lthree"] = 8907,
5445 ["rightthreetimes"] = 8908,
5446 ["rthree"] = 8908,
5447 ["backsimeq"] = 8909,
5448 ["bsime"] = 8909,
5449 ["curlyvee"] = 8910,
5450 ["cuvee"] = 8910,
5451 ["curlywedge"] = 8911,
5452 ["cuwed"] = 8911,
5453 ["Sub"] = 8912,
5454 ["Subset"] = 8912,
5455 ["Sup"] = 8913,
5456 ["Supset"] = 8913,
5457 ["Cap"] = 8914,
5458 ["Cup"] = 8915,
5459 ["fork"] = 8916,
5460 ["pitchfork"] = 8916,
5461 ["epar"] = 8917,
5462 ["lessdot"] = 8918,
5463 ["ltdot"] = 8918,
5464 ["gtdot"] = 8919,
5465 ["gtrdot"] = 8919,
5466 ["Ll"] = 8920,
5467 ["nLl"] = {8920, 824},
5468 ["Gg"] = 8921,
5469 ["ggg"] = 8921,

```



```

5470 ["nGg"] = {8921, 824},
5471 ["LessEqualGreater"] = 8922,
5472 ["leg"] = 8922,
5473 ["lesg"] = {8922, 65024},
5474 ["lesseqgtr"] = 8922,
5475 ["GreaterEqualLess"] = 8923,
5476 ["gel"] = 8923,
5477 ["gesl"] = {8923, 65024},
5478 ["gtreqless"] = 8923,
5479 ["cuepr"] = 8926,
5480 ["curlyeqprec"] = 8926,
5481 ["cuesc"] = 8927,
5482 ["curlyeqsucc"] = 8927,
5483 ["NotPrecedesSlantEqual"] = 8928,
5484 ["nprcue"] = 8928,
5485 ["NotSucceedsSlantEqual"] = 8929,
5486 ["nsccue"] = 8929,
5487 ["NotSquareSubsetEqual"] = 8930,
5488 ["nsqsube"] = 8930,
5489 ["NotSquareSupersetEqual"] = 8931,
5490 ["nsqsupe"] = 8931,
5491 ["lnsim"] = 8934,
5492 ["gnsim"] = 8935,
5493 ["precnsim"] = 8936,
5494 ["prnsim"] = 8936,
5495 ["scnsim"] = 8937,
5496 ["succnsim"] = 8937,
5497 ["NotLeftTriangle"] = 8938,
5498 ["nltri"] = 8938,
5499 ["ntriangleleft"] = 8938,
5500 ["NotRightTriangle"] = 8939,
5501 ["nrtri"] = 8939,
5502 ["ntriangleright"] = 8939,
5503 ["NotLeftTriangleEqual"] = 8940,
5504 ["nltrie"] = 8940,
5505 ["ntrianglelefteq"] = 8940,
5506 ["NotRightTriangleEqual"] = 8941,
5507 ["nrtrie"] = 8941,
5508 ["ntrianglerighteq"] = 8941,
5509 ["vellip"] = 8942,
5510 ["ctdot"] = 8943,
5511 ["utdot"] = 8944,
5512 ["dtdot"] = 8945,
5513 ["disin"] = 8946,
5514 ["isinsv"] = 8947,
5515 ["isins"] = 8948,
5516 ["isindot"] = 8949,

```

```

5517 ["notindot"] = {8949, 824},
5518 ["notinvc"] = 8950,
5519 ["notinvb"] = 8951,
5520 ["isinE"] = 8953,
5521 ["notinE"] = {8953, 824},
5522 ["nisd"] = 8954,
5523 ["xnis"] = 8955,
5524 ["nis"] = 8956,
5525 ["notnivc"] = 8957,
5526 ["notnivb"] = 8958,
5527 ["barwed"] = 8965,
5528 ["barwedge"] = 8965,
5529 ["Barwed"] = 8966,
5530 ["doublebarwedge"] = 8966,
5531 ["LeftCeiling"] = 8968,
5532 ["lceil"] = 8968,
5533 ["RightCeiling"] = 8969,
5534 ["rceil"] = 8969,
5535 ["LeftFloor"] = 8970,
5536 ["lfloor"] = 8970,
5537 ["RightFloor"] = 8971,
5538 ["rfloor"] = 8971,
5539 ["drcrop"] = 8972,
5540 ["dlcrop"] = 8973,
5541 ["urcrop"] = 8974,
5542 ["ulcrop"] = 8975,
5543 ["bnot"] = 8976,
5544 ["profline"] = 8978,
5545 ["profsurf"] = 8979,
5546 ["telrec"] = 8981,
5547 ["target"] = 8982,
5548 ["ulcorn"] = 8988,
5549 ["ulcorner"] = 8988,
5550 ["urcorn"] = 8989,
5551 ["urcorner"] = 8989,
5552 ["dlcorn"] = 8990,
5553 ["llcorner"] = 8990,
5554 ["drcorn"] = 8991,
5555 ["lrcorner"] = 8991,
5556 ["frown"] = 8994,
5557 ["sfrown"] = 8994,
5558 ["smile"] = 8995,
5559 ["ssmile"] = 8995,
5560 ["cylcty"] = 9005,
5561 ["profalar"] = 9006,
5562 ["topbot"] = 9014,
5563 ["ovbar"] = 9021,

```

```

5564 ["solbar"] = 9023,
5565 ["angzarr"] = 9084,
5566 ["lmoust"] = 9136,
5567 ["lmoustache"] = 9136,
5568 ["rmoust"] = 9137,
5569 ["rmoustache"] = 9137,
5570 ["OverBracket"] = 9140,
5571 ["tbrk"] = 9140,
5572 ["UnderBracket"] = 9141,
5573 ["bbrk"] = 9141,
5574 ["bbrktbrk"] = 9142,
5575 ["OverParenthesis"] = 9180,
5576 ["UnderParenthesis"] = 9181,
5577 ["OverBrace"] = 9182,
5578 ["UnderBrace"] = 9183,
5579 ["trpezium"] = 9186,
5580 ["elinters"] = 9191,
5581 ["blank"] = 9251,
5582 ["circledS"] = 9416,
5583 ["oS"] = 9416,
5584 ["HorizontalLine"] = 9472,
5585 ["boxh"] = 9472,
5586 ["boxv"] = 9474,
5587 ["boxdr"] = 9484,
5588 ["boxdl"] = 9488,
5589 ["boxur"] = 9492,
5590 ["boxul"] = 9496,
5591 ["boxvr"] = 9500,
5592 ["boxvl"] = 9508,
5593 ["boxhd"] = 9516,
5594 ["boxhu"] = 9524,
5595 ["boxvh"] = 9532,
5596 ["boxH"] = 9552,
5597 ["boxV"] = 9553,
5598 ["boxdR"] = 9554,
5599 ["boxDr"] = 9555,
5600 ["boxDR"] = 9556,
5601 ["boxdL"] = 9557,
5602 ["boxDL"] = 9558,
5603 ["boxDL"] = 9559,
5604 ["boxuR"] = 9560,
5605 ["boxUr"] = 9561,
5606 ["boxUR"] = 9562,
5607 ["boxuL"] = 9563,
5608 ["boxUL"] = 9564,
5609 ["boxUL"] = 9565,
5610 ["boxvR"] = 9566,

```

```

5611 ["boxVr"] = 9567,
5612 ["boxVR"] = 9568,
5613 ["boxvL"] = 9569,
5614 ["boxVl"] = 9570,
5615 ["boxVL"] = 9571,
5616 ["boxHd"] = 9572,
5617 ["boxhD"] = 9573,
5618 ["boxHD"] = 9574,
5619 ["boxHu"] = 9575,
5620 ["boxhU"] = 9576,
5621 ["boxHU"] = 9577,
5622 ["boxvH"] = 9578,
5623 ["boxVh"] = 9579,
5624 ["boxVH"] = 9580,
5625 ["uhblk"] = 9600,
5626 ["lhblk"] = 9604,
5627 ["block"] = 9608,
5628 ["blk14"] = 9617,
5629 ["blk12"] = 9618,
5630 ["blk34"] = 9619,
5631 ["Square"] = 9633,
5632 ["squ"] = 9633,
5633 ["square"] = 9633,
5634 ["FilledVerySmallSquare"] = 9642,
5635 ["blacksquare"] = 9642,
5636 ["squarf"] = 9642,
5637 ["squf"] = 9642,
5638 ["EmptyVerySmallSquare"] = 9643,
5639 ["rect"] = 9645,
5640 ["marker"] = 9646,
5641 ["fltns"] = 9649,
5642 ["bigtriangleup"] = 9651,
5643 ["xutri"] = 9651,
5644 ["blacktriangle"] = 9652,
5645 ["utrif"] = 9652,
5646 ["triangle"] = 9653,
5647 ["utri"] = 9653,
5648 ["blacktriangleright"] = 9656,
5649 ["rtrif"] = 9656,
5650 ["rtri"] = 9657,
5651 ["triangleright"] = 9657,
5652 ["bigtriangledown"] = 9661,
5653 ["xdtri"] = 9661,
5654 ["blacktriangledown"] = 9662,
5655 ["dtrif"] = 9662,
5656 ["dtri"] = 9663,
5657 ["triangledown"] = 9663,

```

```

5658 ["blacktriangleleft"] = 9666,
5659 ["ltrif"] = 9666,
5660 ["ltri"] = 9667,
5661 ["triangleleft"] = 9667,
5662 ["loz"] = 9674,
5663 ["lozenge"] = 9674,
5664 ["cir"] = 9675,
5665 ["tridot"] = 9708,
5666 ["bigcirc"] = 9711,
5667 ["xcirc"] = 9711,
5668 ["ultri"] = 9720,
5669 ["urtri"] = 9721,
5670 ["lltri"] = 9722,
5671 ["EmptySmallSquare"] = 9723,
5672 ["FilledSmallSquare"] = 9724,
5673 ["bigstar"] = 9733,
5674 ["starf"] = 9733,
5675 ["star"] = 9734,
5676 ["phone"] = 9742,
5677 ["female"] = 9792,
5678 ["male"] = 9794,
5679 ["spades"] = 9824,
5680 ["spadesuit"] = 9824,
5681 ["clubs"] = 9827,
5682 ["clubsuit"] = 9827,
5683 ["hearts"] = 9829,
5684 ["heartsuit"] = 9829,
5685 ["diamondsuit"] = 9830,
5686 ["diams"] = 9830,
5687 ["sung"] = 9834,
5688 ["flat"] = 9837,
5689 ["natur"] = 9838,
5690 ["natural"] = 9838,
5691 ["sharp"] = 9839,
5692 ["check"] = 10003,
5693 ["checkmark"] = 10003,
5694 ["cross"] = 10007,
5695 ["malt"] = 10016,
5696 ["maltese"] = 10016,
5697 ["sext"] = 10038,
5698 ["VerticalSeparator"] = 10072,
5699 ["lbbbrk"] = 10098,
5700 ["rbbbrk"] = 10099,
5701 ["bsolhsub"] = 10184,
5702 ["suphsol"] = 10185,
5703 ["LeftDoubleBracket"] = 10214,
5704 ["lobrk"] = 10214,

```

5705 ["RightDoubleBracket"] = 10215,  
 5706 ["robrk"] = 10215,  
 5707 ["LeftAngleBracket"] = 10216,  
 5708 ["lang"] = 10216,  
 5709 ["langle"] = 10216,  
 5710 ["RightAngleBracket"] = 10217,  
 5711 ["rang"] = 10217,  
 5712 ["rangle"] = 10217,  
 5713 ["Lang"] = 10218,  
 5714 ["Rang"] = 10219,  
 5715 ["loang"] = 10220,  
 5716 ["roang"] = 10221,  
 5717 ["LongLeftArrow"] = 10229,  
 5718 ["longleftarrow"] = 10229,  
 5719 ["xlarr"] = 10229,  
 5720 ["LongRightArrow"] = 10230,  
 5721 ["longrightarrow"] = 10230,  
 5722 ["xrarr"] = 10230,  
 5723 ["LongLeftRightArrow"] = 10231,  
 5724 ["longleftrightarrow"] = 10231,  
 5725 ["xharr"] = 10231,  
 5726 ["DoubleLongLeftArrow"] = 10232,  
 5727 ["Longleftarrow"] = 10232,  
 5728 ["xlArr"] = 10232,  
 5729 ["DoubleLongRightArrow"] = 10233,  
 5730 ["Longrightarrow"] = 10233,  
 5731 ["xrArr"] = 10233,  
 5732 ["DoubleLongLeftRightArrow"] = 10234,  
 5733 ["Longleftrightarrow"] = 10234,  
 5734 ["xhArr"] = 10234,  
 5735 ["longmapsto"] = 10236,  
 5736 ["xmap"] = 10236,  
 5737 ["dzigrarr"] = 10239,  
 5738 ["nvlArr"] = 10498,  
 5739 ["nvrArr"] = 10499,  
 5740 ["nvHarr"] = 10500,  
 5741 ["Map"] = 10501,  
 5742 ["lbarr"] = 10508,  
 5743 ["bkarow"] = 10509,  
 5744 ["rbarr"] = 10509,  
 5745 ["lBarr"] = 10510,  
 5746 ["dbkarow"] = 10511,  
 5747 ["rBarr"] = 10511,  
 5748 ["RBarr"] = 10512,  
 5749 ["drbkarow"] = 10512,  
 5750 ["DDottrahd"] = 10513,  
 5751 ["UpArrowBar"] = 10514,

```

5752 ["DownArrowBar"] = 10515,
5753 ["Rarrtl"] = 10518,
5754 ["latail"] = 10521,
5755 ["ratail"] = 10522,
5756 ["lAtail"] = 10523,
5757 ["rAtail"] = 10524,
5758 ["larrfs"] = 10525,
5759 ["rarrfs"] = 10526,
5760 ["larrbfs"] = 10527,
5761 ["rarrbfs"] = 10528,
5762 ["nwarhk"] = 10531,
5763 ["nearhk"] = 10532,
5764 ["hksearow"] = 10533,
5765 ["searhk"] = 10533,
5766 ["hkswarow"] = 10534,
5767 ["swarhk"] = 10534,
5768 ["nwnear"] = 10535,
5769 ["nesear"] = 10536,
5770 ["toea"] = 10536,
5771 ["seswar"] = 10537,
5772 ["tosa"] = 10537,
5773 ["swnwar"] = 10538,
5774 ["nrarrc"] = {10547, 824},
5775 ["rarrc"] = 10547,
5776 ["cudarrr"] = 10549,
5777 ["ldca"] = 10550,
5778 ["rdca"] = 10551,
5779 ["cudarrrl"] = 10552,
5780 ["larrpl"] = 10553,
5781 ["curarrm"] = 10556,
5782 ["cularrp"] = 10557,
5783 ["rarrpl"] = 10565,
5784 ["harrcir"] = 10568,
5785 ["Uarrocir"] = 10569,
5786 ["lurdshar"] = 10570,
5787 ["ldrushar"] = 10571,
5788 ["LeftRightVector"] = 10574,
5789 ["RightUpDownVector"] = 10575,
5790 ["DownLeftRightVector"] = 10576,
5791 ["LeftUpDownVector"] = 10577,
5792 ["LeftVectorBar"] = 10578,
5793 ["RightVectorBar"] = 10579,
5794 ["RightUpVectorBar"] = 10580,
5795 ["RightDownVectorBar"] = 10581,
5796 ["DownLeftVectorBar"] = 10582,
5797 ["DownRightVectorBar"] = 10583,
5798 ["LeftUpVectorBar"] = 10584,

```

```

5799 ["LeftDownVectorBar"] = 10585,
5800 ["LeftTeeVector"] = 10586,
5801 ["RightTeeVector"] = 10587,
5802 ["RightUpTeeVector"] = 10588,
5803 ["RightDownTeeVector"] = 10589,
5804 ["DownLeftTeeVector"] = 10590,
5805 ["DownRightTeeVector"] = 10591,
5806 ["LeftUpTeeVector"] = 10592,
5807 ["LeftDownTeeVector"] = 10593,
5808 ["lHar"] = 10594,
5809 ["uHar"] = 10595,
5810 ["rHar"] = 10596,
5811 ["dHar"] = 10597,
5812 ["luruhar"] = 10598,
5813 ["ldrdhar"] = 10599,
5814 ["ruluhar"] = 10600,
5815 ["rdldhar"] = 10601,
5816 ["lharul"] = 10602,
5817 ["llhard"] = 10603,
5818 ["rharul"] = 10604,
5819 ["lrhard"] = 10605,
5820 ["UpEquilibrium"] = 10606,
5821 ["udhar"] = 10606,
5822 ["ReverseUpEquilibrium"] = 10607,
5823 ["duhar"] = 10607,
5824 ["RoundImplies"] = 10608,
5825 ["erarr"] = 10609,
5826 ["simrarr"] = 10610,
5827 ["larrsim"] = 10611,
5828 ["rarrsim"] = 10612,
5829 ["rarrap"] = 10613,
5830 ["ltlarr"] = 10614,
5831 ["gtrarr"] = 10616,
5832 ["subrarr"] = 10617,
5833 ["suplarr"] = 10619,
5834 ["lfisht"] = 10620,
5835 ["rfisht"] = 10621,
5836 ["ufisht"] = 10622,
5837 ["dfisht"] = 10623,
5838 ["lopar"] = 10629,
5839 ["ropar"] = 10630,
5840 ["lbrke"] = 10635,
5841 ["rbrke"] = 10636,
5842 ["lbrkslu"] = 10637,
5843 ["rbrksld"] = 10638,
5844 ["lbrksld"] = 10639,
5845 ["rbrkslu"] = 10640,

```



```

5846 ["langd"] = 10641,
5847 ["rangd"] = 10642,
5848 ["lparlt"] = 10643,
5849 ["rpargt"] = 10644,
5850 ["gtlPar"] = 10645,
5851 ["ltrPar"] = 10646,
5852 ["vzigzag"] = 10650,
5853 ["vangrt"] = 10652,
5854 ["angrtvbd"] = 10653,
5855 ["ange"] = 10660,
5856 ["range"] = 10661,
5857 ["dwangle"] = 10662,
5858 ["uwangle"] = 10663,
5859 ["angmsdaa"] = 10664,
5860 ["angmsdab"] = 10665,
5861 ["angmsdac"] = 10666,
5862 ["angmsdad"] = 10667,
5863 ["angmsdae"] = 10668,
5864 ["angmsdaf"] = 10669,
5865 ["angmsdag"] = 10670,
5866 ["angmsdah"] = 10671,
5867 ["bemptyv"] = 10672,
5868 ["demptyv"] = 10673,
5869 ["cemptyv"] = 10674,
5870 ["raemptyv"] = 10675,
5871 ["laemptyv"] = 10676,
5872 ["ohbar"] = 10677,
5873 ["omid"] = 10678,
5874 ["opar"] = 10679,
5875 ["operp"] = 10681,
5876 ["olcross"] = 10683,
5877 ["odsold"] = 10684,
5878 ["olcir"] = 10686,
5879 ["ofcir"] = 10687,
5880 ["olt"] = 10688,
5881 ["ogt"] = 10689,
5882 ["cirscir"] = 10690,
5883 ["cirE"] = 10691,
5884 ["solb"] = 10692,
5885 ["bsolb"] = 10693,
5886 ["boxbox"] = 10697,
5887 ["trish"] = 10701,
5888 ["rtriltri"] = 10702,
5889 ["LeftTriangleBar"] = 10703,
5890 ["NotLeftTriangleBar"] = {10703, 824},
5891 ["NotRightTriangleBar"] = {10704, 824},
5892 ["RightTriangleBar"] = 10704,

```

```

5893 ["i infin"] = 10716,
5894 ["infintie"] = 10717,
5895 ["nvinfin"] = 10718,
5896 ["eparsl"] = 10723,
5897 ["smeparsl"] = 10724,
5898 ["eqvparsl"] = 10725,
5899 ["blacklozenge"] = 10731,
5900 ["lozf"] = 10731,
5901 ["RuleDelayed"] = 10740,
5902 ["dsol"] = 10742,
5903 ["bigodot"] = 10752,
5904 ["xodot"] = 10752,
5905 ["bigoplus"] = 10753,
5906 ["xoplus"] = 10753,
5907 ["bigotimes"] = 10754,
5908 ["xotime"] = 10754,
5909 ["biguplus"] = 10756,
5910 ["xuplus"] = 10756,
5911 ["bigsqcup"] = 10758,
5912 ["xsqcup"] = 10758,
5913 ["iiiint"] = 10764,
5914 ["qint"] = 10764,
5915 ["fpartint"] = 10765,
5916 ["cirfnint"] = 10768,
5917 ["awint"] = 10769,
5918 ["rppolint"] = 10770,
5919 ["scpolint"] = 10771,
5920 ["npolint"] = 10772,
5921 ["pointint"] = 10773,
5922 ["quatint"] = 10774,
5923 ["intlarhk"] = 10775,
5924 ["pluscir"] = 10786,
5925 ["plusacir"] = 10787,
5926 ["simplus"] = 10788,
5927 ["plusdu"] = 10789,
5928 ["plussim"] = 10790,
5929 ["plustwo"] = 10791,
5930 ["mcomma"] = 10793,
5931 ["minusdu"] = 10794,
5932 ["loplus"] = 10797,
5933 ["roplus"] = 10798,
5934 ["Cross"] = 10799,
5935 ["timesd"] = 10800,
5936 ["timesbar"] = 10801,
5937 ["smashp"] = 10803,
5938 ["lotimes"] = 10804,
5939 ["rotimes"] = 10805,

```

```

5940 ["otimesas"] = 10806,
5941 ["Otimes"] = 10807,
5942 ["odiv"] = 10808,
5943 ["triplus"] = 10809,
5944 ["triminus"] = 10810,
5945 ["tritime"] = 10811,
5946 ["intprod"] = 10812,
5947 ["iprod"] = 10812,
5948 ["amalg"] = 10815,
5949 ["capdot"] = 10816,
5950 ["ncup"] = 10818,
5951 ["ncap"] = 10819,
5952 ["capand"] = 10820,
5953 ["cupor"] = 10821,
5954 ["cupcap"] = 10822,
5955 ["capcup"] = 10823,
5956 ["cupbrcap"] = 10824,
5957 ["capbrcup"] = 10825,
5958 ["cupcup"] = 10826,
5959 ["capcap"] = 10827,
5960 ["ccups"] = 10828,
5961 ["ccaps"] = 10829,
5962 ["ccupssm"] = 10832,
5963 ["And"] = 10835,
5964 ["Or"] = 10836,
5965 ["andand"] = 10837,
5966 ["oror"] = 10838,
5967 ["orslope"] = 10839,
5968 ["andslope"] = 10840,
5969 ["andv"] = 10842,
5970 ["orv"] = 10843,
5971 ["andd"] = 10844,
5972 ["ord"] = 10845,
5973 ["wedbar"] = 10847,
5974 ["sdote"] = 10854,
5975 ["simdot"] = 10858,
5976 ["congdots"] = 10861,
5977 ["ncongdots"] = {10861, 824},
5978 ["easter"] = 10862,
5979 ["apacir"] = 10863,
5980 ["apE"] = 10864,
5981 ["napE"] = {10864, 824},
5982 ["eplus"] = 10865,
5983 ["pluse"] = 10866,
5984 ["Esim"] = 10867,
5985 ["Colone"] = 10868,
5986 ["Equal"] = 10869,

```

```

5987 ["ddotseq"] = 10871,
5988 ["eDDot"] = 10871,
5989 ["equivDD"] = 10872,
5990 ["ltcir"] = 10873,
5991 ["gtcir"] = 10874,
5992 ["ltquest"] = 10875,
5993 ["gtquest"] = 10876,
5994 ["LessSlantEqual"] = 10877,
5995 ["NotLessSlantEqual"] = {10877, 824},
5996 ["leqslant"] = 10877,
5997 ["les"] = 10877,
5998 ["nleqslant"] = {10877, 824},
5999 ["nles"] = {10877, 824},
6000 ["GreaterSlantEqual"] = 10878,
6001 ["NotGreaterSlantEqual"] = {10878, 824},
6002 ["geqslant"] = 10878,
6003 ["ges"] = 10878,
6004 ["ngeqslant"] = {10878, 824},
6005 ["nges"] = {10878, 824},
6006 ["lesdot"] = 10879,
6007 ["gesdot"] = 10880,
6008 ["lesdoto"] = 10881,
6009 ["gesdoto"] = 10882,
6010 ["lesdotor"] = 10883,
6011 ["gesdoto1"] = 10884,
6012 ["lap"] = 10885,
6013 ["lessapprox"] = 10885,
6014 ["gap"] = 10886,
6015 ["gtrapprox"] = 10886,
6016 ["lne"] = 10887,
6017 ["lneq"] = 10887,
6018 ["gne"] = 10888,
6019 ["gneq"] = 10888,
6020 ["lnap"] = 10889,
6021 ["lnapprox"] = 10889,
6022 ["gnap"] = 10890,
6023 ["gnapprox"] = 10890,
6024 ["lEg"] = 10891,
6025 ["lesseqqgtr"] = 10891,
6026 ["gEl"] = 10892,
6027 ["gtreqqless"] = 10892,
6028 ["lsime"] = 10893,
6029 ["gsime"] = 10894,
6030 ["lsimg"] = 10895,
6031 ["gsiml"] = 10896,
6032 ["lgE"] = 10897,
6033 ["glE"] = 10898,

```

```

6034 ["lesges"] = 10899,
6035 ["gesles"] = 10900,
6036 ["els"] = 10901,
6037 ["eqslantless"] = 10901,
6038 ["egs"] = 10902,
6039 ["eqslantgtr"] = 10902,
6040 ["elsdot"] = 10903,
6041 ["egsdot"] = 10904,
6042 ["el"] = 10905,
6043 ["eg"] = 10906,
6044 ["siml"] = 10909,
6045 ["simg"] = 10910,
6046 ["simlE"] = 10911,
6047 ["simgE"] = 10912,
6048 ["LessLess"] = 10913,
6049 ["NotNestedLessLess"] = {10913, 824},
6050 ["GreaterGreater"] = 10914,
6051 ["NotNestedGreaterGreater"] = {10914, 824},
6052 ["glj"] = 10916,
6053 ["gla"] = 10917,
6054 ["ltcc"] = 10918,
6055 ["gtcc"] = 10919,
6056 ["lescc"] = 10920,
6057 ["gescc"] = 10921,
6058 ["smt"] = 10922,
6059 ["lat"] = 10923,
6060 ["smte"] = 10924,
6061 ["smtes"] = {10924, 65024},
6062 ["late"] = 10925,
6063 ["lates"] = {10925, 65024},
6064 ["bumpE"] = 10926,
6065 ["NotPrecedesEqual"] = {10927, 824},
6066 ["PrecedesEqual"] = 10927,
6067 ["npre"] = {10927, 824},
6068 ["npreceq"] = {10927, 824},
6069 ["pre"] = 10927,
6070 ["preceq"] = 10927,
6071 ["NotSucceedsEqual"] = {10928, 824},
6072 ["SucceedsEqual"] = 10928,
6073 ["nsce"] = {10928, 824},
6074 ["nsucceq"] = {10928, 824},
6075 ["sce"] = 10928,
6076 ["succeq"] = 10928,
6077 ["prE"] = 10931,
6078 ["scE"] = 10932,
6079 ["precneqq"] = 10933,
6080 ["prnE"] = 10933,

```

```

6081 ["scnE"] = 10934,
6082 ["succneqq"] = 10934,
6083 ["prap"] = 10935,
6084 ["precapprox"] = 10935,
6085 ["scap"] = 10936,
6086 ["succapprox"] = 10936,
6087 ["precnapprox"] = 10937,
6088 ["prnap"] = 10937,
6089 ["scnap"] = 10938,
6090 ["succnapprox"] = 10938,
6091 ["Pr"] = 10939,
6092 ["Sc"] = 10940,
6093 ["subdot"] = 10941,
6094 ["supdot"] = 10942,
6095 ["subplus"] = 10943,
6096 ["supplus"] = 10944,
6097 ["submult"] = 10945,
6098 ["supmult"] = 10946,
6099 ["subedot"] = 10947,
6100 ["supedot"] = 10948,
6101 ["nsubE"] = {10949, 824},
6102 ["nsubseteqq"] = {10949, 824},
6103 ["subE"] = 10949,
6104 ["subseteqq"] = 10949,
6105 ["nsupE"] = {10950, 824},
6106 ["nsupseteqq"] = {10950, 824},
6107 ["supE"] = 10950,
6108 ["supseteqq"] = 10950,
6109 ["subsim"] = 10951,
6110 ["supsim"] = 10952,
6111 ["subnE"] = 10955,
6112 ["subsetneqq"] = 10955,
6113 ["varsubsetneqq"] = {10955, 65024},
6114 ["vsubnE"] = {10955, 65024},
6115 ["supnE"] = 10956,
6116 ["supsetneqq"] = 10956,
6117 ["varsupsetneqq"] = {10956, 65024},
6118 ["vsupnE"] = {10956, 65024},
6119 ["csub"] = 10959,
6120 ["csup"] = 10960,
6121 ["csube"] = 10961,
6122 ["csupe"] = 10962,
6123 ["subsup"] = 10963,
6124 ["supsub"] = 10964,
6125 ["subsub"] = 10965,
6126 ["supsup"] = 10966,
6127 ["suphsub"] = 10967,

```

```

6128 ["supdsub"] = 10968,
6129 ["forkv"] = 10969,
6130 ["topfork"] = 10970,
6131 ["mlcp"] = 10971,
6132 ["Dashv"] = 10980,
6133 ["DoubleLeftTee"] = 10980,
6134 ["Vdashl"] = 10982,
6135 ["Barv"] = 10983,
6136 ["vBar"] = 10984,
6137 ["vBarv"] = 10985,
6138 ["Vbar"] = 10987,
6139 ["Not"] = 10988,
6140 ["bNot"] = 10989,
6141 ["rnmid"] = 10990,
6142 ["cirmid"] = 10991,
6143 ["midcir"] = 10992,
6144 ["topcir"] = 10993,
6145 ["nhpar"] = 10994,
6146 ["parsim"] = 10995,
6147 ["nparsl"] = {11005, 8421},
6148 ["parsl"] = 11005,
6149 ["fflig"] = 64256,
6150 ["filig"] = 64257,
6151 ["fllig"] = 64258,
6152 ["ffilig"] = 64259,
6153 ["ffllig"] = 64260,
6154 ["Ascr"] = 119964,
6155 ["Cscr"] = 119966,
6156 ["Dscr"] = 119967,
6157 ["Gscr"] = 119970,
6158 ["Jscr"] = 119973,
6159 ["Kscr"] = 119974,
6160 ["Nscr"] = 119977,
6161 ["Oscr"] = 119978,
6162 ["Pscr"] = 119979,
6163 ["Qscr"] = 119980,
6164 ["Sscr"] = 119982,
6165 ["Tscr"] = 119983,
6166 ["Uscr"] = 119984,
6167 ["Vscr"] = 119985,
6168 ["Wscr"] = 119986,
6169 ["Xscr"] = 119987,
6170 ["Yscr"] = 119988,
6171 ["Zscr"] = 119989,
6172 ["ascr"] = 119990,
6173 ["bscr"] = 119991,
6174 ["cscr"] = 119992,

```

```

6175 ["dscr"] = 119993,
6176 ["fscr"] = 119995,
6177 ["hscr"] = 119997,
6178 ["iscr"] = 119998,
6179 ["jscr"] = 119999,
6180 ["kscr"] = 120000,
6181 ["lscr"] = 120001,
6182 ["mscr"] = 120002,
6183 ["nscr"] = 120003,
6184 ["pscr"] = 120005,
6185 ["qscr"] = 120006,
6186 ["rscr"] = 120007,
6187 ["sscr"] = 120008,
6188 ["tscr"] = 120009,
6189 ["uscr"] = 120010,
6190 ["vscr"] = 120011,
6191 ["wscr"] = 120012,
6192 ["xscr"] = 120013,
6193 ["yscr"] = 120014,
6194 ["zscr"] = 120015,
6195 ["Afr"] = 120068,
6196 ["Bfr"] = 120069,
6197 ["Dfr"] = 120071,
6198 ["Efr"] = 120072,
6199 ["Ffr"] = 120073,
6200 ["Gfr"] = 120074,
6201 ["Jfr"] = 120077,
6202 ["Kfr"] = 120078,
6203 ["Lfr"] = 120079,
6204 ["Mfr"] = 120080,
6205 ["Nfr"] = 120081,
6206 ["Ofr"] = 120082,
6207 ["Pfr"] = 120083,
6208 ["Qfr"] = 120084,
6209 ["Sfr"] = 120086,
6210 ["Tfr"] = 120087,
6211 ["Ufr"] = 120088,
6212 ["Vfr"] = 120089,
6213 ["Wfr"] = 120090,
6214 ["Xfr"] = 120091,
6215 ["Yfr"] = 120092,
6216 ["afr"] = 120094,
6217 ["bfr"] = 120095,
6218 ["cfr"] = 120096,
6219 ["dfr"] = 120097,
6220 ["efr"] = 120098,
6221 ["ffr"] = 120099,

```



```

6222 ["gfr"] = 120100,
6223 ["hfr"] = 120101,
6224 ["ifr"] = 120102,
6225 ["jfr"] = 120103,
6226 ["kfr"] = 120104,
6227 ["lfr"] = 120105,
6228 ["mfr"] = 120106,
6229 ["nfr"] = 120107,
6230 ["ofr"] = 120108,
6231 ["pfr"] = 120109,
6232 ["qfr"] = 120110,
6233 ["rfr"] = 120111,
6234 ["sfr"] = 120112,
6235 ["tfr"] = 120113,
6236 ["ufr"] = 120114,
6237 ["vfr"] = 120115,
6238 ["wfr"] = 120116,
6239 ["xfr"] = 120117,
6240 ["yfr"] = 120118,
6241 ["zfr"] = 120119,
6242 ["Aopf"] = 120120,
6243 ["Bopf"] = 120121,
6244 ["Dopf"] = 120123,
6245 ["Eopf"] = 120124,
6246 ["Fopf"] = 120125,
6247 ["Gopf"] = 120126,
6248 ["Iopf"] = 120128,
6249 ["Jopf"] = 120129,
6250 ["Kopf"] = 120130,
6251 ["Lopf"] = 120131,
6252 ["Mopf"] = 120132,
6253 ["Oopf"] = 120134,
6254 ["Sopf"] = 120138,
6255 ["Topf"] = 120139,
6256 ["Uopf"] = 120140,
6257 ["Vopf"] = 120141,
6258 ["Wopf"] = 120142,
6259 ["Xopf"] = 120143,
6260 ["Yopf"] = 120144,
6261 ["aopf"] = 120146,
6262 ["bopf"] = 120147,
6263 ["copf"] = 120148,
6264 ["dopf"] = 120149,
6265 ["eopf"] = 120150,
6266 ["fopf"] = 120151,
6267 ["gopf"] = 120152,
6268 ["hopf"] = 120153,

```

```

6269 ["iopf"] = 120154,
6270 ["jopf"] = 120155,
6271 ["kopf"] = 120156,
6272 ["lopf"] = 120157,
6273 ["mopf"] = 120158,
6274 ["nopf"] = 120159,
6275 ["oopf"] = 120160,
6276 ["popf"] = 120161,
6277 ["qopf"] = 120162,
6278 ["ropf"] = 120163,
6279 ["sopf"] = 120164,
6280 ["topf"] = 120165,
6281 ["uopf"] = 120166,
6282 ["vopf"] = 120167,
6283 ["wopf"] = 120168,
6284 ["xopf"] = 120169,
6285 ["yopf"] = 120170,
6286 ["zopf"] = 120171,
6287 }

```

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

```

6288 function entities.dec_entity(s)
6289   local n = tonumber(s)
6290   if n == nil then
6291     return "&#" .. s .. ";" -- fallback for unknown entities
6292   end
6293   return utf8.char(n)
6294 end

```

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

```

6295 function entities.hex_entity(s)
6296   local n = tonumber("0x"..s)
6297   if n == nil then
6298     return "&#x" .. s .. ";" -- fallback for unknown entities
6299   end
6300   return utf8.char(n)
6301 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.

```

6302 function entities.hex_entity_with_x_char(x, s)
6303   local n = tonumber("0x"..s)
6304   if n == nil then
6305     return "&#" .. x .. s .. ";" -- fallback for unknown entities
6306   end

```

```

6307     return utf8.char(n)
6308 end

```

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

```

6309 function entities.char_entity(s)
6310     local code_points = character_entities[s]
6311     if code_points == nil then
6312         return "&" .. s .. ";"
6313     end
6314     if type(code_points) ~= 'table' then
6315         code_points = {code_points}
6316     end
6317     local char_table = {}
6318     for _, code_point in ipairs(code_points) do
6319         table.insert(char_table, utf8.char(code_point))
6320     end
6321     return table.concat(char_table)
6322 end

```

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

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

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

```

6323 M.writer = {}

```

The `writer.new` method creates and returns a new T<sub>E</sub>X writer object associated with the Lua interface options (see Section 2.1.3) `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.

```

6324 local parsers
6325 function M.writer.new(options)
6326     local self = {}

```

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

```

6327     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.

```
6328 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.

```
6329 local slice_specifiers = {}
6330 for specifier in options.slice:gmatch("[^%s]+") do
6331     table.insert(slice_specifiers, specifier)
6332 end
6333
6334 if #slice_specifiers == 2 then
6335     self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
6336     local slice_begin_type = self.slice_begin:sub(1, 1)
6337     if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
6338         self.slice_begin = "^" .. self.slice_begin
6339     end
6340     local slice_end_type = self.slice_end:sub(1, 1)
6341     if slice_end_type ~= "^" and slice_end_type ~= "$" then
6342         self.slice_end = "$" .. self.slice_end
6343     end
6344 elseif #slice_specifiers == 1 then
6345     self.slice_begin = "^" .. slice_specifiers[1]
6346     self.slice_end = "$" .. slice_specifiers[1]
6347 end
6348
6349 self.slice_begin_type = self.slice_begin:sub(1, 1)
6350 self.slice_begin_identifier = self.slice_begin:sub(2) or ""
6351 self.slice_end_type = self.slice_end:sub(1, 1)
6352 self.slice_end_identifier = self.slice_end:sub(2) or ""
6353
6354 if self.slice_begin == "^" and self.slice_end ~= "^" then
6355     self.is_writing = true
6356 else
6357     self.is_writing = false
6358 end
```

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

```
6359 self.space = " "
```

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

```
6360 self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
6361 function self.plain(s)
6362     return s
```

```
6363     end
```

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

```
6364     function self.paragraph(s)
6365         if not self.is_writing then return "" end
6366         return s
6367     end
```

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

```
6368     self.interblocksep_text = "\\markdownRendererInterblockSeparator\\n{"
6369     function self.interblocksep()
6370         if not self.is_writing then return "" end
6371         return self.interblocksep_text
6372     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.

```
6373     self.paragraphsep_text = "\\markdownRendererParagraphSeparator\\n{"
6374     function self.paragraphsep()
6375         if not self.is_writing then return "" end
6376         return self.paragraphsep_text
6377     end
```

Define `writer->undosep` as a function that will remove the output produced by an immediately preceding block element / paragraph separator.

```
6378     self.undosep_text = "\\markdownRendererUndoSeparator\\n{"
6379     function self.undosep()
6380         if not self.is_writing then return "" end
6381         return self.undosep_text
6382     end
```

Define `writer->soft_line_break` as the output format of a soft line break.

```
6383     self.soft_line_break = function()
6384         if self.flatten_inlines then return "\\n" end
6385         return "\\markdownRendererSoftLineBreak\\n{"
6386     end
```

Define `writer->hard_line_break` as the output format of a hard line break.

```
6387     self.hard_line_break = function()
6388         if self.flatten_inlines then return "\\n" end
6389         return "\\markdownRendererHardLineBreak\\n{"
6390     end
```

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

```
6391     self.ellipsis = "\\markdownRendererEllipsis{"
```

Define `writer->thematic_break` as the output format of a thematic break.

```
6392 function self.thematic_break()
6393     if not self.is_writing then return "" end
6394     return "\\markdownRendererThematicBreak{}"
6395 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.

```
6396 self.escaped_uri_chars = {
6397     ["{"] = "\\markdownRendererLeftBrace{}",
6398     ["}"] = "\\markdownRendererRightBrace{}",
6399     ["\\"] = "\\markdownRendererBackslash{}",
6400     ["\r"] = " ",
6401     ["\n"] = " ",
6402 }
6403 self.escaped_minimal_strings = {
6404     ["^"] = "\\markdownRendererCircumflex",
6405     .. "\\markdownRendererCircumflex ",
6406     ["☑"] = "\\markdownRendererTickedBox{}",
6407     ["☐"] = "\\markdownRendererHalfTickedBox{}",
6408     ["□"] = "\\markdownRendererUntickedBox{}",
6409     [entities.hex_entity('FFFD')]
6410     = "\\markdownRendererReplacementCharacter{}",
6411 }
```

Define table `writer->escaped_strings` containing the mapping from character strings that need to be escaped in typeset content.

```
6412 self.escaped_strings = util.table_copy(self.escaped_minimal_strings)
6413 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.

```
6414 self.escaped_chars = {
6415     ["{"] = "\\markdownRendererLeftBrace{}",
6416     ["}"] = "\\markdownRendererRightBrace{}",
6417     ["%"] = "\\markdownRendererPercentSign{}",
6418     ["\\"] = "\\markdownRendererBackslash{}",
6419     ["#"] = "\\markdownRendererHash{}",
6420     ["$"] = "\\markdownRendererDollarSign{}",
6421     ["&"] = "\\markdownRendererAmpersand{}",
6422     ["_"] = "\\markdownRendererUnderscore{}",
6423     ["^"] = "\\markdownRendererCircumflex{}",
6424     ["~"] = "\\markdownRendererTilde{}",
6425     ["|"] = "\\markdownRendererPipe{}",
6426     [entities.hex_entity('0000')]

```

```

6427         = "\\markdownRendererReplacementCharacter{}",
6428     }

```

Use the `writer->escaped_chars`, `writer->escaped_uri_chars`, and `writer->escaped_minimal_strings` tables to create the `escape_typographic_text`, `escape_programmatic_text`, and `escape_minimal` local escaper functions.

```

6429     local function create_escaper(char_escapes, string_escapes)
6430         local escape = util.escaper(char_escapes, string_escapes)
6431         return function(s)
6432             if self.flatten_inlines then return s end
6433             return escape(s)
6434         end
6435     end
6436     local escape_typographic_text = create_escaper(
6437         self.escaped_chars, self.escaped_strings)
6438     local escape_programmatic_text = create_escaper(
6439         self.escaped_uri_chars, self.escaped_minimal_strings)
6440     local escape_minimal = create_escaper(
6441         {}, 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.

```

6442     self.escape = escape_typographic_text
6443     self.math = escape_minimal
6444     if options.hybrid then
6445         self.identifier = escape_minimal
6446         self.string = escape_minimal
6447         self.uri = escape_minimal
6448         self.infostring = escape_minimal
6449     else
6450         self.identifier = escape_programmatic_text
6451         self.string = escape_typographic_text
6452         self.uri = escape_programmatic_text
6453         self.infostring = escape_programmatic_text
6454     end

```

Define `writer->warning` as a function that will transform an input warning `t` with optional more warning text `m` to the output format.

```

6455 function self.warning(t, m)
6456     return {"\\markdownRendererWarning{" , self.escape(t), "}{" ,
6457             escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
6458             escape_minimal(m or ""), "}"}
6459 end

```

Define `writer->error` as a function that will transform an input error text `t` with optional more error text `m` to the output format.

```

6460 function self.error(t, m)
6461     return {"\\markdownRendererError{" , self.escape(t), "}{" ,
6462             escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
6463             escape_minimal(m or ""), "}"}
6464 end

```

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

```

6465 function self.code(s, attributes)
6466     if self.flatten_inlines then return s end
6467     local buf = {}
6468     if attributes ~= nil then
6469         table.insert(buf,
6470             "\\markdownRendererCodeSpanAttributeContextBegin\n")
6471         table.insert(buf, self.attributes(attributes))
6472     end
6473     table.insert(buf,
6474         {"\\markdownRendererCodeSpan{" , self.escape(s), "}"}
6475     if attributes ~= nil then
6476         table.insert(buf,
6477             "\\markdownRendererCodeSpanAttributeContextEnd{")
6478     end
6479     return buf
6480 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.

```

6481 function self.link(lab, src, tit, attributes)
6482     if self.flatten_inlines then return lab end
6483     local buf = {}
6484     if attributes ~= nil then
6485         table.insert(buf,
6486             "\\markdownRendererLinkAttributeContextBegin\n")
6487         table.insert(buf, self.attributes(attributes))
6488     end
6489     table.insert(buf, {"\\markdownRendererLink{" , lab, "}" ,
6490         {" , self.escape(src), "}" ,
6491         {" , self.uri(src), "}" ,
6492         {" , self.string(tit or ""), "}"})

```



```

6493     if attributes ~= nil then
6494         table.insert(buf,
6495             "\\markdownRendererLinkAttributeContextEnd{ }")
6496     end
6497     return buf
6498 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.

```

6499 function self.image(lab, src, tit, attributes)
6500     if self.flatten_inlines then return lab end
6501     local buf = {}
6502     if attributes ~= nil then
6503         table.insert(buf,
6504             "\\markdownRendererImageAttributeContextBegin\n")
6505         table.insert(buf, self.attributes(attributes))
6506     end
6507     table.insert(buf, {"\\markdownRendererImage{",lab,"",
6508         "{",self.string(src),"}",
6509         "{",self.uri(src),"}",
6510         "{",self.string(tit or ""),"}"}))
6511     if attributes ~= nil then
6512         table.insert(buf,
6513             "\\markdownRendererImageAttributeContextEnd{ }")
6514     end
6515     return buf
6516 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.

```

6517 function self.bulletlist(items,tight)
6518     if not self.is_writing then return "" end
6519     local buffer = {}
6520     for _,item in ipairs(items) do
6521         if item ~= "" then
6522             buffer[#buffer + 1] = self.bulletitem(item)
6523         end
6524     end
6525     local contents = util.intersperse(buffer,"\n")
6526     if tight and options.tightLists then
6527         return {"\\markdownRendererUlBeginTight\n",contents,
6528             "\n\\markdownRendererUlEndTight "}
6529     else
6530         return {"\\markdownRendererUlBegin\n",contents,
6531             "\n\\markdownRendererUlEnd "}

```

```

6532     end
6533 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.

```

6534 function self.bulletitem(s)
6535     return {"\\markdownRenderUListItem ",s,
6536           "\\markdownRenderUListItemEnd "}
6537 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.

```

6538 function self.orderedlist(items,tight,startnum)
6539     if not self.is_writing then return "" end
6540     local buffer = {}
6541     local num = startnum
6542     for _,item in ipairs(items) do
6543         if item ~= "" then
6544             buffer[#buffer + 1] = self.ordereditem(item,num)
6545         end
6546         if num ~= nil and item ~= "" then
6547             num = num + 1
6548         end
6549     end
6550     local contents = util.intersperse(buffer,"\n")
6551     if tight and options.tightLists then
6552         return {"\\markdownRenderO1BeginTight\n",contents,
6553               "\n\\markdownRenderO1EndTight "}
6554     else
6555         return {"\\markdownRenderO1Begin\n",contents,
6556               "\n\\markdownRenderO1End "}
6557     end
6558 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.

```

6559 function self.ordereditem(s,num)
6560     if num ~= nil then
6561         return {"\\markdownRenderO1ItemWithNumber{",num,"}",s,
6562               "\\markdownRenderO1ItemEnd "}
6563     else
6564         return {"\\markdownRenderO1Item ",s,
6565               "\\markdownRenderO1ItemEnd "}
6566     end
6567 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.

```
6568 function self.inline_html_comment(contents)
6569     if self.flatten_inlines then return contents end
6570     return {"\\markdownRendererInlineHtmlComment{",contents,""}
6571 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.

```
6572 function self.inline_html_tag(contents)
6573     if self.flatten_inlines then return contents end
6574     return {"\\markdownRendererInlineHtmlTag{",
6575             self.string(contents),"}"}
6576 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.

```
6577 function self.block_html_element(s)
6578     if not self.is_writing then return "" end
6579     local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
6580     return {"\\markdownRendererInputBlockHtmlElement{",name,""}
6581 end
```

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

```
6582 function self.emphasis(s)
6583     if self.flatten_inlines then return s end
6584     return {"\\markdownRendererEmphasis{",s,""}
6585 end
```

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

```
6586 function self.checkbox(f)
6587     if f == 1.0 then
6588         return "☑ "
6589     elseif f == 0.0 then
6590         return "☐ "
6591     else
6592         return "◻ "
6593     end
6594 end
```

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

```
6595 function self.strong(s)
```

```

6596     if self.flatten_inlines then return s end
6597     return {"\\markdownRendererStrongEmphasis{" ,s,""}
6598 end

```

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

```

6599 function self.blockquote(s)
6600     if not self.is_writing then return "" end
6601     return {"\\markdownRendererBlockQuoteBegin\n",s,
6602           "\\markdownRendererBlockQuoteEnd "}
6603 end

```

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

```

6604 function self.verbatim(s)
6605     if not self.is_writing then return "" end
6606     s = s:gsub("\n$", "")
6607     local name = util.cache_verbatim(options.cacheDir, s)
6608     return {"\\markdownRendererInputVerbatim{" ,name,""}
6609 end

```

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

```

6610 function self.document(d)
6611     local buf = {"\\markdownRendererDocumentBegin\n"}
6612
6613     -- warn against the `hybrid` option
6614     if options.hybrid then
6615         local text = "The `hybrid` option has been soft-deprecated."
6616         local more = "Consider using one of the following better options "
6617             .. "for mixing TeX and markdown: `contentBlocks`, "
6618             .. "`rawAttribute`, `texComments`, `texMathDollars`, "
6619             .. "`texMathSingleBackslash`, and "
6620             .. "`texMathDoubleBackslash`. "
6621             .. "For more information, see the user manual at "
6622             .. "<https://witiko.github.io/markdown/>."
6623         table.insert(buf, self.warning(text, more))
6624     end
6625
6626     -- insert the text of the document
6627     table.insert(buf, d)
6628
6629     -- pop all attributes
6630     table.insert(buf, self.pop_attributes())
6631
6632     table.insert(buf, "\\markdownRendererDocumentEnd")
6633
6634     return buf

```

```
6635 end
```

Define `writer->attributes` as a function that will transform input attributes `attrs` to the output format.

```
6636 local seen_identifiers = {}
6637 local key_value_regex = "([^\s= ]+)%s*=%s*(.*)"
6638 local function normalize_attributes(attributes, auto_identifiers)
6639     -- normalize attributes
6640     local normalized_attributes = {}
6641     local has_explicit_identifiers = false
6642     local key, value
6643     for _, attribute in ipairs(attributes or {}) do
6644         if attribute:sub(1, 1) == "#" then
6645             table.insert(normalized_attributes, attribute)
6646             has_explicit_identifiers = true
6647             seen_identifiers[attribute:sub(2)] = true
6648         elseif attribute:sub(1, 1) == "." then
6649             table.insert(normalized_attributes, attribute)
6650         else
6651             key, value = attribute:match(key_value_regex)
6652             if key:lower() == "id" then
6653                 table.insert(normalized_attributes, "#" .. value)
6654             elseif key:lower() == "class" then
6655                 local classes = {}
6656                 for class in value:gmatch("%S+") do
6657                     table.insert(classes, class)
6658                 end
6659                 table.sort(classes)
6660                 for _, class in ipairs(classes) do
6661                     table.insert(normalized_attributes, "." .. class)
6662                 end
6663             else
6664                 table.insert(normalized_attributes, attribute)
6665             end
6666         end
6667     end
6668
6669     -- if no explicit identifiers exist, add auto identifiers
6670     if not has_explicit_identifiers and auto_identifiers ~= nil then
6671         local seen_auto_identifiers = {}
6672         for _, auto_identifier in ipairs(auto_identifiers) do
6673             if seen_auto_identifiers[auto_identifier] == nil then
6674                 seen_auto_identifiers[auto_identifier] = true
6675             if seen_identifiers[auto_identifier] == nil then
6676                 seen_identifiers[auto_identifier] = true
6677                 table.insert(normalized_attributes,
6678                     "#" .. auto_identifier)
```

```

6679         else
6680             local auto_identifier_number = 1
6681             while true do
6682                 local numbered_auto_identifier = auto_identifier .. "-"
6683                                     .. auto_identifier_number
6684                 if seen_identifiers[numbered_auto_identifier] == nil then
6685                     seen_identifiers[numbered_auto_identifier] = true
6686                     table.insert(normalized_attributes,
6687                                 "#" .. numbered_auto_identifier)
6688                     break
6689                 end
6690                 auto_identifier_number = auto_identifier_number + 1
6691             end
6692         end
6693     end
6694 end
6695 end
6696
6697 -- sort and deduplicate normalized attributes
6698 table.sort(normalized_attributes)
6699 local seen_normalized_attributes = {}
6700 local deduplicated_normalized_attributes = {}
6701 for _, attribute in ipairs(normalized_attributes) do
6702     if seen_normalized_attributes[attribute] == nil then
6703         seen_normalized_attributes[attribute] = true
6704         table.insert(deduplicated_normalized_attributes, attribute)
6705     end
6706 end
6707
6708 return deduplicated_normalized_attributes
6709 end
6710
6711 function self.attributes(attributes, should_normalize_attributes)
6712     local normalized_attributes
6713     if should_normalize_attributes == false then
6714         normalized_attributes = attributes
6715     else
6716         normalized_attributes = normalize_attributes(attributes)
6717     end
6718
6719     local buf = {}
6720     local key, value
6721     for _, attribute in ipairs(normalized_attributes) do
6722         if attribute:sub(1, 1) == "#" then
6723             table.insert(buf, {"\\markdownRendererAttributeIdentifier{",
6724                                 attribute:sub(2), "}"}))
6725         elseif attribute:sub(1, 1) == "." then

```

```

6726         table.insert(buf, {"\\markdownRendererAttributeClassName{",
6727                               attribute:sub(2), "}"}))
6728     else
6729         key, value = attribute:match(key_value_regex)
6730         table.insert(buf, {"\\markdownRendererAttributeKeyValue{",
6731                               key, "}{" , value, "}"}))
6732     end
6733 end
6734
6735 return buf
6736 end

```

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

```

6737 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`.

```

6738 self.attribute_type_levels = {}
6739 setmetatable(self.attribute_type_levels,
6740             { __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`.

```

6741 local function apply_attributes()
6742     local buf = {}
6743     for i = 1, #self.active_attributes do
6744         local start_output = self.active_attributes[i][3]
6745         if start_output ~= nil then
6746             table.insert(buf, start_output)
6747         end
6748     end
6749     return buf
6750 end
6751
6752 local function tear_down_attributes()
6753     local buf = {}
6754     for i = #self.active_attributes, 1, -1 do
6755         local end_output = self.active_attributes[i][4]
6756         if end_output ~= nil then
6757             table.insert(buf, end_output)
6758         end
6759     end
6760     return buf
6761 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.

```

6762 function self.push_attributes(attribute_type, attributes,
6763                               start_output, end_output)
6764     local attribute_type_level
6765     = self.attribute_type_levels[attribute_type]
6766     self.attribute_type_levels[attribute_type]
6767     = attribute_type_level + 1
6768
6769     -- index attributes in a hash table for easy lookup
6770     attributes = attributes or {}
6771     for i = 1, #attributes do
6772         attributes[attributes[i]] = true
6773     end
6774
6775     local buf = {}
6776     -- handle slicing
6777     if attributes["#" .. self.slice_end_identifier] ~= nil and
6778         self.slice_end_type == "^" then
6779         if self.is_writing then
6780             table.insert(buf, self.undosep())
6781             table.insert(buf, tear_down_attributes())
6782         end
6783         self.is_writing = false
6784     end
6785     if attributes["#" .. self.slice_begin_identifier] ~= nil and
6786         self.slice_begin_type == "^" then
6787         table.insert(buf, apply_attributes())
6788         self.is_writing = true
6789     end
6790     if self.is_writing and start_output ~= nil then
6791         table.insert(buf, start_output)
6792     end
6793     table.insert(self.active_attributes,
6794                 {attribute_type, attributes,
6795                  start_output, end_output})
6796     return buf
6797 end
6798

```

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](#)).

```

6799 function self.pop_attributes(attribute_type)
6800     local buf = {}
6801     -- pop attributes until we find attributes of correct type
6802     -- or until no attributes remain
6803     local current_attribute_type = false
6804     while current_attribute_type ~= attribute_type and
6805           #self.active_attributes > 0 do
6806         local attributes, _, end_output
6807         current_attribute_type, attributes, _, end_output = table.unpack(
6808             self.active_attributes[#self.active_attributes])
6809         local attribute_type_level
6810         = self.attribute_type_levels[current_attribute_type]
6811         self.attribute_type_levels[current_attribute_type]
6812         = attribute_type_level - 1
6813         if self.is_writing and end_output ~= nil then
6814             table.insert(buf, end_output)
6815         end
6816         table.remove(self.active_attributes, #self.active_attributes)
6817         -- handle slicing
6818         if attributes["#" .. self.slice_end_identifier] ~= nil
6819             and self.slice_end_type == "$" then
6820             if self.is_writing then
6821                 table.insert(buf, self.undosep())
6822                 table.insert(buf, tear_down_attributes())
6823             end
6824             self.is_writing = false
6825         end
6826         if attributes["#" .. self.slice_begin_identifier] ~= nil and
6827             self.slice_begin_type == "$" then
6828             self.is_writing = true
6829             table.insert(buf, apply_attributes())
6830         end
6831     end
6832     return buf
6833 end

```

Create an auto identifier string by stripping and converting characters from string [s](#).

```

6834 local function create_auto_identifier(s)
6835     local buffer = {}
6836     local prev_space = false
6837     local letter_found = false
6838     local normalized_s = s
6839     if not options.unicodeNormalization
6840         or options.unicodeNormalizationForm ~= "nfc" then
6841         normalized_s = uni_algos.normalize.NFC(normalized_s)

```

```

6842     end
6843
6844     for _, code in utf8.codes(normalized_s) do
6845         local char = utf8.char(code)
6846
6847         -- Remove everything up to the first letter.
6848         if not letter_found then
6849             local is_letter = lpeg.match(
6850                 parsers.unicode.following_alpha,
6851                 char
6852             )
6853             if is_letter then
6854                 letter_found = true
6855             else
6856                 goto continue
6857             end
6858         end
6859
6860         -- Remove all non-alphanumeric characters, except underscores,
6861         -- hyphens, and periods.
6862         if not lpeg.match(
6863             ( parsers.underscore
6864               + parsers.dash
6865               + parsers.period
6866               + parsers.unicode.following_word
6867               + parsers.unicode.following_whitespace ),
6868             char
6869         ) then
6870             goto continue
6871         end
6872
6873         -- Replace all spaces and newlines with hyphens.
6874         if lpeg.match(
6875             ( parsers.newline
6876               + parsers.unicode.following_whitespace ),
6877             char
6878         ) then
6879             char = "-"
6880             if prev_space then
6881                 goto continue
6882             else
6883                 prev_space = true
6884             end
6885         else
6886             -- Case-fold all alphabetic characters.
6887             char = uni_algos.case.casefold(char)
6888             prev_space = false

```

```

6889         end
6890
6891         table.insert(buffer, char)
6892
6893         ::continue::
6894     end
6895
6896     if prev_space then
6897         table.remove(buffer)
6898     end
6899
6900     local identifier = #buffer == 0 and "section"
6901                     or table.concat(buffer, "")
6902     return identifier
6903 end

```

Create an GitHub-flavored auto identifier string by stripping and converting characters from string `s`.

```

6904 local function create_gfm_auto_identifier(s)
6905     local buffer = {}
6906     local prev_space = false
6907     local letter_found = false
6908     local normalized_s = s
6909     if not options.unicodeNormalization
6910         or options.unicodeNormalizationForm ~= "nfc" then
6911         normalized_s = uni_algos.normalize.NFC(normalized_s)
6912     end
6913
6914     for _, code in utf8.codes(normalized_s) do
6915         local char = utf8.char(code)
6916
6917         -- Remove everything up to the first non-space.
6918         if not letter_found then
6919             local is_letter = not lpeg.match(
6920                 parsers.unicode.following_whitespace,
6921                 char
6922             )
6923             if is_letter then
6924                 letter_found = true
6925             else
6926                 goto continue
6927             end
6928         end
6929
6930         -- Remove all non-alphanumeric characters, except underscores
6931         -- and hyphens.
6932         if not lpeg.match(

```

```

6933     ( parsers.underscore
6934     + parsers.dash
6935     + parsers.unicode.following_word
6936     + parsers.unicode.following_whitespace ),
6937     char
6938   ) then
6939     prev_space = false
6940     goto continue
6941   end
6942
6943   -- Replace all spaces and newlines with hyphens.
6944   if lpeg.match(
6945     ( parsers.newline
6946     + parsers.unicode.following_whitespace ),
6947     char
6948   ) then
6949     char = "-"
6950     if prev_space then
6951       goto continue
6952     else
6953       prev_space = true
6954     end
6955   else
6956     -- Case-fold all alphabetic characters.
6957     char = uni_algos.case.casefold(char)
6958     prev_space = false
6959   end
6960
6961   table.insert(buffer, char)
6962
6963   ::continue::
6964 end
6965
6966 if prev_space then
6967   table.remove(buffer)
6968 end
6969
6970 local identifier = #buffer == 0 and "section"
6971                  or table.concat(buffer, "")
6972 return identifier
6973 end

```

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

```

6974 self.secbegin_text = "\\markdownRendererSectionBegin\n"
6975 self.seccend_text = "\n\\markdownRendererSectionEnd "
6976 function self.heading(s, level, attributes)

```

```

6977     local buf = {}
6978     local flat_text, inlines = table.unpack(s)
6979
6980     -- push empty attributes for implied sections
6981     while self.attribute_type_levels["heading"] < level - 1 do
6982         table.insert(buf,
6983             self.push_attributes("heading",
6984                 nil,
6985                 self.secbegin_text,
6986                 self.secend_text))
6987     end
6988
6989     -- pop attributes for sections that have ended
6990     while self.attribute_type_levels["heading"] >= level do
6991         table.insert(buf, self.pop_attributes("heading"))
6992     end
6993
6994     -- construct attributes for the new section
6995     local auto_identifiers = {}
6996     if self.options.autoIdentifiers then
6997         table.insert(auto_identifiers, create_auto_identifier(flat_text))
6998     end
6999     if self.options.gfmAutoIdentifiers then
7000         table.insert(auto_identifiers,
7001             create_gfm_auto_identifier(flat_text))
7002     end
7003     local normalized_attributes = normalize_attributes(attributes,
7004                                                         auto_identifiers)
7005
7006     -- push attributes for the new section
7007     local start_output = {}
7008     local end_output = {}
7009     table.insert(start_output, self.secbegin_text)
7010     table.insert(end_output, self.secend_text)
7011
7012     table.insert(buf, self.push_attributes("heading",
7013         normalized_attributes,
7014         start_output,
7015         end_output))
7016     assert(self.attribute_type_levels["heading"] == level)
7017
7018     -- render the heading and its attributes
7019     if self.is_writing and #normalized_attributes > 0 then
7020         table.insert(buf,
7021             "\\markdownRendererHeaderAttributeContextBegin\n")
7022         table.insert(buf, self.attributes(normalized_attributes, false))
7023     end

```

```

7024
7025     local cmd
7026     level = level + options.shiftHeadings
7027     if level <= 1 then
7028         cmd = "\\markdownRendererHeadingOne"
7029     elseif level == 2 then
7030         cmd = "\\markdownRendererHeadingTwo"
7031     elseif level == 3 then
7032         cmd = "\\markdownRendererHeadingThree"
7033     elseif level == 4 then
7034         cmd = "\\markdownRendererHeadingFour"
7035     elseif level == 5 then
7036         cmd = "\\markdownRendererHeadingFive"
7037     elseif level >= 6 then
7038         cmd = "\\markdownRendererHeadingSix"
7039     else
7040         cmd = ""
7041     end
7042     if self.is_writing then
7043         table.insert(buf, {cmd, "{", inlines, "}"})
7044     end
7045
7046     if self.is_writing and #normalized_attributes > 0 then
7047         table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd{")
7048     end
7049
7050     return buf
7051 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.

```

7052 function self.get_state()
7053     return {
7054         is_writing=self.is_writing,
7055         flatten_inlines=self.flatten_inlines,
7056         active_attributes={table.unpack(self.active_attributes)},
7057     }
7058 end

```

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

```

7059 function self.set_state(s)
7060     local previous_state = self.get_state()
7061     for key, value in pairs(s) do
7062         self[key] = value
7063     end
7064     return previous_state
7065 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`.

```

7066 function self.defer_call(f)
7067     local previous_state = self.get_state()
7068     return function(...)
7069         local state = self.set_state(previous_state)
7070         local return_value = f(...)
7071         self.set_state(state)
7072         return return_value
7073     end
7074 end
7075
7076 return self
7077 end

```

### 3.1.4 Parsers

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

```

7078 parsers = {}

```

#### 3.1.4.1 Basic Parsers

```

7079 parsers.percent = P("%")
7080 parsers.at = P("@")
7081 parsers.comma = P(",")
7082 parsers.asterisk = P("*")
7083 parsers.dash = P("-")
7084 parsers.plus = P("+")
7085 parsers.underscore = P("_")
7086 parsers.period = P(".")
7087 parsers.hash = P("#")
7088 parsers.dollar = P("$")
7089 parsers.ampersand = P("&")
7090 parsers.backtick = P("`")
7091 parsers.less = P("<")
7092 parsers.more = P(">")
7093 parsers.space = P(" ")
7094 parsers.squote = P("'")
7095 parsers.dquote = P('"')
7096 parsers.lparent = P("(")
7097 parsers.rparent = P(")")
7098 parsers.lbracket = P("[")
7099 parsers.rbracket = P("]")
7100 parsers.lbrace = P("{")

```

```

7101 parsers.rbrace           = P("}")
7102 parsers.circumflex       = P("^")
7103 parsers.slash             = P("/")
7104 parsers.equal             = P("=")
7105 parsers.colon             = P(":")
7106 parsers.semicolon        = P(";")
7107 parsers.exclamation       = P("!")
7108 parsers.pipe              = P("|")
7109 parsers.tilde             = P("~")
7110 parsers.backslash         = P("\\")
7111 parsers.tab               = P("\t")
7112 parsers.newline           = P("\n")
7113
7114 parsers.digit             = R("09")
7115 parsers.hexdigit          = R("09","af","AF")
7116 parsers.letter            = R("AZ","az")
7117 parsers.alphanumeric      = R("AZ","az","09")
7118 parsers.keyword           = parsers.letter
7119                           * (parsers.alphanumeric + parsers.dash)^0
7120
7121 parsers.doubleasterisks    = P("**")
7122 parsers.doubleunderscores  = P("__")
7123 parsers.doubletildes       = P("~")
7124 parsers.fourspace         = P(" ")
7125
7126 parsers.any                = P(1)
7127 parsers.succeed            = P(true)
7128 parsers.fail               = P(false)
7129
7130 parsers.internal_punctuation = S(",:;.?.")
7131 parsers.ascii_punctuation  = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")
7132

```

### 3.1.5 Unicode data

This section documents different Unicode character categories recognized by the markdown reader. The parsers for the different categories are organized in the table [parsers.unicode\\_data](#) according to the number of bytes occupied after conversion to UTF8.

All code from this section will be executed during the compilation of the Markdown package and the standard output will be stored in a file named [markdown-unicode-data.lua](#) with the precompiled parser of Unicode punctuation.

```

7133 ;(function()
7134   local pathname = assert(kpse.find_file("UnicodeData.txt"),
7135     [[Could not locate file "UnicodeData.txt"]])
7136   local file = assert(io.open(pathname, "r"),

```



```
7137     [[Could not open file "UnicodeData.txt"]])
```

In order to minimize the size and speed of the parser, we will first construct prefix tree of UTF-8 encodings for all codepoints of a given Unicode category and code length.

```
7138     local categories = {"L", "N", "P", "Pc", "S", "Z"}
7139     local prefix_trees = {}
7140     for _, category in ipairs(categories) do
7141         prefix_trees[category] = {}
7142         for char_length = 1, 4 do
7143             prefix_trees[category][char_length] = {}
7144         end
7145     end
7146     for line in file:lines() do
7147         local codepoint, full_category = line:match("^(%x+);[^\;]*;(%a*)")
7148         assert(#full_category >= 1)
7149         local major_category = full_category:sub(1, 1)
7150         for _, category in ipairs({full_category, major_category}) do
7151             if prefix_trees[category] == nil then
7152                 goto continue
7153             end
7154             local code = utf8.char(tonumber(codepoint, 16))
7155             local node = prefix_trees[category][#code]
7156             for i = 1, #code do
7157                 local byte = code:sub(i, i)
7158                 if i < #code then
7159                     if node[byte] == nil then
7160                         node[byte] = {}
7161                     end
7162                     node = node[byte]
7163                 else
7164                     table.insert(node, byte)
7165                 end
7166             end
7167             ::continue::
7168         end
7169     end
7170     assert(file:close())
7171
```

Next, we will construct parsers out of the prefix trees.

```
7172     local function depth_first_search(node, path, visit, leave)
7173         visit(node, path)
7174         for label, child in pairs(node) do
7175             if type(child) == "table" then
7176                 depth_first_search(child, path .. label, visit, leave)
7177             else
7178                 visit(child, path)
7179             end
7180         end
7181     end
```

```

7179         end
7180     end
7181     leave(node, path)
7182 end
7183
7184 print("M.categories = {}")
7185 print("local P = lpeg.P")
7186 print("local fail = P(false)")
7187 print("-- luacheck: push no max line length")
7188 for _, category in ipairs(categories) do
7189     print("M.categories." .. category .. " = {}")
7190     for length, prefix_tree in pairs(prefix_trees[category]) do
7191         local subparsers = {}
7192         depth_first_search(prefix_tree, "", function(node, path)
7193             if type(node) == "string" then
7194                 local suffix
7195                 if node == "]" then
7196                     suffix = "P('" .. node .. "')"
7197                 else
7198                     suffix = "P([[ " .. node .. " ]])"
7199                 end
7200                 if subparsers[path] ~= nil then
7201                     subparsers[path] = subparsers[path] .. " + " .. suffix
7202                 else
7203                     subparsers[path] = suffix
7204                 end
7205             end
7206         end, function(_, path)
7207             if #path > 0 then
7208                 local byte = path:sub(#path, #path)
7209                 local parent_path = path:sub(1, #path-1)
7210                 local prefix
7211                 if byte == "]" then
7212                     prefix = "P('" .. byte .. "')"
7213                 else
7214                     prefix = "P([[ " .. byte .. " ]])"
7215                 end
7216                 local suffix
7217                 if subparsers[path]:find(" %+ ") then
7218                     suffix = prefix .. " * (" .. subparsers[path] .. ")"
7219                 else
7220                     suffix = prefix .. " * " .. subparsers[path]
7221                 end
7222                 if subparsers[parent_path] ~= nil then
7223                     subparsers[parent_path] = subparsers[parent_path]
7224                         .. " + " .. suffix
7225                 else

```

```

7226         subparsers[parent_path] = suffix
7227     end
7228     else
7229         print(
7230             "M.categories." .. category .. "[" .. length .. "]" = "
7231             .. (subparsers[path] or "fail")
7232         )
7233     end
7234 end)
7235 end
7236 end
7237 print("-- luacheck: pop")
7238 end)()
7239 print("return M")

```

Back in the Markdown package, we will load the precompiled parsers of Unicode categories.

```

7240 local unicode_data = require("markdown-unicode-data")
7241 if metadata.version ~= unicode_data.metadata.version then
7242     util.warning(
7243         "markdown.lua " .. metadata.version .. " used with " ..
7244         "markdown-unicode-data.lua " .. unicode_data.metadata.version .. "."
7245     )
7246 end

```

Finally, we define high-level parsers for specific types of characters that are interesting for us.

```

7247 parsers.unicode = {}
7248 parsers.unicode.preceding_punctuation = parsers.fail
7249 parsers.unicode.following_punctuation = parsers.fail
7250 parsers.unicode.following_alpha = parsers.fail
7251 parsers.unicode.following_word = parsers.fail
7252 parsers.unicode.preceding_whitespace = parsers.fail
7253 parsers.unicode.following_whitespace = parsers.fail
7254 for n = 1, 4 do

```

For punctuation, accept any characters from Unicode categories P (punctuation) and S (symbol), as mandated by the CommonMark standard<sup>35</sup>.

(CommonMark Spec, Version 0.31.2 (2024-01-28))

```

7255     local punctuation_of_length_n
7256     = unicode_data.categories.P[n]
7257     + unicode_data.categories.S[n]
7258     parsers.unicode.preceding_punctuation
7259     = parsers.unicode.preceding_punctuation

```

<sup>35</sup>See <https://spec.commonmark.org/0.31.2/#unicode-punctuation-character>.

```

7260     + B(punctuation_of_length_n)
7261     parsers.unicode.following_punctuation
7262     = parsers.unicode.following_punctuation
7263     + #punctuation_of_length_n

```

For alphabetical characters, accept any characters from Unicode category L (letter), similar to the character class ‘Unicode.

```

7264     local alpha_of_length_n = unicode_data.categories.L[n]
7265     parsers.unicode.following_alpha
7266     = parsers.unicode.following_alpha
7267     + alpha_of_length_n

```

For word characters, accept any characters from Unicode categories L (letter), N (number), and Pc (connector punctuation), similar to the character class ‘

```

7268     local word_of_length_n
7269     = unicode_data.categories.L[n]
7270     + unicode_data.categories.N[n]
7271     + unicode_data.categories.Pc[n]
7272     parsers.unicode.following_word
7273     = parsers.unicode.following_word
7274     + word_of_length_n

```

For space characters, accept any characters from Unicode category Z (separator), as well as the ASCII control characters 9 (horizontal tab) through 13 (carriage return), similar to the character class ‘Lua library Selene Unicode.

```

7275     local whitespace_of_length_n = unicode_data.categories.Z[n]
7276     if n == 1 then
7277         whitespace_of_length_n
7278         = whitespace_of_length_n
7279         + R("\t\r")
7280     end
7281     parsers.unicode.preceding_whitespace
7282     = parsers.unicode.preceding_whitespace
7283     + B(whitespace_of_length_n)
7284     parsers.unicode.following_whitespace
7285     = parsers.unicode.following_whitespace
7286     + #whitespace_of_length_n
7287 end
7288
7289 parsers.escapable           = parsers.ascii_punctuation
7290 parsers.anyescaped         = parsers.backslash / ""
7291                             * parsers.escapable
7292                             + parsers.any
7293
7294 parsers.spacechar          = S("\t ")
7295 parsers.spacing            = S(" \n\r\t")
7296 parsers.nonpacechar        = parsers.any - parsers.spacing
7297 parsers.optionalspace      = parsers.spacechar^0

```

```

7298
7299 parsers.normalchar      = parsers.any - (V("SpecialChar")
7300                                + parsers.spacing)
7301 parsers.eof              = -parsers.any
7302 parsers.nonindentspace   = parsers.space-3 * - parsers.spacechar
7303 parsers.indent           = parsers.space-3 * parsers.tab
7304                          + parsers.fourspace / ""
7305 parsers.linechar         = P(1 - parsers.newline)
7306
7307 parsers.blankline        = parsers.optionalspace
7308                          * parsers.newline / "\n"
7309 parsers.blanklines       = parsers.blankline0
7310 parsers.skipblanklines   = ( parsers.optionalspace
7311                          * parsers.newline)0
7312 parsers.indentedline     = parsers.indent /""
7313                          * C( parsers.linechar1
7314                          * parsers.newline-1)
7315 parsers.optionallyindentedline = parsers.indent-1 /""
7316                          * C( parsers.linechar1
7317                          * parsers.newline-1)
7318 parsers.sp               = parsers.spacing0
7319 parsers.spnl             = parsers.optionalspace
7320                          * ( parsers.newline
7321                          * parsers.optionalspace)-1
7322 parsers.line              = parsers.linechar0 * parsers.newline
7323 parsers.nonemptyline     = parsers.line - parsers.blankline

```

### 3.1.5.1 Parsers Used for Indentation

```

7324
7325 parsers.leader           = parsers.space-3
7326

```

Check if a trail exists and is non-empty in the indent table `indent_table`.

```

7327 local function has_trail(indent_table)
7328   return indent_table ~= nil and
7329     indent_table.trail ~= nil and
7330     next(indent_table.trail) ~= nil
7331 end
7332

```

Check if indent table `indent_table` has any indents.

```

7333 local function has_indents(indent_table)
7334   return indent_table ~= nil and
7335     indent_table.indents ~= nil and
7336     next(indent_table.indents) ~= nil
7337 end
7338

```

Add a trail `trail_info` to the indent table `indent_table`.

```
7339 local function add_trail(indent_table, trail_info)
7340     indent_table.trail = trail_info
7341     return indent_table
7342 end
7343
```

Remove a trail `trail_info` from the indent table `indent_table`.

```
7344 local function remove_trail(indent_table)
7345     indent_table.trail = nil
7346     return indent_table
7347 end
7348
```

Update the indent table `indent_table` by adding or removing a new indent `add`.

```
7349 local function update_indent_table(indent_table, new_indent, add)
7350     indent_table = remove_trail(indent_table)
7351
7352     if not has_indents(indent_table) then
7353         indent_table.indents = {}
7354     end
7355
7356     if add then
7357         indent_table.indents[#indent_table.indents + 1] = new_indent
7358     else
7359         if indent_table.indents[#indent_table.indents].name
7360             == new_indent.name then
7361             indent_table.indents[#indent_table.indents] = nil
7362         end
7363     end
7364
7365     return indent_table
7366 end
7367
7368
```

Remove an indent by its name `name`.

```
7369 local function remove_indent(name)
7370     local remove_indent_level =
7371         function(s, i, indent_table) -- luacheck: ignore s i
7372             indent_table = update_indent_table(indent_table, {name=name},
7373                                                 false)
7374             return true, indent_table
7375         end
7376
7377     return Cg(Cmt(Cb("indent_info"), remove_indent_level), "indent_info")
7378 end
7379
```

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`.

```

7380 local function process_starter_spacing(indent, spacing,
7381                                     minimum, left_strip_length)
7382     left_strip_length = left_strip_length or 0
7383
7384     local count = 0
7385     local tab_value = 4 - (indent) % 4
7386
7387     local code_started, minimum_found = false, false
7388     local code_start, minimum_remainder = "", ""
7389
7390     local left_total_stripped = 0
7391     local full_remainder = ""
7392
7393     if spacing ~= nil then
7394         for i = 1, #spacing do
7395             local character = spacing:sub(i, i)
7396
7397             if character == "\t" then
7398                 count = count + tab_value
7399                 tab_value = 4
7400             elseif character == " " then
7401                 count = count + 1
7402                 tab_value = 4 - (1 - tab_value) % 4
7403             end
7404
7405             if (left_strip_length ~= 0) then
7406                 local possible_to_strip = math.min(count, left_strip_length)
7407                 count = count - possible_to_strip
7408                 left_strip_length = left_strip_length - possible_to_strip
7409                 left_total_stripped = left_total_stripped + possible_to_strip
7410             else
7411                 full_remainder = full_remainder .. character
7412             end
7413
7414             if (minimum_found) then
7415                 minimum_remainder = minimum_remainder .. character
7416             elseif (count >= minimum) then
7417                 minimum_found = true
7418                 minimum_remainder = minimum_remainder
7419                     .. string.rep(" ", count - minimum)
7420             end

```

```

7421
7422     if (code_started) then
7423         code_start = code_start .. character
7424     elseif (count >= minimum + 4) then
7425         code_started = true
7426         code_start = code_start
7427             .. string.rep(" ", count - (minimum + 4))
7428     end
7429 end
7430 end
7431
7432 local remainder
7433 if (code_started) then
7434     remainder = code_start
7435 else
7436     remainder = string.rep(" ", count - minimum)
7437 end
7438
7439 local is_minimum = count >= minimum
7440 return {
7441     is_code = code_started,
7442     remainder = remainder,
7443     left_total_stripped = left_total_stripped,
7444     is_minimum = is_minimum,
7445     minimum_remainder = minimum_remainder,
7446     total_length = count,
7447     full_remainder = full_remainder
7448 }
7449 end
7450

```

Count the total width of all indents in the indent table `indent_table`.

```

7451 local function count_indent_tab_level(indent_table)
7452     local count = 0
7453     if not has_indents(indent_table) then
7454         return count
7455     end
7456
7457     for i=1, #indent_table.indents do
7458         count = count + indent_table.indents[i].length
7459     end
7460     return count
7461 end
7462

```

Count the total width of a delimiter `delimiter`.

```

7463 local function total_delimiter_length(delimiter)
7464     local count = 0

```



```

7465   if type(delimiter) == "string" then return #delimiter end
7466   for _, value in pairs(delimiter) do
7467     count = count + total_delimiter_length(value)
7468   end
7469   return count
7470 end
7471

```

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`.

```

7472 local function process_starter_indent(_, _, indent_table, starter,
7473                                     is_blank, indent_type, breakable)
7474   local last_trail = starter[1]
7475   local delimiter = starter[2]
7476   local raw_new_trail = starter[3]
7477
7478   if indent_type == "bq" and not breakable then
7479     indent_table.ignore_blockquote_blank = true
7480   end
7481
7482   if has_trail(indent_table) then
7483     local trail = indent_table.trail
7484     if trail.is_code then
7485       return false
7486     end
7487     last_trail = trail.remainder
7488   else
7489     local sp = process_starter_spacing(0, last_trail, 0, 0)
7490
7491     if sp.is_code then
7492       return false
7493     end
7494     last_trail = sp.remainder
7495   end
7496
7497   local preceding_indentation = count_indent_tab_level(indent_table) % 4
7498   local last_trail_length = #last_trail
7499   local delimiter_length = total_delimiter_length(delimiter)
7500
7501   local total_indent_level = preceding_indentation + last_trail_length
7502                           + delimiter_length
7503
7504   local sp = {}
7505   if not is_blank then
7506     sp = process_starter_spacing(total_indent_level, raw_new_trail,
7507                                0, 1)
7508   end

```

```

7509
7510 local del_trail_length = sp.left_total_stripped
7511 if is_blank then
7512     del_trail_length = 1
7513 elseif not sp.is_code then
7514     del_trail_length = del_trail_length + #sp.remainder
7515 end
7516
7517 local indent_length = last_trail_length + delimiter_length
7518                     + del_trail_length
7519 local new_indent_info = {name=indent_type, length=indent_length}
7520
7521 indent_table = update_indent_table(indent_table, new_indent_info,
7522                                   true)
7523 indent_table = add_trail(indent_table,
7524                          {is_code=sp.is_code,
7525                           remainder=sp.remainder,
7526                           total_length=sp.total_length,
7527                           full_remainder=sp.full_remainder})
7528
7529 return true, indent_table
7530 end
7531

```

Return the pattern corresponding with the indent name `name`.

```

7532 local function decode_pattern(name)
7533     local delimiter = parsers.succeed
7534     if name == "bq" then
7535         delimiter = parsers.more
7536     end
7537
7538     return C(parsers.optionalspace) * C(delimiter)
7539         * C(parsers.optionalspace) * Cp()
7540 end
7541

```

Find the first blank-only indent of the indent table `indent_table` followed by blank-only indents.

```

7542 local function left_blank_starter(indent_table)
7543     local blank_starter_index
7544
7545     if not has_indents(indent_table) then
7546         return
7547     end
7548
7549     for i = #indent_table.indents,1,-1 do
7550         local value = indent_table.indents[i]
7551         if value.name == "li" then

```

```

7552     blank_starter_index = i
7553   else
7554     break
7555   end
7556 end
7557
7558 return blank_starter_index
7559 end
7560

```

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.

```

7561 local function traverse_indent(s, i, indent_table, is_optional,
7562                               is_blank, current_line_indents)
7563   local new_index = i
7564
7565   local preceding_indentation = 0
7566   local current_trail = {}
7567
7568   local blank_starter = left_blank_starter(indent_table)
7569
7570   if current_line_indents == nil then
7571     current_line_indents = {}
7572   end
7573
7574   for index = 1, #indent_table.indents do
7575     local value = indent_table.indents[index]
7576     local pattern = decode_pattern(value.name)
7577
7578     -- match decoded pattern
7579     local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
7580     if new_indent_info == nil then
7581       local blankline_end = lpeg.match(
7582         Ct(parsers.blankline * Cg(Cp(), "pos")), s, new_index)
7583       if is_optional or not indent_table.ignore_blockquote_blank
7584         or not blankline_end then
7585         return is_optional, new_index, current_trail,
7586             current_line_indents
7587       end
7588
7589       return traverse_indent(s, tonumber(blankline_end.pos),
7590                             indent_table, is_optional, is_blank,
7591                             current_line_indents)
7592     end

```

```

7593
7594     local raw_last_trail = new_indent_info[1]
7595     local delimiter = new_indent_info[2]
7596     local raw_new_trail = new_indent_info[3]
7597     local next_index = new_indent_info[4]
7598
7599     local space_only = delimiter == ""
7600
7601     -- check previous trail
7602     if not space_only and next(current_trail) == nil then
7603         local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
7604         current_trail = {is_code=sp.is_code, remainder=sp.remainder,
7605                         total_length=sp.total_length,
7606                         full_remainder=sp.full_remainder}
7607     end
7608
7609     if next(current_trail) ~= nil then
7610         if not space_only and current_trail.is_code then
7611             return is_optional, new_index, current_trail,
7612                    current_line_indents
7613         end
7614         if current_trail.internal_remainder ~= nil then
7615             raw_last_trail = current_trail.internal_remainder
7616         end
7617     end
7618
7619     local raw_last_trail_length = 0
7620     local delimiter_length = 0
7621
7622     if not space_only then
7623         delimiter_length = #delimiter
7624         raw_last_trail_length = #raw_last_trail
7625     end
7626
7627     local total_indent_level = preceding_indentation
7628                             + raw_last_trail_length + delimiter_length
7629
7630     local spacing_to_process
7631     local minimum = 0
7632     local left_strip_length = 0
7633
7634     if not space_only then
7635         spacing_to_process = raw_new_trail
7636         left_strip_length = 1
7637     else
7638         spacing_to_process = raw_last_trail
7639         minimum = value.length

```

```

7640     end
7641
7642     local sp = process_starter_spacing(total_indent_level,
7643                                     spacing_to_process, minimum,
7644                                     left_strip_length)
7645
7646     if space_only and not sp.is_minimum then
7647         return is_optional or (is_blank and blank_starter <= index),
7648             new_index, current_trail, current_line_indents
7649     end
7650
7651     local indent_length = raw_last_trail_length + delimiter_length
7652                       + sp.left_total_stripped
7653
7654     -- update info for the next pattern
7655     if not space_only then
7656         preceding_indentation = preceding_indentation + indent_length
7657     else
7658         preceding_indentation = preceding_indentation + value.length
7659     end
7660
7661     current_trail = {is_code=sp.is_code, remainder=sp.remainder,
7662                    internal_remainder=sp.minimum_remainder,
7663                    total_length=sp.total_length,
7664                    full_remainder=sp.full_remainder}
7665
7666     current_line_indents[#current_line_indents + 1] = new_indent_info
7667     new_index = next_index
7668 end
7669
7670 return true, new_index, current_trail, current_line_indents
7671 end
7672

```

Check if a code trail is expected.

```

7673 local function check_trail(expect_code, is_code)
7674     return (expect_code and is_code) or (not expect_code and not is_code)
7675 end
7676

```

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](#).

```

7677 local check_trail_joined =
7678     function(s, i, indent_table, -- luacheck: ignore s i
7679             spacing, expect_code, omit_remainder)
7680         local is_code
7681         local remainder

```

```

7682
7683     if has_trail(indent_table) then
7684         local trail = indent_table.trail
7685         is_code = trail.is_code
7686         if is_code then
7687             remainder = trail.remainder
7688         else
7689             remainder = trail.full_remainder
7690         end
7691     else
7692         local sp = process_starter_spacing(0, spacing, 0, 0)
7693         is_code = sp.is_code
7694         if is_code then
7695             remainder = sp.remainder
7696         else
7697             remainder = sp.full_remainder
7698         end
7699     end
7700
7701     local result = check_trail(expect_code, is_code)
7702     if omit_remainder then
7703         return result
7704     end
7705     return result, remainder
7706 end
7707

```

Check if the current trail of the `indent_table` is of length between `min` and `max`.

```

7708 local check_trail_length =
7709     function(s, i, indent_table, -- luacheck: ignore s i
7710         spacing, min, max)
7711         local trail
7712
7713         if has_trail(indent_table) then
7714             trail = indent_table.trail
7715         else
7716             trail = process_starter_spacing(0, spacing, 0, 0)
7717         end
7718
7719         local total_length = trail.total_length
7720         if total_length == nil then
7721             return false
7722         end
7723
7724         return min <= total_length and total_length <= max
7725     end
7726

```

Check the indentation of the continuation line, optionally with the mode `is_optional` selected. Check blank line exclusively with `is_blank`.

```
7727 local function check_continuation_indentation(s, i, indent_table,
7728                                             is_optional, is_blank)
7729   if not has_indents(indent_table) then
7730     return true
7731   end
7732
7733   local passes, new_index, current_trail, current_line_indents =
7734     traverse_indent(s, i, indent_table, is_optional, is_blank)
7735
7736   if passes then
7737     indent_table.current_line_indents = current_line_indents
7738     indent_table = add_trail(indent_table, current_trail)
7739     return new_index, indent_table
7740   end
7741   return false
7742 end
7743
```

Get name of the last indent from the `indent_table`.

```
7744 local function get_last_indent_name(indent_table)
7745   if has_indents(indent_table) then
7746     return indent_table.indents[#indent_table.indents].name
7747   end
7748 end
7749
```

Remove the remainder altogether if the last indent from the `indent_table` is blank-only.

```
7750 local function remove_remainder_if_blank(indent_table, remainder)
7751   if get_last_indent_name(indent_table) == "li" then
7752     return ""
7753   end
7754   return remainder
7755 end
7756
```

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.

```
7757 local check_trail_type =
7758   function(s, i, -- luacheck: ignore s i
7759           trail, spacing, trail_type)
7760     if trail == nil then
7761       trail = process_starter_spacing(0, spacing, 0, 0)
7762     end
7763
7764     if trail_type == "non-code" then
```

```

7765     return check_trail(false, trail.is_code)
7766 end
7767 if trail_type == "code" then
7768     return check_trail(true, trail.is_code)
7769 end
7770 if trail_type == "full-code" then
7771     if (trail.is_code) then
7772         return i, trail.remainder
7773     end
7774     return i, ""
7775 end
7776 if trail_type == "full-any" then
7777     return i, trail.internal_remainder
7778 end
7779 end
7780

```

Stores or restores an `is_freezing` trail from indent table `indent_table`.

```

7781 local trail_freezing =
7782   function(s, i, -- luacheck: ignore s i
7783     indent_table, is_freezing)
7784     if is_freezing then
7785         if indent_table.is_trail_frozen then
7786             indent_table.trail = indent_table.frozen_trail
7787         else
7788             indent_table.frozen_trail = indent_table.trail
7789             indent_table.is_trail_frozen = true
7790         end
7791     else
7792         indent_table.frozen_trail = nil
7793         indent_table.is_trail_frozen = false
7794     end
7795     return true, indent_table
7796 end
7797

```

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`.

```

7798 local check_continuation_indentation_and_trail =
7799   function(s, i, indent_table, is_optional, is_blank, trail_type,
7800     reset_rem, omit_remainder)
7801     if not has_indents(indent_table) then
7802         local spacing, new_index = lpeg.match( C(parsers.spacechar~0)
7803           * Cp(), s, i)
7804         local result, remainder = check_trail_type(s, i,
7805           indent_table.trail, spacing, trail_type)
7806         if remainder == nil then

```



```

7807         if result then
7808             return new_index
7809         end
7810         return false
7811     end
7812     if result then
7813         return new_index, remainder
7814     end
7815     return false
7816 end
7817
7818 local passes, new_index, current_trail = traverse_indent(s, i,
7819     indent_table, is_optional, is_blank)
7820
7821 if passes then
7822     local spacing
7823     if current_trail == nil then
7824         local newer_spacing, newer_index = lpeg.match(
7825             C(parsers.spacechar^0) * Cp(), s, i)
7826         current_trail = process_starter_spacing(0, newer_spacing, 0, 0)
7827         new_index = newer_index
7828         spacing = newer_spacing
7829     else
7830         spacing = current_trail.remainder
7831     end
7832     local result, remainder = check_trail_type(s, new_index,
7833         current_trail, spacing, trail_type)
7834     if remainder == nil or omit_remainder then
7835         if result then
7836             return new_index
7837         end
7838         return false
7839     end
7840
7841     if is_blank and reset_rem then
7842         remainder = remove_remainder_if_blank(indent_table, remainder)
7843     end
7844     if result then
7845         return new_index, remainder
7846     end
7847     return false
7848 end
7849 return false
7850 end
7851

```

The following patterns check whitespace indentation at the start of a block.

```

7852 parsers.check_trail = Cmt( Cb("indent_info") * C(parsers.spacechar^0)

```

```

7853             * Cc(false), check_trail_joined)
7854
7855 parsers.check_trail_no_rem = Cmt( Cb("indent_info")
7856             * C(parsers.spacechar^0) * Cc(false)
7857             * Cc(true), check_trail_joined)
7858
7859 parsers.check_code_trail = Cmt( Cb("indent_info")
7860             * C(parsers.spacechar^0)
7861             * Cc(true), check_trail_joined)
7862
7863 parsers.check_trail_length_range = function(min, max)
7864   return Cmt( Cb("indent_info") * C(parsers.spacechar^0) * Cc(min)
7865             * Cc(max), check_trail_length)
7866 end
7867
7868 parsers.check_trail_length = function(n)
7869   return parsers.check_trail_length_range(n, n)
7870 end
7871

```

The following patterns handle trail backup, to prevent a failing pattern to modify it before passing it to the next.

```

7872 parsers.freeze_trail = Cg( Cmt(Cb("indent_info")
7873             * Cc(true), trail_freezing), "indent_info")
7874
7875 parsers.unfreeze_trail = Cg(Cmt(Cb("indent_info") * Cc(false),
7876             trail_freezing), "indent_info")
7877

```

The following patterns check indentation in continuation lines as defined by the container start.

```

7878 parsers.check_minimal_indent = Cmt(Cb("indent_info") * Cc(false),
7879             check_continuation_indentation)
7880
7881 parsers.check_optional_indent = Cmt(Cb("indent_info") * Cc(true),
7882             check_continuation_indentation)
7883
7884 parsers.check_minimal_blank_indent
7885   = Cmt( Cb("indent_info") * Cc(false)
7886         * Cc(true)
7887         , check_continuation_indentation)
7888

```

The following patterns check indentation in continuation lines as defined by the container start. Additionally the subsequent trail is also directly checked.

```

7889
7890 parsers.check_minimal_indent_and_trail =
7891   Cmt( Cb("indent_info")

```

```

7892     * Cc(false) * Cc(false) * Cc("non-code") * Cc(true)
7893     , check_continuation_indentation_and_trail)
7894
7895 parsers.check_minimal_indent_and_code_trail =
7896     Cmt( Cb("indent_info")
7897         * Cc(false) * Cc(false) * Cc("code") * Cc(false)
7898         , check_continuation_indentation_and_trail)
7899
7900 parsers.check_minimal_blank_indent_and_full_code_trail =
7901     Cmt( Cb("indent_info")
7902         * Cc(false) * Cc(true) * Cc("full-code") * Cc(true)
7903         , check_continuation_indentation_and_trail)
7904
7905 parsers.check_minimal_indent_and_any_trail =
7906     Cmt( Cb("indent_info")
7907         * Cc(false) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
7908         , check_continuation_indentation_and_trail)
7909
7910 parsers.check_minimal_blank_indent_and_any_trail =
7911     Cmt( Cb("indent_info")
7912         * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
7913         , check_continuation_indentation_and_trail)
7914
7915 parsers.check_minimal_blank_indent_and_any_trail_no_rem =
7916     Cmt( Cb("indent_info")
7917         * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(true)
7918         , check_continuation_indentation_and_trail)
7919
7920 parsers.check_optional_indent_and_any_trail =
7921     Cmt( Cb("indent_info")
7922         * Cc(true) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
7923         , check_continuation_indentation_and_trail)
7924
7925 parsers.check_optional_blank_indent_and_any_trail =
7926     Cmt( Cb("indent_info")
7927         * Cc(true) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
7928         , check_continuation_indentation_and_trail)
7929

```

The following patterns specify behaviour around newlines.

```

7930
7931 parsers.spnlc_noexc = parsers.optionalspace
7932     * ( parsers.newline
7933         * parsers.check_minimal_indent_and_any_trail)^-1
7934
7935 parsers.spnlc = parsers.optionalspace
7936     * (V("EndlineNoSub"))^-1
7937

```

```

7938 parsers.spnlc_sep = parsers.optionalspace * V("EndlineNoSub")
7939                   + parsers.spacechar^1
7940
7941 parsers.only_blank = parsers.spacechar^0
7942                   * (parsers.newline + parsers.eof)
7943

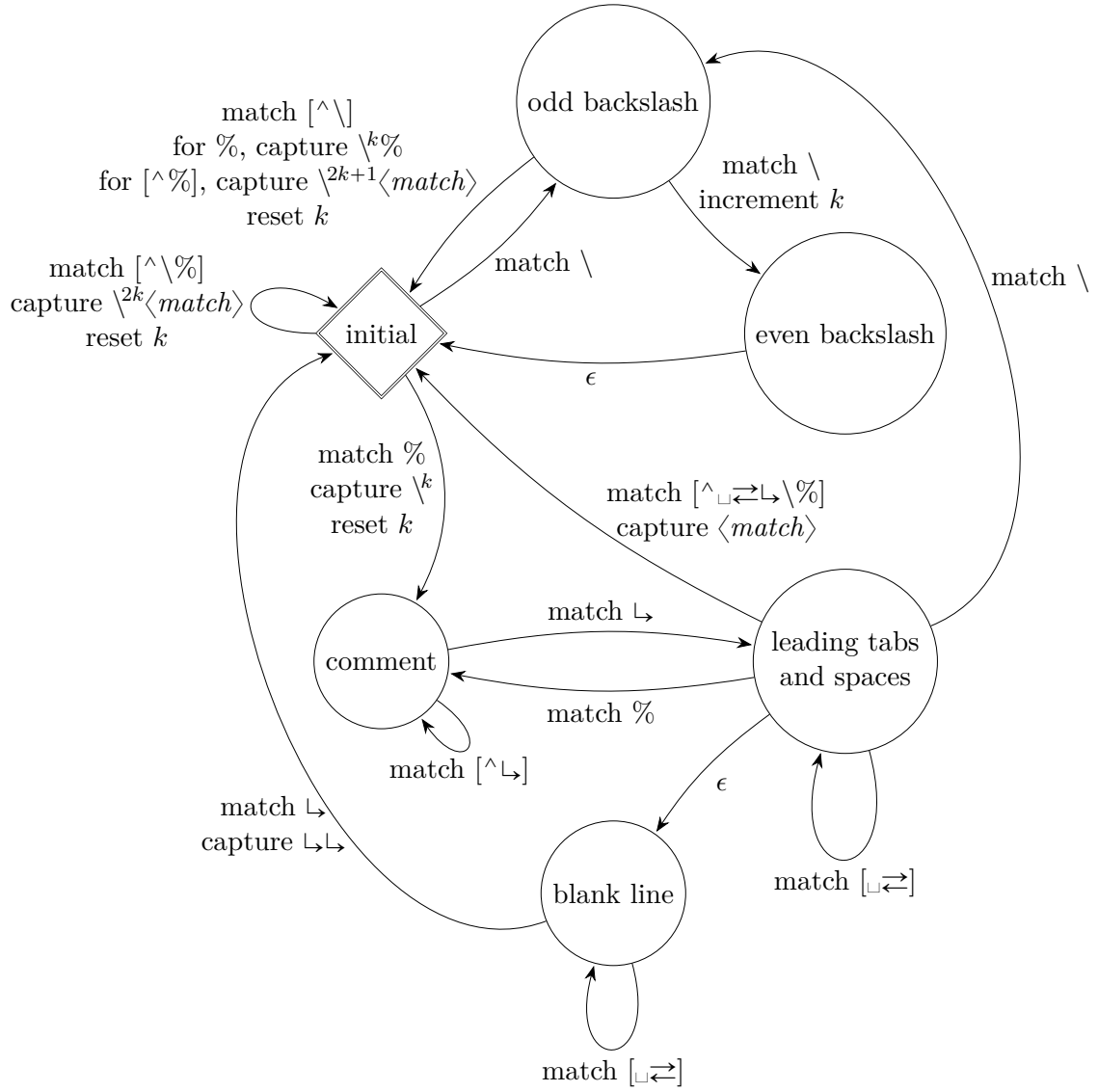
```

The `parsers.commented_line^1` parser recognizes the regular language of T<sub>E</sub>X comments, see an equivalent finite automaton in Figure 8.

```

7944 parsers.commented_line_letter = parsers.linechar
7945                               + parsers.newline
7946                               - parsers.backslash
7947                               - parsers.percent
7948 parsers.commented_line = Cg(Cc(""), "backslashes")
7949                       * ((#(parsers.commented_line_letter
7950                           - parsers.newline)
7951                           * Cb("backslashes")
7952                           * Cs(parsers.commented_line_letter
7953                               - parsers.newline)^1 -- initial
7954                           * Cg(Cc(""), "backslashes"))
7955                       + #( parsers.backslash
7956                           * (parsers.backslash + parsers.newline))
7957                       * Cg((parsers.backslash -- even backslash
7958                           * ( parsers.backslash
7959                               + #parsers.newline))^1, "backslashes")
7960                       + (parsers.backslash
7961                           * (#parsers.percent
7962                           * Cb("backslashes")
7963                           / function(backslashes)
7964                               return string.rep("\\", #backslashes / 2)
7965                           end
7966                           * C(parsers.percent)
7967                           + #parsers.commented_line_letter
7968                           * Cb("backslashes")
7969                           * Cc("\\")
7970                           * C(parsers.commented_line_letter))
7971                           * Cg(Cc(""), "backslashes"))^0
7972                       * (#parsers.percent
7973                           * Cb("backslashes")
7974                           / function(backslashes)
7975                               return string.rep("\\", #backslashes / 2)
7976                           end
7977                       * ((parsers.percent -- comment
7978                           * parsers.line
7979                           * #parsers.blankline) -- blank line
7980                           / "\\n"
7981                           + parsers.percent -- comment

```



**Figure 8: A pushdown automaton that recognizes TeX comments**

```

7982         * parsers.line
7983         * parsers.optionalspace) -- leading spaces
7984     + #(parsers.newline)
7985     * Cb("backslashes")
7986     * C(parsers.newline))
7987
7988 parsers.chunk = parsers.line * (parsers.optionallyindentedline
7989                               - parsers.blankline)^0
7990
7991 parsers.attribute_key_char = parsers.alphanumeric + S("-_:.")
7992 parsers.attribute_raw_char = parsers.alphanumeric + S("-_")
7993 parsers.attribute_key = (parsers.attribute_key_char
7994                        - parsers.dash - parsers.digit)
7995                        * parsers.attribute_key_char^0
7996 parsers.attribute_value = ( (parsers.dquote / "\"")
7997                          * (parsers.anyescaped - parsers.dquote)^0
7998                          * (parsers.dquote / "\""))
7999                          + ( (parsers.squote / "\"")
8000                          * (parsers.anyescaped - parsers.squote)^0
8001                          * (parsers.squote / "\""))
8002                          + ( parsers.anyescaped
8003                          - parsers.dquote
8004                          - parsers.rbrace
8005                          - parsers.space)^0
8006 parsers.attribute_identifier = parsers.attribute_key_char^1
8007 parsers.attribute_classname = parsers.letter
8008                             * parsers.attribute_key_char^0
8009 parsers.attribute_raw = parsers.attribute_raw_char^1
8010
8011 parsers.attribute = (parsers.dash * Cc(".unnumbered"))
8012                   + C( parsers.hash
8013                     * parsers.attribute_identifier)
8014                   + C( parsers.period
8015                     * parsers.attribute_classname)
8016                   + Cs( parsers.attribute_key
8017                     * parsers.optionalspace
8018                     * parsers.equal
8019                     * parsers.optionalspace
8020                     * parsers.attribute_value)
8021 parsers.attributes = parsers.lbrace
8022                   * parsers.optionalspace
8023                   * parsers.attribute
8024                   * (parsers.spacechar^1
8025                   * parsers.attribute)^0
8026                   * parsers.optionalspace
8027                   * parsers.rbrace
8028

```

```

8029 parsers.raw_attribute = parsers.lbrace
8030             * parsers.optionalspace
8031             * parsers.equal
8032             * C(parsers.attribute_raw)
8033             * parsers.optionalspace
8034             * parsers.rbrace
8035
8036 -- block followed by 0 or more optionally
8037 -- indented blocks with first line indented.
8038 parsers.indented_blocks = function(bl)
8039   return Cs( bl
8040             * ( parsers.blankline^1
8041             * parsers.indent
8042             * -parsers.blankline
8043             * bl)^0
8044             * (parsers.blankline^1 + parsers.eof) )
8045 end

```

### 3.1.5.2 Parsers Used for HTML Entities

```

8046 local function repeat_between(pattern, min, max)
8047   return -pattern^(max + 1) * pattern^min
8048 end
8049
8050 parsers.hexentity = parsers.ampersand * parsers.hash * C(S("Xx"))
8051             * C(repeat_between(parsers.hexdigit, 1, 6))
8052             * parsers.semicolon
8053 parsers.decentity = parsers.ampersand * parsers.hash
8054             * C(repeat_between(parsers.digit, 1, 7))
8055             * parsers.semicolon
8056 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
8057             * parsers.semicolon
8058
8059 parsers.html_entities
8060   = parsers.hexentity / entities.hex_entity_with_x_char
8061   + parsers.decentity / entities.dec_entity
8062   + parsers.tagentity / entities.char_entity

```

### 3.1.5.3 Parsers Used for Markdown Lists

```

8063 parsers.bullet = function(bullet_char, interrupting)
8064   local allowed_end
8065   if interrupting then
8066     allowed_end = C(parsers.spacechar^1) * #parsers.linechar
8067   else
8068     allowed_end = C(parsers.spacechar^1)
8069                 + #(parsers.newline + parsers.eof)
8070   end

```

```

8071 return parsers.check_trail
8072     * Ct(C(bullet_char) * Cc(""))
8073     * allowed_end
8074 end
8075
8076 local function tickbox(interior)
8077     return parsers.optionalspace * parsers.lbracket
8078         * interior * parsers.rbracket * parsers.spacechar^1
8079 end
8080
8081 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
8082 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
8083 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
8084

```

#### 3.1.5.4 Parsers Used for Markdown Code Spans

```

8085 parsers.openticks = Cg(parsers.backtick^1, "ticks")
8086
8087 local function captures_equal_length(_,i,a,b)
8088     return #a == #b and i
8089 end
8090
8091 parsers.closeticks = Cmt(C(parsers.backtick^1)
8092     * Cb("ticks"), captures_equal_length)
8093
8094 parsers.intickschar = (parsers.any - S("\n\r`"))
8095     + V("NoSoftLineBreakEndline")
8096     + (parsers.backtick^1 - parsers.closeticks)
8097
8098 local function process_inticks(s)
8099     s = s:gsub("\n", " ")
8100     s = s:gsub("^ (.*) $", "%1")
8101     return s
8102 end
8103
8104 parsers.inticks = parsers.openticks
8105     * C(parsers.space^0)
8106     * parsers.closeticks
8107     + parsers.openticks
8108     * Cs(Cs(parsers.intickschar^0) / process_inticks)
8109     * parsers.closeticks
8110

```

#### 3.1.5.5 Parsers Used for HTML

```

8111 -- case-insensitive match (we assume s is lowercase)
8112 -- must be single byte encoding

```



```

8113 parsers.keyword_exact = function(s)
8114   local parser = P(0)
8115   for i=1,#s do
8116     local c = s:sub(i,i)
8117     local m = c .. upper(c)
8118     parser = parser * S(m)
8119   end
8120   return parser
8121 end
8122
8123 parsers.special_block_keyword =
8124   parsers.keyword_exact("pre") +
8125   parsers.keyword_exact("script") +
8126   parsers.keyword_exact("style") +
8127   parsers.keyword_exact("textarea")
8128
8129 parsers.block_keyword =
8130   parsers.keyword_exact("address") +
8131   parsers.keyword_exact("article") +
8132   parsers.keyword_exact("aside") +
8133   parsers.keyword_exact("base") +
8134   parsers.keyword_exact("basefont") +
8135   parsers.keyword_exact("blockquote") +
8136   parsers.keyword_exact("body") +
8137   parsers.keyword_exact("caption") +
8138   parsers.keyword_exact("center") +
8139   parsers.keyword_exact("col") +
8140   parsers.keyword_exact("colgroup") +
8141   parsers.keyword_exact("dd") +
8142   parsers.keyword_exact("details") +
8143   parsers.keyword_exact("dialog") +
8144   parsers.keyword_exact("dir") +
8145   parsers.keyword_exact("div") +
8146   parsers.keyword_exact("dl") +
8147   parsers.keyword_exact("dt") +
8148   parsers.keyword_exact("fieldset") +
8149   parsers.keyword_exact("figcaption") +
8150   parsers.keyword_exact("figure") +
8151   parsers.keyword_exact("footer") +
8152   parsers.keyword_exact("form") +
8153   parsers.keyword_exact("frame") +
8154   parsers.keyword_exact("frameset") +
8155   parsers.keyword_exact("h1") +
8156   parsers.keyword_exact("h2") +
8157   parsers.keyword_exact("h3") +
8158   parsers.keyword_exact("h4") +
8159   parsers.keyword_exact("h5") +

```

```

8160     parsers.keyword_exact("h6") +
8161     parsers.keyword_exact("head") +
8162     parsers.keyword_exact("header") +
8163     parsers.keyword_exact("hr") +
8164     parsers.keyword_exact("html") +
8165     parsers.keyword_exact("iframe") +
8166     parsers.keyword_exact("legend") +
8167     parsers.keyword_exact("li") +
8168     parsers.keyword_exact("link") +
8169     parsers.keyword_exact("main") +
8170     parsers.keyword_exact("menu") +
8171     parsers.keyword_exact("menuitem") +
8172     parsers.keyword_exact("nav") +
8173     parsers.keyword_exact("noframes") +
8174     parsers.keyword_exact("ol") +
8175     parsers.keyword_exact("optgroup") +
8176     parsers.keyword_exact("option") +
8177     parsers.keyword_exact("p") +
8178     parsers.keyword_exact("param") +
8179     parsers.keyword_exact("section") +
8180     parsers.keyword_exact("source") +
8181     parsers.keyword_exact("summary") +
8182     parsers.keyword_exact("table") +
8183     parsers.keyword_exact("tbody") +
8184     parsers.keyword_exact("td") +
8185     parsers.keyword_exact("tfoot") +
8186     parsers.keyword_exact("th") +
8187     parsers.keyword_exact("thead") +
8188     parsers.keyword_exact("title") +
8189     parsers.keyword_exact("tr") +
8190     parsers.keyword_exact("track") +
8191     parsers.keyword_exact("ul")
8192
8193 -- end conditions
8194 parsers.html_blankline_end_condition
8195 = parsers.linechar^0
8196 * ( parsers.newline
8197     * (parsers.check_minimal_blank_indent_and_any_trail
8198         * #parsers.blankline
8199         + parsers.check_minimal_indent_and_any_trail)
8200     * parsers.linechar^1)^0
8201 * (parsers.newline^-1 / "")
8202
8203 local function remove_trailing_blank_lines(s)
8204     return s:gsub("[\n\r]+%s*$", "")
8205 end
8206

```

```

8207 parsers.html_until_end = function(end_marker)
8208   return Cs(Cs((parsers.newline
8209     * (parsers.check_minimal_blank_indent_and_any_trail
8210     * #parsers.blankline
8211     + parsers.check_minimal_indent_and_any_trail)
8212     + parsers.linechar - end_marker)^0
8213     * parsers.linechar^0 * parsers.newline~1)
8214     / remove_trailing_blank_lines)
8215 end
8216
8217 -- attributes
8218 parsers.html_attribute_spacing = parsers.optionalspace
8219                               * V("NoSoftLineBreakEndline")
8220                               * parsers.optionalspace
8221                               + parsers.spacechar^1
8222
8223 parsers.html_attribute_name = ( parsers.letter
8224                               + parsers.colon
8225                               + parsers.underscore)
8226                               * ( parsers.alphanumeric
8227                               + parsers.colon
8228                               + parsers.underscore
8229                               + parsers.period
8230                               + parsers.dash)^0
8231
8232 parsers.html_attribute_value = parsers.squote
8233                               * (parsers.linechar - parsers.squote)^0
8234                               * parsers.squote
8235                               + parsers.dquote
8236                               * (parsers.linechar - parsers.dquote)^0
8237                               * parsers.dquote
8238                               + ( parsers.any
8239                               - parsers.spacechar
8240                               - parsers.newline
8241                               - parsers.dquote
8242                               - parsers.squote
8243                               - parsers.backtick
8244                               - parsers.equal
8245                               - parsers.less
8246                               - parsers.more)^1
8247
8248 parsers.html_inline_attribute_value = parsers.squote
8249                                     * (V("NoSoftLineBreakEndline")
8250                                     + parsers.any
8251                                     - parsers.blankline^2
8252                                     - parsers.squote)^0
8253                                     * parsers.squote

```

```

8254             + parsers.dquote
8255             * (V("NoSoftLineBreakEndline")
8256             + parsers.any
8257             - parsers.blankline^2
8258             - parsers.dquote)^0
8259             * parsers.dquote
8260             + (parsers.any
8261             - parsers.spacechar
8262             - parsers.newline
8263             - parsers.dquote
8264             - parsers.squote
8265             - parsers.backtick
8266             - parsers.equal
8267             - parsers.less
8268             - parsers.more)^1
8269
8270 parsers.html_attribute_value_specification
8271   = parsers.optionalspace
8272   * parsers.equal
8273   * parsers.optionalspace
8274   * parsers.html_attribute_value
8275
8276 parsers.html_spnl = parsers.optionalspace
8277                   * (V("NoSoftLineBreakEndline")
8278                   * parsers.optionalspace)^-1
8279
8280 parsers.html_inline_attribute_value_specification
8281   = parsers.html_spnl
8282   * parsers.equal
8283   * parsers.html_spnl
8284   * parsers.html_inline_attribute_value
8285
8286 parsers.html_attribute
8287   = parsers.html_attribute_spacing
8288   * parsers.html_attribute_name
8289   * parsers.html_inline_attribute_value_specification^-1
8290
8291 parsers.html_non_newline_attribute
8292   = parsers.spacechar^1
8293   * parsers.html_attribute_name
8294   * parsers.html_attribute_value_specification^-1
8295
8296 parsers.nested_breaking_blank = parsers.newline
8297                               * parsers.check_minimal_blank_indent
8298                               * parsers.blankline
8299
8300 parsers.html_comment_start = P("<!--")

```

```

8301
8302 parsers.html_comment_end = P("-->")
8303
8304 parsers.html_comment
8305     = Cs( parsers.html_comment_start
8306           * parsers.html_until_end(parsers.html_comment_end))
8307
8308 parsers.html_inline_comment = (parsers.html_comment_start / "")
8309                               * -P(">") * -P("->")
8310                               * Cs(( V("NoSoftLineBreakEndline")
8311                                     + parsers.any
8312                                     - parsers.nested_breaking_blank
8313                                     - parsers.html_comment_end)^0)
8314                               * (parsers.html_comment_end / "")
8315
8316 parsers.html_cdatasection_start = P("<![CDATA[")
8317
8318 parsers.html_cdatasection_end = P("]]>")
8319
8320 parsers.html_cdatasection
8321     = Cs( parsers.html_cdatasection_start
8322           * parsers.html_until_end(parsers.html_cdatasection_end))
8323
8324 parsers.html_inline_cdatasection
8325     = parsers.html_cdatasection_start
8326     * Cs(V("NoSoftLineBreakEndline") + parsers.any
8327         - parsers.nested_breaking_blank - parsers.html_cdatasection_end)^0
8328     * parsers.html_cdatasection_end
8329
8330 parsers.html_declaration_start = P("<!") * parsers.letter
8331
8332 parsers.html_declaration_end = P(">")
8333
8334 parsers.html_declaration
8335     = Cs( parsers.html_declaration_start
8336           * parsers.html_until_end(parsers.html_declaration_end))
8337
8338 parsers.html_inline_declaration
8339     = parsers.html_declaration_start
8340     * Cs(V("NoSoftLineBreakEndline") + parsers.any
8341         - parsers.nested_breaking_blank - parsers.html_declaration_end)^0
8342     * parsers.html_declaration_end
8343
8344 parsers.html_instruction_start = P("<?")
8345
8346 parsers.html_instruction_end = P("?>")
8347

```

```

8348 parsers.html_instruction
8349     = Cs( parsers.html_instruction_start
8350           * parsers.html_until_end(parsers.html_instruction_end))
8351
8352 parsers.html_inline_instruction = parsers.html_instruction_start
8353                               * Cs( V("NoSoftLineBreakEndline")
8354                                     + parsers.any
8355                                     - parsers.nested_breaking_blank
8356                                     - parsers.html_instruction_end)^0
8357                               * parsers.html_instruction_end
8358
8359 parsers.html_blankline = parsers.newline
8360                       * parsers.optionalspace
8361                       * parsers.newline
8362
8363 parsers.html_tag_start = parsers.less
8364
8365 parsers.html_tag_closing_start = parsers.less
8366                               * parsers.slash
8367
8368 parsers.html_tag_end = parsers.html_spnl
8369                     * parsers.more
8370
8371 parsers.html_empty_tag_end = parsers.html_spnl
8372                           * parsers.slash
8373                           * parsers.more
8374
8375 -- opening tags
8376 parsers.html_any_open_inline_tag = parsers.html_tag_start
8377                                 * parsers.keyword
8378                                 * parsers.html_attribute^0
8379                                 * parsers.html_tag_end
8380
8381 parsers.html_any_open_tag = parsers.html_tag_start
8382                           * parsers.keyword
8383                           * parsers.html_non_newline_attribute^0
8384                           * parsers.html_tag_end
8385
8386 parsers.html_open_tag = parsers.html_tag_start
8387                       * parsers.block_keyword
8388                       * parsers.html_attribute^0
8389                       * parsers.html_tag_end
8390
8391 parsers.html_open_special_tag = parsers.html_tag_start
8392                              * parsers.special_block_keyword
8393                              * parsers.html_attribute^0
8394                              * parsers.html_tag_end

```

```

8395
8396 -- incomplete tags
8397 parsers.incomplete_tag_following = parsers.spacechar
8398                                + parsers.more
8399                                + parsers.slash * parsers.more
8400                                + #(parsers.newline + parsers.eof)
8401
8402 parsers.incomplete_special_tag_following = parsers.spacechar
8403                                         + parsers.more
8404                                         + #( parsers.newline
8405                                           + parsers.eof)
8406
8407 parsers.html_incomplete_open_tag = parsers.html_tag_start
8408                                * parsers.block_keyword
8409                                * parsers.incomplete_tag_following
8410
8411 parsers.html_incomplete_open_special_tag
8412 = parsers.html_tag_start
8413   * parsers.special_block_keyword
8414   * parsers.incomplete_special_tag_following
8415
8416 parsers.html_incomplete_close_tag = parsers.html_tag_closing_start
8417                                * parsers.block_keyword
8418                                * parsers.incomplete_tag_following
8419
8420 parsers.html_incomplete_close_special_tag
8421 = parsers.html_tag_closing_start
8422   * parsers.special_block_keyword
8423   * parsers.incomplete_tag_following
8424
8425 -- closing tags
8426 parsers.html_close_tag = parsers.html_tag_closing_start
8427                       * parsers.block_keyword
8428                       * parsers.html_tag_end
8429
8430 parsers.html_any_close_tag = parsers.html_tag_closing_start
8431                           * parsers.keyword
8432                           * parsers.html_tag_end
8433
8434 parsers.html_close_special_tag = parsers.html_tag_closing_start
8435                               * parsers.special_block_keyword
8436                               * parsers.html_tag_end
8437
8438 -- empty tags
8439 parsers.html_any_empty_inline_tag = parsers.html_tag_start
8440                                * parsers.keyword
8441                                * parsers.html_attribute^0

```

```

8442                                     * parsers.html_empty_tag_end
8443
8444 parsers.html_any_empty_tag = parsers.html_tag_start
8445                             * parsers.keyword
8446                             * parsers.html_non_newline_attribute~0
8447                             * parsers.optionalspace
8448                             * parsers.slash
8449                             * parsers.more
8450
8451 parsers.html_empty_tag = parsers.html_tag_start
8452                         * parsers.block_keyword
8453                         * parsers.html_attribute~0
8454                         * parsers.html_empty_tag_end
8455
8456 parsers.html_empty_special_tag = parsers.html_tag_start
8457                                 * parsers.special_block_keyword
8458                                 * parsers.html_attribute~0
8459                                 * parsers.html_empty_tag_end
8460
8461 parsers.html_incomplete_blocks
8462   = parsers.html_incomplete_open_tag
8463   + parsers.html_incomplete_open_special_tag
8464   + parsers.html_incomplete_close_tag
8465
8466 -- parse special html blocks
8467 parsers.html_blankline_ending_special_block_opening
8468   = ( parsers.html_close_special_tag
8469       + parsers.html_empty_special_tag)
8470   * #( parsers.optionalspace
8471         * (parsers.newline + parsers.eof))
8472
8473 parsers.html_blankline_ending_special_block
8474   = parsers.html_blankline_ending_special_block_opening
8475   * parsers.html_blankline_end_condition
8476
8477 parsers.html_special_block_opening
8478   = parsers.html_incomplete_open_special_tag
8479   - parsers.html_empty_special_tag
8480
8481 parsers.html_closing_special_block
8482   = parsers.html_special_block_opening
8483   * parsers.html_until_end(parsers.html_close_special_tag)
8484
8485 parsers.html_special_block
8486   = parsers.html_blankline_ending_special_block
8487   + parsers.html_closing_special_block
8488

```



```

8489 -- parse html blocks
8490 parsers.html_block_opening = parsers.html_incomplete_open_tag
8491                             + parsers.html_incomplete_close_tag
8492
8493 parsers.html_block = parsers.html_block_opening
8494                     * parsers.html_blankline_end_condition
8495
8496 -- parse any html blocks
8497 parsers.html_any_block_opening
8498 = ( parsers.html_any_open_tag
8499   + parsers.html_any_close_tag
8500   + parsers.html_any_empty_tag)
8501 * #(parsers.optionalspace * (parsers.newline + parsers.eof))
8502
8503 parsers.html_any_block = parsers.html_any_block_opening
8504                         * parsers.html_blankline_end_condition
8505
8506 parsers.html_inline_comment_full = parsers.html_comment_start
8507                                   * -P(">") * -P("->")
8508                                   * Cs(( V("NoSoftLineBreakEndline")
8509                                         + parsers.any - P("--")
8510                                         - parsers.nested_breaking_blank
8511                                         - parsers.html_comment_end)^0)
8512                                   * parsers.html_comment_end
8513
8514 parsers.html_inline_tags = parsers.html_inline_comment_full
8515                           + parsers.html_any_empty_inline_tag
8516                           + parsers.html_inline_instruction
8517                           + parsers.html_inline_cdatasection
8518                           + parsers.html_inline_declaration
8519                           + parsers.html_any_open_inline_tag
8520                           + parsers.html_any_close_tag
8521

```

### 3.1.5.6 Parsers Used for Markdown Tags and Links

```

8522 parsers.urlchar = parsers.anyescaped
8523                 - parsers.newline
8524                 - parsers.more
8525
8526 parsers.auto_link_scheme_part = parsers.alphanumeric
8527                               + parsers.plus
8528                               + parsers.period
8529                               + parsers.dash
8530
8531 parsers.auto_link_scheme = parsers.letter
8532                           * parsers.auto_link_scheme_part

```

```

8533             * parsers.auto_link_scheme_part^-30
8534
8535 parsers.absolute_uri = parsers.auto_link_scheme * parsers.colon
8536             * ( parsers.any - parsers.spacing
8537             - parsers.less - parsers.more)^0
8538
8539 parsers.printable_characters = S("!.#$%&'*/+=?^_`{|}~-")
8540
8541 parsers.email_address_local_part_char = parsers.alphanumeric
8542             + parsers.printable_characters
8543
8544 parsers.email_address_local_part
8545     = parsers.email_address_local_part_char^1
8546
8547 parsers.email_address_dns_label = parsers.alphanumeric
8548             * ( parsers.alphanumeric
8549             + parsers.dash)^-62
8550             * B(parsers.alphanumeric)
8551
8552 parsers.email_address_domain = parsers.email_address_dns_label
8553             * ( parsers.period
8554             * parsers.email_address_dns_label)^0
8555
8556 parsers.email_address = parsers.email_address_local_part
8557             * parsers.at
8558             * parsers.email_address_domain
8559
8560 parsers.auto_link_url = parsers.less
8561             * C(parsers.absolute_uri)
8562             * parsers.more
8563
8564 parsers.auto_link_email = parsers.less
8565             * C(parsers.email_address)
8566             * parsers.more
8567
8568 parsers.auto_link_relative_reference = parsers.less
8569             * C(parsers.urlchar^1)
8570             * parsers.more
8571
8572 parsers.autolink = parsers.auto_link_url
8573             + parsers.auto_link_email
8574
8575 -- content in balanced brackets, parentheses, or quotes:
8576 parsers.bracketed = P{ parsers.lbracket
8577             * (( parsers.backslash / '"' * parsers.rbracket
8578             + parsers.any - (parsers.lbracket
8579             + parsers.rbracket

```

```

8580                                     + parsers.blankline^2)
8581                                 ) + V(1))^0
8582                             * parsers.rbracket }
8583
8584 parsers.inparens = P{ parsers.lparent
8585                       * ((parsers.anyescaped - (parsers.lparent
8586                                                         + parsers.rparent
8587                                                         + parsers.blankline^2)
8588                       ) + V(1))^0
8589                       * parsers.rparent }
8590
8591 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
8592                       * ((parsers.anyescaped - (parsers.squote
8593                                                         + parsers.blankline^2)
8594                       ) + V(1))^0
8595                       * parsers.squote }
8596
8597 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
8598                       * ((parsers.anyescaped - (parsers.dquote
8599                                                         + parsers.blankline^2)
8600                       ) + V(1))^0
8601                       * parsers.dquote }
8602
8603 parsers.link_text = parsers.lbracket
8604                   * Cs((parsers.alphanumeric^1
8605                       + parsers.bracketed
8606                       + parsers.inticks
8607                       + parsers.autolink
8608                       + V("InlineHtml")
8609                       + ( parsers.backslash * parsers.backslash)
8610                       + ( parsers.backslash
8611                           * ( parsers.lbracket
8612                               + parsers.rbracket)
8613                       + V("NoSoftLineBreakSpace")
8614                       + V("NoSoftLineBreakEndline")
8615                       + (parsers.any
8616                           - ( parsers.newline
8617                               + parsers.lbracket
8618                               + parsers.rbracket
8619                               + parsers.blankline^2))))^0)
8620                   * parsers.rbracket
8621
8622 parsers.link_label_body = -#(parsers.sp * parsers.rbracket)
8623                       * #( ( parsers.any
8624                           - parsers.rbracket)^-999
8625                       * parsers.rbracket)
8626                       * Cs((parsers.alphanumeric^1

```

```

8627         + parsers.inticks
8628         + parsers.autolink
8629         + V("InlineHtml")
8630         + ( parsers.backslash * parsers.backslash)
8631         + ( parsers.backslash
8632           * ( parsers.lbracket
8633             + parsers.rbracket)
8634           + V("NoSoftLineBreakSpace")
8635           + V("NoSoftLineBreakEndline")
8636           + (parsers.any
8637             - ( parsers.newline
8638               + parsers.lbracket
8639               + parsers.rbracket
8640               + parsers.blankline^2))))^1)
8641
8642 parsers.link_label = parsers.lbracket
8643                   * parsers.link_label_body
8644                   * parsers.rbracket
8645
8646 parsers.inparens_url = P{ parsers.lparent
8647                          * ((parsers.anyescaped - (parsers.lparent
8648                                                         + parsers.rparent
8649                                                         + parsers.spacing)
8650                             ) + V(1))^0
8651                          * parsers.rparent }
8652
8653 -- url for markdown links, allowing nested brackets:
8654 parsers.url = parsers.less * Cs((parsers.anyescaped
8655                                  - parsers.newline
8656                                  - parsers.less
8657                                  - parsers.more)^0)
8658                               * parsers.more
8659 + -parsers.less
8660 * Cs((parsers.inparens_url + (parsers.anyescaped
8661                               - parsers.spacing
8662                               - parsers.lparent
8663                               - parsers.rparent))^1)
8664
8665 -- quoted text:
8666 parsers.title_s = parsers.squote
8667                 * Cs((parsers.html_entities
8668                       + V("NoSoftLineBreakSpace")
8669                       + V("NoSoftLineBreakEndline")
8670                       + ( parsers.anyescaped
8671                         - parsers.newline
8672                         - parsers.squote
8673                         - parsers.blankline^2))^0)

```

```

8674             * parsers.squote
8675
8676 parsers.title_d    = parsers.dquote
8677             * Cs((parsers.html_entities
8678                 + V("NoSoftLineBreakSpace")
8679                 + V("NoSoftLineBreakEndline")
8680                 + ( parsers.anyescaped
8681                   - parsers.newline
8682                   - parsers.dquote
8683                   - parsers.blankline^2))^0)
8684             * parsers.dquote
8685
8686 parsers.title_p    = parsers.lparent
8687             * Cs((parsers.html_entities
8688                 + V("NoSoftLineBreakSpace")
8689                 + V("NoSoftLineBreakEndline")
8690                 + ( parsers.anyescaped
8691                   - parsers.newline
8692                   - parsers.lparent
8693                   - parsers.rparent
8694                   - parsers.blankline^2))^0)
8695             * parsers.rparent
8696
8697 parsers.title
8698   = parsers.title_d + parsers.title_s + parsers.title_p
8699
8700 parsers.optionaltitle
8701   = parsers.spnlc * parsers.title * parsers.spacechar^0 + Cc("")
8702

```

### 3.1.5.7 Helpers for Links and Link Reference Definitions

```

8703 -- parse a reference definition: [foo]: /bar "title"
8704 parsers.define_reference_parser = (parsers.check_trail / "")
8705                               * parsers.link_label * parsers.colon
8706                               * parsers.spnlc * parsers.url
8707                               * ( parsers.spnlc_sep * parsers.title
8708                                 * parsers.only_blank
8709                                 + Cc("") * parsers.only_blank)

```

### 3.1.5.8 Inline Elements

```

8710 parsers.Inline      = V("Inline")
8711
8712 -- parse many p between starter and ender
8713 parsers.between = function(p, starter, ender)
8714   local ender2 = B(parsers.nonspacechar) * ender
8715   return ( starter

```

```

8716         * #parsers.nonspacechar
8717         * Ct(p * (p - ender2)^0)
8718         * ender2)
8719 end
8720

```

### 3.1.5.9 Block Elements

```

8721 parsers.lineof = function(c)
8722     return ( parsers.check_trail_no_rem
8723             * (P(c) * parsers.optionalspace)^3
8724             * (parsers.newline + parsers.eof))
8725 end
8726
8727 parsers.thematic_break_lines = parsers.lineof(parsers.asterisk)
8728                               + parsers.lineof(parsers.dash)
8729                               + parsers.lineof(parsers.underscore)

```

### 3.1.5.10 Headings

```

8730 -- parse Atx heading start and return level
8731 parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
8732                       * -parsers.hash / length
8733
8734 -- parse setext header ending and return level
8735 parsers.heading_level
8736   = parsers.nonindentSPACE * parsers.equal^1
8737   * parsers.optionalspace * #parsers.newline * Cc(1)
8738   + parsers.nonindentSPACE * parsers.dash^1
8739   * parsers.optionalspace * #parsers.newline * Cc(2)
8740
8741 local function strip_atx_end(s)
8742     return s:gsub("%s+#+%s*\n$", "")
8743 end
8744
8745 parsers.atx_heading = parsers.check_trail_no_rem
8746                   * Cg(parsers.heading_start, "level")
8747                   * (C( parsers.optionalspace
8748                       * parsers.hash^0
8749                       * parsers.optionalspace
8750                       * parsers.newline)
8751                     + parsers.spacechar^1
8752                     * 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 TeX 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`.

```
8753 M.reader = {}
8754 function M.reader.new(writer, options)
8755     local self = {}
```

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

```
8756     self.writer = writer
8757     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.

```
8758     self.parsers = {}
8759     (function(parsers)
8760         setmetatable(self.parsers, {
8761             __index = function (_, key)
8762                 return parsers[key]
8763             end
8764         })
8765     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.

```
8766     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.

```
8767     function self.normalize_tag(tag)
8768         tag = util.rope_to_string(tag)
8769         tag = tag:gsub("[ \\n\\r\\t]+", " ")
8770         tag = tag:gsub("^ ", ""):gsub(" $", "")
8771         tag = uni_algos.case.casefold(tag, true, false)
8772         return tag
```

```
8773 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.

```
8774 local function iterlines(s, f)
8775     local rope = lpeg.match(Ct((parsers.line / f)^1), s)
8776     return util.rope_to_string(rope)
8777 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.

```
8778 if options.preserveTabs then
8779     self.expandtabs = function(s) return s end
8780 else
8781     self.expandtabs = function(s)
8782         if s:find("\t") then
8783             return iterlines(s, util.expand_tabs_in_line)
8784         else
8785             return s
8786         end
8787     end
8788 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.

```
8789 self.parser_functions = {}
8790 self.create_parser = function(name, grammar, toplevel)
8791     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.

```
8792         if toplevel and options.stripIndent then
8793             local min_prefix_length, min_prefix = nil, ''
8794             str = iterlines(str, function(line)
8795                 if lpeg.match(parsers.nonemptyline, line) == nil then
8796                     return line
8797                 end
8798                 line = util.expand_tabs_in_line(line)
8799                 local prefix = lpeg.match(C(parsers.optionalspace), line)
```



```

8800         local prefix_length = #prefix
8801         local is_shorter = min_prefix_length == nil
8802         if not is_shorter then
8803             is_shorter = prefix_length < min_prefix_length
8804         end
8805         if is_shorter then
8806             min_prefix_length, min_prefix = prefix_length, prefix
8807         end
8808         return line
8809     end)
8810     str = str:gsub('^' .. min_prefix, '')
8811 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.

```

8812     if toplevel and (options.texComments or options.hybrid) then
8813         str = lpeg.match(Ct(parsers.commented_line^1), str)
8814         str = util.rope_to_string(str)
8815     end
8816     local res = lpeg.match(grammar(), str)
8817     if res == nil then
8818         return writer.error(
8819             format("Parser `%s` failed to process the input text.", name),
8820             format("Here are the first 20 characters of the remaining "
8821                 .. "unprocessed text: `%s`.", str:sub(1,20))
8822         )
8823     else
8824         return res
8825     end
8826 end
8827 end
8828
8829 self.create_parser("parse_blocks",
8830     function()
8831         return parsers.blocks
8832     end, true)
8833
8834 self.create_parser("parse_blocks_nested",
8835     function()
8836         return parsers.blocks_nested
8837     end, false)
8838
8839 self.create_parser("parse_inlines",
8840     function()
8841         return parsers.inlines
8842     end, false)

```

```

8843
8844 self.create_parser("parse_inlines_no_inline_note",
8845                     function()
8846                         return parsers.inlines_no_inline_note
8847                     end, false)
8848
8849 self.create_parser("parse_inlines_no_html",
8850                     function()
8851                         return parsers.inlines_no_html
8852                     end, false)
8853
8854 self.create_parser("parse_inlines_nbsp",
8855                     function()
8856                         return parsers.inlines_nbsp
8857                     end, false)
8858 self.create_parser("parse_inlines_no_link_or_emphasis",
8859                     function()
8860                         return parsers.inlines_no_link_or_emphasis
8861                     end, false)

```

### 3.1.6.3 Parsers Used for Indentation (local)

The following patterns represent basic building blocks of indented content.

```

8862 parsers.minimally_indented_blankline
8863     = parsers.check_minimal_indent * (parsers.blankline / "")
8864
8865 parsers.minimally_indented_block
8866     = parsers.check_minimal_indent * V("Block")
8867
8868 parsers.minimally_indented_block_or_paragraph
8869     = parsers.check_minimal_indent * V("BlockOrParagraph")
8870
8871 parsers.minimally_indented_paragraph
8872     = parsers.check_minimal_indent * V("Paragraph")
8873
8874 parsers.minimally_indented_plain
8875     = parsers.check_minimal_indent * V("Plain")
8876
8877 parsers.minimally_indented_par_or_plain
8878     = parsers.minimally_indented_paragraph
8879     + parsers.minimally_indented_plain
8880
8881 parsers.minimally_indented_par_or_plain_no_blank
8882     = parsers.minimally_indented_par_or_plain
8883     - parsers.minimally_indented_blankline
8884
8885 parsers.minimally_indented_ref

```

```

8886     = parsers.check_minimal_indent * V("Reference")
8887
8888 parsers.minimally_indented_blank
8889     = parsers.check_minimal_indent * V("Blank")
8890
8891 parsers.conditionally_indented_blankline
8892     = parsers.check_minimal_blank_indent * (parsers.blankline / "")
8893
8894 parsers.minimally_indented_ref_or_block
8895     = parsers.minimally_indented_ref
8896     + parsers.minimally_indented_block
8897     - parsers.minimally_indented_blankline
8898
8899 parsers.minimally_indented_ref_or_block_or_par
8900     = parsers.minimally_indented_ref
8901     + parsers.minimally_indented_block_or_paragraph
8902     - parsers.minimally_indented_blankline
8903

```

The following pattern parses the properly indented content that follows the initial container start.

```

8904
8905 function parsers.separator_loop(separated_block, paragraph,
8906                                block_separator, paragraph_separator)
8907     return separated_block
8908         + block_separator
8909         * paragraph
8910         * separated_block
8911         + paragraph_separator
8912         * paragraph
8913 end
8914
8915 function parsers.create_loop_body_pair(separated_block, paragraph,
8916                                       block_separator,
8917                                       paragraph_separator)
8918     return {
8919         block = parsers.separator_loop(separated_block, paragraph,
8920                                       block_separator, block_separator),
8921         par = parsers.separator_loop(separated_block, paragraph,
8922                                     block_separator, paragraph_separator)
8923     }
8924 end
8925
8926 parsers.block_sep_group = function(blank)
8927     return blank^0 * parsers.eof
8928         + ( blank^2 / writer.paragraphsep
8929           + blank^0 / writer.interblocksep

```

```

8930         )
8931     end
8932
8933     parsers.par_sep_group = function(blank)
8934         return blank^0 * parsers.eof
8935             + blank^0 / writer.paragraphsep
8936     end
8937
8938     parsers.sep_group_no_output = function(blank)
8939         return blank^0 * parsers.eof
8940             + blank^0
8941     end
8942
8943     parsers.content_blank = parsers.minimally_indented_blankline
8944
8945     parsers.ref_or_block_separated
8946     = parsers.sep_group_no_output(parsers.content_blank)
8947     * ( parsers.minimally_indented_ref
8948         - parsers.content_blank)
8949     + parsers.block_sep_group(parsers.content_blank)
8950     * ( parsers.minimally_indented_block
8951         - parsers.content_blank)
8952
8953     parsers.loop_body_pair =
8954     parsers.create_loop_body_pair(
8955         parsers.ref_or_block_separated,
8956         parsers.minimally_indented_par_or_plain_no_blank,
8957         parsers.block_sep_group(parsers.content_blank),
8958         parsers.par_sep_group(parsers.content_blank))
8959
8960     parsers.content_loop = ( V("Block")
8961                             * parsers.loop_body_pair.block^0
8962                             + (V("Paragraph") + V("Plain")))
8963                             * parsers.ref_or_block_separated
8964                             * parsers.loop_body_pair.block^0
8965                             + (V("Paragraph") + V("Plain")))
8966                             * parsers.loop_body_pair.par^0)
8967     * parsers.content_blank^0
8968
8969     parsers.indented_content = function()
8970         return Ct( (V("Reference") + (parsers.blankline / ""))
8971                 * parsers.content_blank^0
8972                 * parsers.check_minimal_indent
8973                 * parsers.content_loop
8974                 + (V("Reference") + (parsers.blankline / ""))
8975                 * parsers.content_blank^0
8976                 + parsers.content_loop)

```

```

8977 end
8978
8979 parsers.add_indent = function(pattern, name, breakable)
8980     return Cg(Cmt( Cb("indent_info")
8981         * Ct(pattern)
8982         * ( #parsers.linechar -- check if starter is blank
8983             * Cc(false) + Cc(true))
8984         * Cc(name)
8985         * Cc(breakable),
8986         process_starter_indent), "indent_info")
8987 end
8988

```

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

```

8989 if options.hashEnumerators then
8990     parsers.dig = parsers.digit + parsers.hash
8991 else
8992     parsers.dig = parsers.digit
8993 end
8994
8995 parsers.enumerator = function(delimiter_type, interrupting)
8996     local delimiter_range
8997     local allowed_end
8998     if interrupting then
8999         delimiter_range = P("1")
9000         allowed_end = C(parsers.spacechar^1) * #parsers.linechar
9001     else
9002         delimiter_range = parsers.dig * parsers.dig^-8
9003         allowed_end = C(parsers.spacechar^1)
9004             + #(parsers.newline + parsers.eof)
9005     end
9006
9007     return parsers.check_trail
9008         * Ct(C(delimiter_range) * C(delimiter_type))
9009         * allowed_end
9010 end
9011
9012 parsers.starter = parsers.bullet(parsers.dash)
9013     + parsers.bullet(parsers.asterisk)
9014     + parsers.bullet(parsers.plus)
9015     + parsers.enumerator(parsers.period)
9016     + parsers.enumerator(parsers.rparent)
9017

```

#### 3.1.6.5 Parsers Used for Blockquotes (local)

```

9018 parsers.blockquote_start

```

```

9019     = parsers.check_trail
9020     * C(parsers.more)
9021     * C(parsers.spacechar^0)
9022
9023     parsers.blockquote_body
9024     = parsers.add_indent(parsers.blockquote_start, "bq", true)
9025     * parsers.indented_content()
9026     * remove_indent("bq")
9027
9028     if not options.breakableBlockquotes then
9029         parsers.blockquote_body
9030         = parsers.add_indent(parsers.blockquote_start, "bq", false)
9031         * parsers.indented_content()
9032         * remove_indent("bq")
9033     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.

```

9034     local function parse_content_part(content_part)
9035         local rope = util.rope_to_string(content_part)
9036         local parsed
9037         = self.parser_functions.parse_inlines_no_link_or_emphasis(rope)
9038         parsed.indent_info = nil
9039         return parsed
9040     end
9041

```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

9042     local collect_emphasis_content =
9043         function(t, opening_index, closing_index)
9044             local content = {}
9045
9046             local content_part = {}
9047             for i = opening_index, closing_index do
9048                 local value = t[i]
9049
9050                 if value.rendered ~= nil then
9051                     content[#content + 1] = parse_content_part(content_part)
9052                     content_part = {}
9053                     content[#content + 1] = value.rendered
9054                     value.rendered = nil
9055                 else
9056                     if value.type == "delimiter"
9057                         and value.element == "emphasis" then
9058                         if value.is_active then

```

```

9059         content_part[#content_part + 1]
9060         = string.rep(value.character, value.current_count)
9061     end
9062 else
9063     content_part[#content_part + 1] = value.content
9064 end
9065 value.content = ''
9066 value.is_active = false
9067 end
9068 end
9069
9070 if next(content_part) ~= nil then
9071     content[#content + 1] = parse_content_part(content_part)
9072 end
9073
9074 return content
9075 end
9076

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as emphasis.

```

9077 local function fill_emph(t, opening_index, closing_index)
9078     local content
9079     = collect_emphasis_content(t, opening_index + 1,
9080                               closing_index - 1)
9081     t[opening_index + 1].is_active = true
9082     t[opening_index + 1].rendered = writer.emphasis(content)
9083 end
9084

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as strong emphasis.

```

9085 local function fill_strong(t, opening_index, closing_index)
9086     local content
9087     = collect_emphasis_content(t, opening_index + 1,
9088                               closing_index - 1)
9089     t[opening_index + 1].is_active = true
9090     t[opening_index + 1].rendered = writer.strong(content)
9091 end
9092

```

Check whether the opening delimiter `opening_delimiter` and closing delimiter `closing_delimiter` break rule three together.

```

9093 local function breaks_three_rule(opening_delimiter, closing_delimiter)
9094     return ( opening_delimiter.is_closing
9095             or closing_delimiter.is_opening)
9096     and (( opening_delimiter.original_count
9097            + closing_delimiter.original_count) % 3 == 0)

```

```

9098         and ( opening_delimiter.original_count % 3 ~= 0
9099             or closing_delimiter.original_count % 3 ~= 0)
9100     end
9101

```

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`.

```

9102     local find_emphasis_opener = function(t, bottom_index, latest_index,
9103                                         character, closing_delimiter)
9104         for i = latest_index, bottom_index, -1 do
9105             local value = t[i]
9106             if value.is_active and
9107                 value.is_opening and
9108                 value.type == "delimiter" and
9109                 value.element == "emphasis" and
9110                 (value.character == character) and
9111                 (value.current_count > 0) then
9112                 if not breaks_three_rule(value, closing_delimiter) then
9113                     return i
9114                 end
9115             end
9116         end
9117     end
9118

```

Iterate over the delimiters in the delimiter table `t`, producing emphasis or strong emphasis macros.

```

9119     local function process_emphasis(t, opening_index, closing_index)
9120         for i = opening_index, closing_index do
9121             local value = t[i]
9122             if value.type == "delimiter" and value.element == "emphasis" then
9123                 local delimiter_length = string.len(value.content)
9124                 value.character = string.sub(value.content, 1, 1)
9125                 value.current_count = delimiter_length
9126                 value.original_count = delimiter_length
9127             end
9128         end
9129
9130         local openers_bottom = {
9131             ['*'] = {
9132                 [true] = {opening_index, opening_index, opening_index},
9133                 [false] = {opening_index, opening_index, opening_index}
9134             },
9135             ['_'] = {
9136                 [true] = {opening_index, opening_index, opening_index},
9137                 [false] = {opening_index, opening_index, opening_index}
9138             }
9139         }
9140

```



```

9139     }
9140
9141     local current_position = opening_index
9142     local max_position = closing_index
9143
9144     while current_position <= max_position do
9145         local value = t[current_position]
9146
9147         if value.type ~= "delimiter" or
9148            value.element ~= "emphasis" or
9149            not value.is_active or
9150            not value.is_closing or
9151            (value.current_count <= 0) then
9152             current_position = current_position + 1
9153             goto continue
9154         end
9155
9156         local character = value.character
9157         local is_opening = value.is_opening
9158         local closing_length_modulo_three = value.original_count % 3
9159
9160         local current_openers_bottom
9161             = openers_bottom[character][is_opening]
9162               [closing_length_modulo_three + 1]
9163
9164         local opener_position
9165             = find_emphasis_opener(t, current_openers_bottom,
9166                                   current_position - 1, character, value)
9167
9168         if (opener_position == nil) then
9169             openers_bottom[character][is_opening]
9170                 [closing_length_modulo_three + 1]
9171                 = current_position
9172             current_position = current_position + 1
9173             goto continue
9174         end
9175
9176         local opening_delimiter = t[opener_position]
9177
9178         local current_opening_count = opening_delimiter.current_count
9179         local current_closing_count = t[current_position].current_count
9180
9181         if (current_opening_count >= 2)
9182            and (current_closing_count >= 2) then
9183             opening_delimiter.current_count = current_opening_count - 2
9184             t[current_position].current_count = current_closing_count - 2
9185             fill_strong(t, opener_position, current_position)

```

```

9186         else
9187             opening_delimiter.current_count = current_opening_count - 1
9188             t[current_position].current_count = current_closing_count - 1
9189             fill_emph(t, opener_position, current_position)
9190         end
9191
9192         ::continue::
9193     end
9194 end
9195
9196 parsers.delimiter_run = function(character)
9197     return (B(parsers.backslash * character) + -B(character))
9198           * character^1
9199           * -#character
9200 end
9201
9202 parsers.left_flanking_delimiter_run = function(character)
9203     return (B( parsers.any)
9204           * ( parsers.unicode.preceding_punctuation
9205             + parsers.unicode.preceding_whitespace)
9206           + -B(parsers.any))
9207           * parsers.delimiter_run(character)
9208           * parsers.unicode.following_punctuation
9209           + parsers.delimiter_run(character)
9210           * -( parsers.unicode.following_punctuation
9211             + parsers.unicode.following_whitespace
9212             + parsers.eof)
9213 end
9214
9215 parsers.right_flanking_delimiter_run = function(character)
9216     return parsers.unicode.preceding_punctuation
9217           * parsers.delimiter_run(character)
9218           * ( parsers.unicode.following_punctuation
9219             + parsers.unicode.following_whitespace
9220             + parsers.eof)
9221           + (B(parsers.any)
9222             * -( parsers.unicode.preceding_punctuation
9223               + parsers.unicode.preceding_whitespace))
9224           * parsers.delimiter_run(character)
9225 end
9226
9227 if options.underscores then
9228     parsers.emph_start
9229         = parsers.left_flanking_delimiter_run(parsers.asterisk)
9230         + ( -#parsers.right_flanking_delimiter_run(parsers.underscore)
9231           + ( parsers.unicode.preceding_punctuation
9232             * #parsers.right_flanking_delimiter_run(parsers.underscore)))

```

```

9233     * parsers.left_flanking_delimiter_run(parsers.underscore)
9234
9235     parsers.emph_end
9236     = parsers.right_flanking_delimiter_run(parsers.asterisk)
9237     + ( -#parsers.left_flanking_delimiter_run(parsers.underscore)
9238         + #( parsers.left_flanking_delimiter_run(parsers.underscore)
9239             * parsers.unicode.following_punctuation))
9240     * parsers.right_flanking_delimiter_run(parsers.underscore)
9241 else
9242     parsers.emph_start
9243     = parsers.left_flanking_delimiter_run(parsers.asterisk)
9244
9245     parsers.emph_end
9246     = parsers.right_flanking_delimiter_run(parsers.asterisk)
9247 end
9248
9249 parsers.emph_capturing_open_and_close
9250 = #parsers.emph_start * #parsers.emph_end
9251 * Ct( Cg(Cc("delimiter"), "type")
9252     * Cg(Cc("emphasis"), "element")
9253     * Cg(C(parsers.emph_start), "content")
9254     * Cg(Cc(true), "is_opening")
9255     * Cg(Cc(true), "is_closing"))
9256
9257 parsers.emph_capturing_open = Ct( Cg(Cc("delimiter"), "type")
9258     * Cg(Cc("emphasis"), "element")
9259     * Cg(C(parsers.emph_start), "content")
9260     * Cg(Cc(true), "is_opening")
9261     * Cg(Cc(false), "is_closing"))
9262
9263 parsers.emph_capturing_close = Ct( Cg(Cc("delimiter"), "type")
9264     * Cg(Cc("emphasis"), "element")
9265     * Cg(C(parsers.emph_end), "content")
9266     * Cg(Cc(false), "is_opening")
9267     * Cg(Cc(true), "is_closing"))
9268
9269 parsers.emph_open_or_close = parsers.emph_capturing_open_and_close
9270 + parsers.emph_capturing_open
9271 + parsers.emph_capturing_close
9272
9273 parsers.emph_open = parsers.emph_capturing_open_and_close
9274 + parsers.emph_capturing_open
9275
9276 parsers.emph_close = parsers.emph_capturing_open_and_close
9277 + parsers.emph_capturing_close
9278

```

### 3.1.6.7 Helpers for Links and Link Reference Definitions (local)

```
9279  -- List of references defined in the document
9280  local references
9281
9282  -- List of note references defined in the document
9283  parsers.rawnotes = {}
9284
```

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.

```
9285  function self.register_link(_, tag, url, title,
9286                               attributes)
9287      local normalized_tag = self.normalize_tag(tag)
9288      if references[normalized_tag] == nil then
9289          references[normalized_tag] = {
9290              url = url,
9291              title = title,
9292              attributes = attributes
9293          }
9294      end
9295      return ""
9296  end
9297
```

The `reader->lookup_reference` method looks up a reference with link label `tag`.

```
9298  function self.lookup_reference(tag)
9299      return references[self.normalize_tag(tag)]
9300  end
9301
```

The `reader->lookup_note_reference` method looks up a note reference with label `tag`.

```
9302  function self.lookup_note_reference(tag)
9303      return parsers.rawnotes[self.normalize_tag(tag)]
9304  end
9305
9306  parsers.title_s_direct_ref = parsers.squote
9307                               * Cs((parsers.html_entities
9308                                     + ( parsers.anyescaped
9309                                       - parsers.squote
9310                                       - parsers.blankline^2))^0)
9311                               * parsers.squote
9312
9313  parsers.title_d_direct_ref = parsers.dquote
9314                               * Cs((parsers.html_entities
9315                                     + ( parsers.anyescaped
9316                                       - parsers.dquote
```

```

9317             - parsers.blankline^2))^0)
9318         * parsers.dquote
9319
9320 parsers.title_p_direct_ref = parsers.lparent
9321     * Cs((parsers.html_entities
9322         + ( parsers.anyescaped
9323             - parsers.lparent
9324             - parsers.rparent
9325             - parsers.blankline^2))^0)
9326     * parsers.rparent
9327
9328 parsers.title_direct_ref = parsers.title_s_direct_ref
9329     + parsers.title_d_direct_ref
9330     + parsers.title_p_direct_ref
9331
9332 parsers.inline_direct_ref_inside = parsers.lparent * parsers.spnl
9333     * Cg(parsers.url + Cc(""), "url")
9334     * parsers.spnl
9335     * Cg( parsers.title_direct_ref
9336         + Cc(""), "title")
9337     * parsers.spnl * parsers.rparent
9338
9339 parsers.inline_direct_ref = parsers.lparent * parsers.spnlc
9340     * Cg(parsers.url + Cc(""), "url")
9341     * parsers.spnlc
9342     * Cg(parsers.title + Cc(""), "title")
9343     * parsers.spnlc * parsers.rparent
9344
9345 parsers.empty_link = parsers.lbracket
9346     * parsers.rbracket
9347
9348 parsers.inline_link = parsers.link_text
9349     * parsers.inline_direct_ref
9350
9351 parsers.full_link = parsers.link_text
9352     * parsers.link_label
9353
9354 parsers.shortcut_link = parsers.link_label
9355     * -(parsers.empty_link + parsers.link_label)
9356
9357 parsers.collapsed_link = parsers.link_label
9358     * parsers.empty_link
9359
9360 parsers.image_opening = #(parsers.exclamation * parsers.inline_link)
9361     * Cg(Cc("inline"), "link_type")
9362     + #(parsers.exclamation * parsers.full_link)
9363     * Cg(Cc("full"), "link_type")

```

```

9364         + #( parsers.exclamation
9365           * parsers.collapsed_link)
9366         * Cg(Cc("collapsed"), "link_type")
9367         + #(parsers.exclamation * parsers.shortcut_link)
9368         * Cg(Cc("shortcut"), "link_type")
9369         + #(parsers.exclamation * parsers.empty_link)
9370         * Cg(Cc("empty"), "link_type")
9371
9372     parsers.link_opening = #parsers.inline_link
9373       * Cg(Cc("inline"), "link_type")
9374       + #parsers.full_link
9375       * Cg(Cc("full"), "link_type")
9376       + #parsers.collapsed_link
9377       * Cg(Cc("collapsed"), "link_type")
9378       + #parsers.shortcut_link
9379       * Cg(Cc("shortcut"), "link_type")
9380       + #parsers.empty_link
9381       * Cg(Cc("empty_link"), "link_type")
9382       + #parsers.link_text
9383       * Cg(Cc("link_text"), "link_type")
9384
9385     parsers.note_opening = #(parsers.circumflex * parsers.link_text)
9386       * Cg(Cc("note_inline"), "link_type")
9387
9388     parsers.raw_note_opening = #( parsers.lbracket
9389       * parsers.circumflex
9390       * parsers.link_label_body
9391       * parsers.rbracket)
9392       * Cg(Cc("raw_note"), "link_type")
9393
9394     local inline_note_element = Cg(Cc("note"), "element")
9395       * parsers.note_opening
9396       * Cg( parsers.circumflex
9397         * parsers.lbracket, "content")
9398
9399     local image_element = Cg(Cc("image"), "element")
9400       * parsers.image_opening
9401       * Cg( parsers.exclamation
9402         * parsers.lbracket, "content")
9403
9404     local note_element = Cg(Cc("note"), "element")
9405       * parsers.raw_note_opening
9406       * Cg( parsers.lbracket
9407         * parsers.circumflex, "content")
9408
9409     local link_element = Cg(Cc("link"), "element")
9410       * parsers.link_opening

```

```

9411             * Cg(parsers.lbracket, "content")
9412
9413     local opening_elements = parsers.fail
9414
9415     if options.inlineNotes then
9416         opening_elements = opening_elements + inline_note_element
9417     end
9418
9419     opening_elements = opening_elements + image_element
9420
9421     if options.notes then
9422         opening_elements = opening_elements + note_element
9423     end
9424
9425     opening_elements = opening_elements + link_element
9426
9427     parsers.link_image_opening = Ct( Cg(Cc("delimiter"), "type")
9428                                     * Cg(Cc(true), "is_opening")
9429                                     * Cg(Cc(false), "is_closing")
9430                                     * opening_elements)
9431
9432     parsers.link_image_closing = Ct( Cg(Cc("delimiter"), "type")
9433                                     * Cg(Cc("link"), "element")
9434                                     * Cg(Cc(false), "is_opening")
9435                                     * Cg(Cc(true), "is_closing")
9436                                     * ( Cg(Cc(true), "is_direct")
9437                                       * Cg( parsers.rbracket
9438                                           * #parsers.inline_direct_ref,
9439                                           "content")
9440                                       + Cg(Cc(false), "is_direct")
9441                                       * Cg(parsers.rbracket, "content")))
9442
9443     parsers.link_image_open_or_close = parsers.link_image_opening
9444                                     + parsers.link_image_closing
9445
9446     if options.html then
9447         parsers.link_emph_precedence = parsers.inticks
9448                                     + parsers.autolink
9449                                     + parsers.html_inline_tags
9450     else
9451         parsers.link_emph_precedence = parsers.inticks
9452                                     + parsers.autolink
9453     end
9454
9455     parsers.link_and_emph_endline = parsers.newline
9456                                     * ((parsers.check_minimal_indent
9457                                     * -V("EndlineExceptions"))

```

```

9458             + parsers.check_optional_indent
9459             * -V("EndlineExceptions")
9460             * -V("ListStarter")) / "")
9461             * parsers.spacechar^0 / "\n"
9462
9463 parsers.link_and_emph_content
9464   = Ct( Cg(Cc("content"), "type")
9465         * Cg(Cs(( parsers.link_emph_precedence
9466                   + parsers.backslash * parsers.linechar
9467                   + parsers.link_and_emph_endline
9468                   + (parsers.linechar
9469                     - parsers.blankline^2
9470                     - parsers.link_image_open_or_close
9471                     - parsers.emph_open_or_close))^0), "content"))
9472
9473 parsers.link_and_emph_table
9474   = (parsers.link_image_opening + parsers.emph_open)
9475     * parsers.link_and_emph_content
9476     * ((parsers.link_image_open_or_close + parsers.emph_open_or_close)
9477         * parsers.link_and_emph_content)^1
9478

```

Collect the content between the [opening\\_index](#) and [closing\\_index](#) in the delimiter table [t](#).

```

9479 local function collect_link_content(t, opening_index, closing_index)
9480   local content = {}
9481   for i = opening_index, closing_index do
9482     content[#content + 1] = t[i].content
9483   end
9484   return util.rope_to_string(content)
9485 end
9486

```

Look for the closest potential link opener in the delimiter table [t](#) in the range from [bottom\\_index](#) to [latest\\_index](#).

```

9487 local function find_link_opener(t, bottom_index, latest_index)
9488   for i = latest_index, bottom_index, -1 do
9489     local value = t[i]
9490     if value.type == "delimiter" and
9491        value.is_opening and
9492        ( value.element == "link"
9493          or value.element == "image"
9494          or value.element == "note")
9495        and not value.removed then
9496       if value.is_active then
9497         return i
9498       end
9499       value.removed = true

```



```

9500         return nil
9501     end
9502 end
9503 end
9504

```

Find the position of a delimiter that closes a full link after an index `latest_index` in the delimiter table `t`.

```

9505     local function find_next_link_closing_index(t, latest_index)
9506         for i = latest_index, #t do
9507             local value = t[i]
9508             if value.is_closing and
9509                 value.element == "link" and
9510                 not value.removed then
9511                 return i
9512             end
9513         end
9514     end
9515

```

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.

```

9516     local function disable_previous_link_openers(t, opening_index)
9517         if t[opening_index].element == "image" then
9518             return
9519         end
9520
9521         for i = opening_index, 1, -1 do
9522             local value = t[i]
9523             if value.is_active and
9524                 value.type == "delimiter" and
9525                 value.is_opening and
9526                 value.element == "link" then
9527                 value.is_active = false
9528             end
9529         end
9530     end
9531

```

Disable the delimiters between the `opening_index` and `closing_index` in the delimiter table `t` by marking them inactive with the `is_active` property.

```

9532     local function disable_range(t, opening_index, closing_index)
9533         for i = opening_index, closing_index do
9534             local value = t[i]
9535             if value.is_active then
9536                 value.is_active = false
9537                 if value.type == "delimiter" then
9538                     value.removed = true

```

```

9539         end
9540     end
9541 end
9542 end
9543

```

Clear the parsed content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

9544     local delete_parsed_content_in_range =
9545         function(t, opening_index, closing_index)
9546             for i = opening_index, closing_index do
9547                 t[i].rendered = nil
9548             end
9549         end
9550

```

Clear the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

9551     local function empty_content_in_range(t, opening_index, closing_index)
9552         for i = opening_index, closing_index do
9553             t[i].content = ''
9554         end
9555     end
9556

```

Join the attributes from the link reference definition `reference_attributes` with the link's own attributes `own_attributes`.

```

9557     local function join_attributes(reference_attributes, own_attributes)
9558         local merged_attributes = {}
9559         for _, attribute in ipairs(reference_attributes or {}) do
9560             table.insert(merged_attributes, attribute)
9561         end
9562         for _, attribute in ipairs(own_attributes or {}) do
9563             table.insert(merged_attributes, attribute)
9564         end
9565         if next(merged_attributes) == nil then
9566             merged_attributes = nil
9567         end
9568         return merged_attributes
9569     end
9570

```

Parse content between two delimiters in the delimiter table `t`. Produce the respective link and image macros.

```

9571     local render_link_or_image =
9572         function(t, opening_index, closing_index, content_end_index,
9573             reference)
9574         process_emphasis(t, opening_index, content_end_index)

```

```

9575     local mapped = collect_emphasis_content(t, opening_index + 1,
9576                                           content_end_index - 1)
9577
9578     local rendered = {}
9579     if (t[opening_index].element == "link") then
9580         rendered = writer.link(mapped, reference.url,
9581                               reference.title, reference.attributes)
9582     end
9583
9584     if (t[opening_index].element == "image") then
9585         rendered = writer.image(mapped, reference.url, reference.title,
9586                                reference.attributes)
9587     end
9588
9589     if (t[opening_index].element == "note") then
9590         if (t[opening_index].link_type == "note_inline") then
9591             rendered = writer.note(mapped)
9592         end
9593         if (t[opening_index].link_type == "raw_note") then
9594             rendered = writer.note(reference)
9595         end
9596     end
9597
9598     t[opening_index].rendered = rendered
9599     delete_parsed_content_in_range(t, opening_index + 1,
9600                                   closing_index)
9601     empty_content_in_range(t, opening_index, closing_index)
9602     disable_previous_link_openers(t, opening_index)
9603     disable_range(t, opening_index, closing_index)
9604 end
9605

```

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.

```

9606     local resolve_inline_following_content =
9607         function(t, closing_index, match_reference, match_link_attributes)
9608             local content = ""
9609             for i = closing_index + 1, #t do
9610                 content = content .. t[i].content
9611             end
9612
9613             local matching_content = parsers.succeed
9614
9615             if match_reference then
9616                 matching_content = matching_content
9617                     * parsers.inline_direct_ref_inside

```

```

9618     end
9619
9620     if match_link_attributes then
9621         matching_content = matching_content
9622             * Cg(Ct(parsers.attributes~-1), "attributes")
9623     end
9624
9625     local matched = lpeg.match(Ct( matching_content
9626                                     * Cg(Cp(), "end_position")), content)
9627
9628     local matched_count = matched.end_position - 1
9629     for i = closing_index + 1, #t do
9630         local value = t[i]
9631
9632         local chars_left = matched_count
9633         matched_count = matched_count - #value.content
9634
9635         if matched_count <= 0 then
9636             value.content = value.content:sub(chars_left + 1)
9637             break
9638         end
9639
9640         value.content = ''
9641         value.is_active = false
9642     end
9643
9644     local attributes = matched.attributes
9645     if attributes == nil or next(attributes) == nil then
9646         attributes = nil
9647     end
9648
9649     return {
9650         url = matched.url or "",
9651         title = matched.title or "",
9652         attributes = attributes
9653     }
9654 end
9655

```

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.

```

9656 local function resolve_inline_link(t, opening_index, closing_index)
9657     local inline_content
9658     = resolve_inline_following_content(t, closing_index, true,
9659                                       t.match_link_attributes)
9660     render_link_or_image(t, opening_index, closing_index,

```

```

9661             closing_index, inline_content)
9662     end
9663

```

Resolve an inline note `^[a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`.

```

9664     local resolve_note_inline_link =
9665         function(t, opening_index, closing_index)
9666             local inline_content
9667                 = resolve_inline_following_content(t, closing_index,
9668                                                     false, false)
9669             render_link_or_image(t, opening_index, closing_index,
9670                                 closing_index, inline_content)
9671         end
9672

```

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.

```

9673     local function resolve_shortcut_link(t, opening_index, closing_index)
9674         local content
9675             = collect_link_content(t, opening_index + 1, closing_index - 1)
9676         local r = self.lookup_reference(content)
9677
9678         if r then
9679             local inline_content
9680                 = resolve_inline_following_content(t, closing_index, false,
9681                                                     t.match_link_attributes)
9682             r.attributes
9683                 = join_attributes(r.attributes, inline_content.attributes)
9684             render_link_or_image(t, opening_index, closing_index,
9685                                 closing_index, r)
9686         end
9687     end
9688

```

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.

```

9689     local function resolve_raw_note_link(t, opening_index, closing_index)
9690         local content
9691             = collect_link_content(t, opening_index + 1, closing_index - 1)
9692         local r = self.lookup_note_reference(content)
9693
9694         if r then
9695             local parsed_ref = self.parser_functions.parse_blocks_nested(r)
9696             render_link_or_image(t, opening_index, closing_index,
9697                                 closing_index, parsed_ref)
9698         end

```

```

9699 end
9700

```

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.

```

9701 local function resolve_full_link(t, opening_index, closing_index)
9702     local next_link_closing_index
9703     = find_next_link_closing_index(t, closing_index + 4)
9704     local next_link_content
9705     = collect_link_content(t, closing_index + 3,
9706                           next_link_closing_index - 1)
9707     local r = self.lookup_reference(next_link_content)
9708
9709     if r then
9710         local inline_content
9711         = resolve_inline_following_content(t, next_link_closing_index,
9712                                           false,
9713                                           t.match_link_attributes)
9714         r.attributes
9715         = join_attributes(r.attributes, inline_content.attributes)
9716         render_link_or_image(t, opening_index, next_link_closing_index,
9717                             closing_index, r)
9718     end
9719 end
9720

```

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.

```

9721 local function resolve_collapsed_link(t, opening_index, closing_index)
9722     local next_link_closing_index
9723     = find_next_link_closing_index(t, closing_index + 4)
9724     local content
9725     = collect_link_content(t, opening_index + 1, closing_index - 1)
9726     local r = self.lookup_reference(content)
9727
9728     if r then
9729         local inline_content
9730         = resolve_inline_following_content(t, closing_index, false,
9731                                           t.match_link_attributes)
9732         r.attributes
9733         = join_attributes(r.attributes, inline_content.attributes)
9734         render_link_or_image(t, opening_index, next_link_closing_index,
9735                             closing_index, r)
9736     end
9737 end
9738

```

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.

```

9739 local function process_links_and_emphasis(t)
9740   for _,value in ipairs(t) do
9741     value.is_active = true
9742   end
9743
9744   for i,value in ipairs(t) do
9745     if not value.is_closing
9746       or value.type ~= "delimiter"
9747       or not ( value.element == "link"
9748         or value.element == "image"
9749         or value.element == "note")
9750       or value.removed then
9751       goto continue
9752     end
9753
9754     local opener_position = find_link_opener(t, 1, i - 1)
9755     if (opener_position == nil) then
9756       goto continue
9757     end
9758
9759     local opening_delimiter = t[opener_position]
9760     opening_delimiter.removed = true
9761
9762     local link_type = opening_delimiter.link_type
9763
9764     if (link_type == "inline") then
9765       resolve_inline_link(t, opener_position, i)
9766     end
9767     if (link_type == "shortcut") then
9768       resolve_shortcut_link(t, opener_position, i)
9769     end
9770     if (link_type == "full") then
9771       resolve_full_link(t, opener_position, i)
9772     end
9773     if (link_type == "collapsed") then
9774       resolve_collapsed_link(t, opener_position, i)
9775     end
9776     if (link_type == "note_inline") then
9777       resolve_note_inline_link(t, opener_position, i)
9778     end
9779     if (link_type == "raw_note") then
9780       resolve_raw_note_link(t, opener_position, i)

```

```

9781         end
9782
9783         ::continue::
9784     end
9785
9786     t[#t].content = t[#t].content:gsub("%s*$","")
9787
9788     process_emphasis(t, 1, #t)
9789     local final_result = collect_emphasis_content(t, 1, #t)
9790     return final_result
9791 end
9792
9793 function self.defer_link_and_emphasis_processing(delimiter_table)
9794     return writer.defer_call(function()
9795         return process_links_and_emphasis(delimiter_table)
9796     end)
9797 end
9798

```

### 3.1.6.8 Inline Elements (local)

```

9799 parsers.Str      = ( parsers.normalchar
9800                     * (parsers.normalchar + parsers.at)^0)
9801                     / writer.string
9802
9803 parsers.Symbol    = (parsers.backtick^1 + V("SpecialChar"))
9804                     / writer.string
9805
9806 parsers.Ellipsis  = P("...") / writer.ellipsis
9807
9808 parsers.Smart     = parsers.Ellipsis
9809
9810 parsers.Code      = parsers.inticks / writer.code
9811
9812 if options.blankBeforeBlockquote then
9813     parsers.bqstart = parsers.fail
9814 else
9815     parsers.bqstart = parsers.blockquote_start
9816 end
9817
9818 if options.blankBeforeHeading then
9819     parsers.headerstart = parsers.fail
9820 else
9821     parsers.headerstart = parsers.atx_heading
9822 end
9823
9824 if options.blankBeforeList then

```



```

9825     parsers.interrupting_bullets = parsers.fail
9826     parsers.interrupting_enumerators = parsers.fail
9827 else
9828     parsers.interrupting_bullets
9829     = parsers.bullet(parsers.dash, true)
9830     + parsers.bullet(parsers.asterisk, true)
9831     + parsers.bullet(parsers.plus, true)
9832
9833     parsers.interrupting_enumerators
9834     = parsers.enumerator(parsers.period, true)
9835     + parsers.enumerator(parsers.rparent, true)
9836 end
9837
9838 if options.html then
9839     parsers.html_interrupting
9840     = parsers.check_trail
9841     * ( parsers.html_incomplete_open_tag
9842         + parsers.html_incomplete_close_tag
9843         + parsers.html_incomplete_open_special_tag
9844         + parsers.html_comment_start
9845         + parsers.html_cdatasection_start
9846         + parsers.html_declaration_start
9847         + parsers.html_instruction_start
9848         - parsers.html_close_special_tag
9849         - parsers.html_empty_special_tag)
9850 else
9851     parsers.html_interrupting = parsers.fail
9852 end
9853
9854 parsers.ListStarter = parsers.starter
9855
9856 parsers.EndlineExceptions
9857     = parsers.blankline -- paragraph break
9858     + parsers.eof       -- end of document
9859     + parsers.bqstart
9860     + parsers.thematic_break_lines
9861     + parsers.interrupting_bullets
9862     + parsers.interrupting_enumerators
9863     + parsers.headerstart
9864     + parsers.html_interrupting
9865
9866 parsers.NoSoftLineBreakEndlineExceptions = parsers.EndlineExceptions
9867
9868 parsers.endline = parsers.newline
9869     * (parsers.check_minimal_indent
9870     * -V("EndlineExceptions")
9871     + parsers.check_optional_indent

```

```

9872         * -V("EndlineExceptions")
9873         * -V("ListStarter")) / function(_) return end
9874     * parsers.spacechar^0
9875
9876     parsers.Endline = parsers.endline
9877                     / writer.soft_line_break
9878
9879     parsers.EndlineNoSub = parsers.endline
9880
9881     parsers.NoSoftLineBreakEndline
9882         = parsers.newline
9883         * (parsers.check_minimal_indent
9884           * -V("NoSoftLineBreakEndlineExceptions")
9885           + parsers.check_optional_indent
9886           * -V("NoSoftLineBreakEndlineExceptions")
9887           * -V("ListStarter"))
9888         * parsers.spacechar^0
9889         / writer.space
9890
9891     parsers.EndlineBreak = parsers.backslash * parsers.endline
9892                           / writer.hard_line_break
9893
9894     parsers.OptionalIndent
9895         = parsers.spacechar^1 / writer.space
9896
9897     parsers.Space        = parsers.spacechar^2 * parsers.endline
9898                           / writer.hard_line_break
9899
9900         + parsers.spacechar^1
9901         * parsers.endline~-1
9902         * parsers.eof / self.expandtabs
9903         + parsers.spacechar^1 * parsers.endline
9904                           / writer.soft_line_break
9905
9906         + parsers.spacechar^1
9907         * -parsers.newline / self.expandtabs
9908         + parsers.spacechar^1
9909
9910     parsers.NoSoftLineBreakSpace
9911         = parsers.spacechar^2 * parsers.endline
9912                           / writer.hard_line_break
9913
9914         + parsers.spacechar^1
9915         * parsers.endline~-1
9916         * parsers.eof / self.expandtabs
9917         + parsers.spacechar^1 * parsers.endline
9918                           / writer.soft_line_break
9919
9920         + parsers.spacechar^1
9921         * -parsers.newline / self.expandtabs
9922         + parsers.spacechar^1

```

```

9919
9920 parsers.NonbreakingEndline
9921         = parsers.endline
9922         / writer.nbsp
9923
9924 parsers.NonbreakingSpace
9925         = parsers.spacechar^2 * parsers.endline
9926         / writer.nbsp
9927         + parsers.spacechar^1
9928         * parsers.endline~-1 * parsers.eof / ""
9929         + parsers.spacechar^1 * parsers.endline
9930         * parsers.optionalspace
9931         / writer.nbsp
9932         + parsers.spacechar^1 * parsers.optionalspace
9933         / writer.nbsp
9934

```

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.

```

9935 function self.auto_link_url(url, attributes)
9936   return writer.link(writer.escape(url),
9937                     url, nil, attributes)
9938 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.

```

9939 function self.auto_link_email(email, attributes)
9940   return writer.link(writer.escape(email),
9941                     "mailto:".email,
9942                     nil, attributes)
9943 end
9944
9945 parsers.AutoLinkUrl = parsers.auto_link_url
9946                     / self.auto_link_url
9947
9948 parsers.AutoLinkEmail
9949         = parsers.auto_link_email
9950         / self.auto_link_email
9951
9952 parsers.AutoLinkRelativeReference
9953         = parsers.auto_link_relative_reference
9954         / self.auto_link_url
9955
9956 parsers.LinkAndEmph = Ct(parsers.link_and_emph_table)
9957                     / self.defer_link_and_emphasis_processing

```

```

9958
9959 parsers.EscapedChar = parsers.backslash
9960                        * C(parsers.escapable) / writer.string
9961
9962 parsers.InlineHtml = Cs(parsers.html_inline_comment)
9963                       / writer.inline_html_comment
9964                       + Cs(parsers.html_any_empty_inline_tag
9965                           + parsers.html_inline_instruction
9966                           + parsers.html_inline_cdatasection
9967                           + parsers.html_inline_declaration
9968                           + parsers.html_any_open_inline_tag
9969                           + parsers.html_any_close_tag)
9970                       / writer.inline_html_tag
9971
9972 parsers.HtmlEntity = parsers.html_entities / writer.string

```

### 3.1.6.9 Block Elements (local)

```

9973 parsers.DisplayHtml = Cs(parsers.check_trail
9974                          * ( parsers.html_comment
9975                              + parsers.html_special_block
9976                              + parsers.html_block
9977                              + parsers.html_any_block
9978                              + parsers.html_instruction
9979                              + parsers.html_cdatasection
9980                              + parsers.html_declaration))
9981                          / writer.block_html_element
9982
9983 parsers.indented_non_blank_line = parsers.indentedline
9984                                   - parsers.blankline
9985
9986 parsers.Verbatim
9987   = Cs( parsers.check_code_trail
9988         * (parsers.line - parsers.blankline)
9989         * (( parsers.check_minimal_blank_indent_and_full_code_trail
9990             * parsers.blankline)^0
9991           * ( (parsers.check_minimal_indent / "")
9992             * parsers.check_code_trail
9993             * (parsers.line - parsers.blankline))^1)^0)
9994       / self.expandtabs / writer.verbatim
9995
9996 parsers.Blockquote   = parsers.blockquote_body
9997                       / writer.blockquote
9998
9999 parsers.ThematicBreak = parsers.thematic_break_lines
10000                       / writer.thematic_break
10001

```

```

10002 parsers.Reference      = parsers.define_reference_parser
10003                          / self.register_link
10004
10005 parsers.Paragraph        = parsers.freeze_trail
10006                          * (Ct((parsers.Inline)^1)
10007                          * (parsers.newline + parsers.eof)
10008                          * parsers.unfreeze_trail
10009                          / writer.paragraph)
10010
10011 parsers.Plain            = parsers.nonindent_space * Ct(parsers.Inline^1)
10012                          / writer.plain

```

### 3.1.6.10 Lists (local)

```

10013
10014 if options.taskLists then
10015   parsers.tickbox = ( parsers.ticked_box
10016                     + parsers.halfticked_box
10017                     + parsers.unticked_box
10018                     ) / writer.tickbox
10019 else
10020   parsers.tickbox = parsers.fail
10021 end
10022
10023 parsers.list_blank = parsers.conditionally_indented_blankline
10024
10025 parsers.ref_or_block_list_separated
10026   = parsers.sep_group_no_output(parsers.list_blank)
10027   * parsers.minimally_indented_ref
10028   + parsers.block_sep_group(parsers.list_blank)
10029   * parsers.minimally_indented_block
10030
10031 parsers.ref_or_block_non_separated
10032   = parsers.minimally_indented_ref
10033   + (parsers.succeed / writer.interblocksep)
10034   * parsers.minimally_indented_block
10035   - parsers.minimally_indented_blankline
10036
10037 parsers.tight_list_loop_body_pair =
10038   parsers.create_loop_body_pair(
10039     parsers.ref_or_block_non_separated,
10040     parsers.minimally_indented_par_or_plain_no_blank,
10041     (parsers.succeed / writer.interblocksep),
10042     (parsers.succeed / writer.paragraphsep))
10043
10044 parsers.loose_list_loop_body_pair =
10045   parsers.create_loop_body_pair(

```

```

10046     parsers.ref_or_block_list_separated,
10047     parsers.minimally_indented_par_or_plain,
10048     parsers.block_sep_group(parsers.list_blank),
10049     parsers.par_sep_group(parsers.list_blank))
10050
10051 parsers.tight_list_content_loop
10052     = V("Block")
10053     * parsers.tight_list_loop_body_pair.block^0
10054     + (V("Paragraph") + V("Plain"))
10055     * parsers.ref_or_block_non_separated
10056     * parsers.tight_list_loop_body_pair.block^0
10057     + (V("Paragraph") + V("Plain"))
10058     * parsers.tight_list_loop_body_pair.par^0
10059
10060 parsers.loose_list_content_loop
10061     = V("Block")
10062     * parsers.loose_list_loop_body_pair.block^0
10063     + (V("Paragraph") + V("Plain"))
10064     * parsers.ref_or_block_list_separated
10065     * parsers.loose_list_loop_body_pair.block^0
10066     + (V("Paragraph") + V("Plain"))
10067     * parsers.loose_list_loop_body_pair.par^0
10068
10069 parsers.list_item_tightness_condition
10070     = -#( parsers.list_blank^0
10071         * parsers.minimally_indented_ref_or_block_or_par)
10072     * remove_indent("li")
10073     + remove_indent("li")
10074     * parsers.fail
10075
10076 parsers.indented_content_tight
10077     = Ct( (parsers.blankline / "")
10078         * #parsers.list_blank
10079         * remove_indent("li")
10080         + ( (V("Reference") + (parsers.blankline / ""))
10081             * parsers.check_minimal_indent
10082             * parsers.tight_list_content_loop
10083             + (V("Reference") + (parsers.blankline / ""))
10084             + (parsers.tickbox^1 / writer.escape)
10085             * parsers.tight_list_content_loop
10086             )
10087         * parsers.list_item_tightness_condition)
10088
10089 parsers.indented_content_loose
10090     = Ct( (parsers.blankline / "")
10091         * #parsers.list_blank
10092         + ( (V("Reference") + (parsers.blankline / ""))

```

```

10093         * parsers.check_minimal_indent
10094         * parsers.loose_list_content_loop
10095         + (V("Reference") + (parsers.blankline / ""))
10096         + (parsers.tickbox~-1 / writer.escape)
10097         * parsers.loose_list_content_loop))
10098
10099     parsers.TightListItem = function(starter)
10100         return -parsers.ThematicBreak
10101             * parsers.add_indent(starter, "li")
10102             * parsers.indented_content_tight
10103     end
10104
10105     parsers.LooseListItem = function(starter)
10106         return -parsers.ThematicBreak
10107             * parsers.add_indent(starter, "li")
10108             * parsers.indented_content_loose
10109             * remove_indent("li")
10110     end
10111
10112     parsers.BulletListOfType = function(bullet_type)
10113         local bullet = parsers.bullet(bullet_type)
10114         return ( Ct( parsers.TightListItem(bullet)
10115             * ( (parsers.check_minimal_indent / "")
10116                 * parsers.TightListItem(bullet)
10117             )^0
10118             )
10119             * Cc(true)
10120             * -#( (parsers.list_blank^0 / "")
10121                 * parsers.check_minimal_indent
10122                 * (bullet - parsers.ThematicBreak)
10123             )
10124             + Ct( parsers.LooseListItem(bullet)
10125                 * ( (parsers.list_blank^0 / "")
10126                     * (parsers.check_minimal_indent / "")
10127                     * parsers.LooseListItem(bullet)
10128                 )^0
10129             )
10130             * Cc(false)
10131         ) / writer.bulletlist
10132     end
10133
10134     parsers.BulletList = parsers.BulletListOfType(parsers.dash)
10135         + parsers.BulletListOfType(parsers.asterisk)
10136         + parsers.BulletListOfType(parsers.plus)
10137
10138     local function ordered_list(items,tight,starter)
10139         local startnum = starter[2][1]

```

```

10140     if options.startNumber then
10141         startnum = tonumber(startnum) or 1 -- fallback for '#'
10142         if startnum ~= nil then
10143             startnum = math.floor(startnum)
10144         end
10145     else
10146         startnum = nil
10147     end
10148     return writer.orderedlist(items,tight,startnum)
10149 end
10150
10151 parsers.OrderedListOfType = function(delimiter_type)
10152     local enumerator = parsers.enumerator(delimiter_type)
10153     return Cg(enumerator, "listtype")
10154         * (Ct( parsers.TightListItem(Cb("listtype"))
10155             * ( (parsers.check_minimal_indent / "")
10156                 * parsers.TightListItem(enumerator))^0)
10157         * Cc(true)
10158         * -#((parsers.list_blank^0 / "")
10159             * parsers.check_minimal_indent * enumerator)
10160     + Ct( parsers.LooseListItem(Cb("listtype"))
10161         * ((parsers.list_blank^0 / "")
10162             * (parsers.check_minimal_indent / "")
10163             * parsers.LooseListItem(enumerator))^0)
10164         * Cc(false)
10165     ) * Ct(Cb("listtype")) / ordered_list
10166 end
10167
10168 parsers.OrderedList = parsers.OrderedListOfType(parsers.period)
10169     + parsers.OrderedListOfType(parsers.rparent)

```

### 3.1.6.11 Blank (local)

```

10170 parsers.Blank          = parsers.blankline / ""
10171                        + V("Reference")

```

### 3.1.6.12 Headings (local)

```

10172 function parsers.parse_heading_text(s)
10173     local inlines = self.parser_functions.parse_inlines(s)
10174     local flatten_inlines = self.writer.flatten_inlines
10175     self.writer.flatten_inlines = true
10176     local flat_text = self.parser_functions.parse_inlines(s)
10177     flat_text = util.rope_to_string(flat_text)
10178     self.writer.flatten_inlines = flatten_inlines
10179     return {flat_text, inlines}
10180 end
10181

```



```

10182  -- parse atx header
10183  parsers.AtxHeading = parsers.check_trail_no_rem
10184                      * Cg(parsers.heading_start, "level")
10185                      * ((C( parsers.optionalspace
10186                          * parsers.hash^0
10187                          * parsers.optionalspace
10188                          * parsers.newline)
10189                          + parsers.spacechar^1
10190                          * C(parsers.line))
10191                      / strip_atx_end
10192                      / parsers.parse_heading_text)
10193                      * Cb("level")
10194                      / writer.heading
10195
10196  parsers.heading_line = parsers.linechar^1
10197                      - parsers.thematic_break_lines
10198
10199  parsers.heading_text = parsers.heading_line
10200                      * ( (V("Endline") / "\n")
10201                      * ( parsers.heading_line
10202                        - parsers.heading_level))^0
10203                      * parsers.newline^-1
10204
10205  parsers.SettextHeading = parsers.freeze_trail
10206                      * parsers.check_trail_no_rem
10207                      * #( parsers.heading_text
10208                        * parsers.check_minimal_indent
10209                        * parsers.check_trail
10210                        * parsers.heading_level)
10211                      * Cs(parsers.heading_text)
10212                      / parsers.parse_heading_text
10213                      * parsers.check_minimal_indent_and_trail
10214                      * parsers.heading_level
10215                      * parsers.newline
10216                      * parsers.unfreeze_trail
10217                      / writer.heading
10218
10219  parsers.Heading = parsers.AtxHeading + parsers.SettextHeading

```

### 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.

```
10220  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.

```

10221     local walkable_syntax = (function(global_walkable_syntax)
10222         local local_walkable_syntax = {}
10223         for lhs, rule in pairs(global_walkable_syntax) do
10224             local_walkable_syntax[lhs] = util.table_copy(rule)
10225         end
10226         return local_walkable_syntax
10227     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.

```

10228     local current_extension_name = nil
10229     self.insert_pattern = function(selector, pattern, pattern_name)
10230         assert(pattern_name == nil or type(pattern_name) == "string")
10231         local _, _, lhs, pos, rhs
10232             = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
10233         assert(lhs ~= nil,
10234             [[Expected selector in form ]]
10235             .. ["LHS (before|after|instead of) RHS", not "]]
10236             .. selector .. ["]])
10237         assert(walkable_syntax[lhs] ~= nil,
10238             [[Rule ]] .. lhs
10239             .. [[ -> ... does not exist in markdown grammar]])
10240         assert(pos == "before" or pos == "after" or pos == "instead of",
10241             [[Expected positional specifier "before", "after", ]]
10242             .. [[or "instead of", not "]]
10243             .. pos .. ["]])
10244         local rule = walkable_syntax[lhs]
10245         local index = nil
10246         for current_index, current_rhs in ipairs(rule) do
10247             if type(current_rhs) == "string" and current_rhs == rhs then
10248                 index = current_index
10249                 if pos == "after" then
10250                     index = index + 1
10251                 end
10252                 break
10253             end
10254         end
10255         assert(index ~= nil,
10256             [[Rule ]] .. lhs .. [[ -> ]] .. rhs
10257             .. [[ does not exist in markdown grammar]])
10258         local accountable_pattern
10259         if current_extension_name then
10260             accountable_pattern

```

```

10261         = {pattern, current_extension_name, pattern_name}
10262     else
10263         assert(type(pattern) == "string",
10264             [[reader->insert_pattern() was called outside ]]
10265             .. [[an extension with ]]
10266             .. [[a PEG pattern instead of a rule name]])
10267         accountable_pattern = pattern
10268     end
10269     if pos == "instead of" then
10270         rule[index] = accountable_pattern
10271     else
10272         table.insert(rule, index, accountable_pattern)
10273     end
10274 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.

```

10275     local syntax =
10276         { "Blocks",
10277
10278           Blocks = V("InitializeState")
10279                 * V("ExpectedJekyllData")
10280                 * 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.

```

10281         * ( V("Block")
10282           * ( V("Blank")^0 * parsers.eof
10283             + ( V("Blank")^2 / writer.paragraphsep
10284               + V("Blank")^0 / writer.interblocksep
10285             )
10286           )
10287         + ( V("Paragraph") + V("Plain") )
10288         * ( V("Blank")^0 * parsers.eof
10289           + ( V("Blank")^2 / writer.paragraphsep
10290             + V("Blank")^0 / writer.interblocksep
10291           )
10292         )
10293         * V("Block")
10294         * ( V("Blank")^0 * parsers.eof
10295           + ( V("Blank")^2 / writer.paragraphsep
10296             + V("Blank")^0 / writer.interblocksep
10297           )
10298         )
10299         + ( V("Paragraph") + V("Plain") )
10300         * ( V("Blank")^0 * parsers.eof
10301           + V("Blank")^0 / writer.paragraphsep

```

```

10302         )
10303     )~0,
10304
10305     ExpectedJekyllData = parsers.succeed,
10306
10307     Blank              = parsers.Blank,
10308     Reference          = parsers.Reference,
10309
10310     Blockquote         = parsers.Blockquote,
10311     Verbatim           = parsers.Verbatim,
10312     ThematicBreak      = parsers.ThematicBreak,
10313     BulletList         = parsers.BulletList,
10314     OrderedList        = parsers.OrderedList,
10315     DisplayHtml        = parsers.DisplayHtml,
10316     Heading            = parsers.Heading,
10317     Paragraph          = parsers.Paragraph,
10318     Plain              = parsers.Plain,
10319
10320     ListStarter         = parsers.ListStarter,
10321     EndlineExceptions   = parsers.EndlineExceptions,
10322     NoSoftLineBreakEndlineExceptions
10323                         = parsers.NoSoftLineBreakEndlineExceptions,
10324
10325     Str                = parsers.Str,
10326     Space              = parsers.Space,
10327     NoSoftLineBreakSpace
10328                         = parsers.NoSoftLineBreakSpace,
10329     OptionalIndent     = parsers.OptionalIndent,
10330     Endline            = parsers.Endline,
10331     EndlineNoSub       = parsers.EndlineNoSub,
10332     NoSoftLineBreakEndline
10333                         = parsers.NoSoftLineBreakEndline,
10334     EndlineBreak       = parsers.EndlineBreak,
10335     LinkAndEmph        = parsers.LinkAndEmph,
10336     Code               = parsers.Code,
10337     AutoLinkUrl        = parsers.AutoLinkUrl,
10338     AutoLinkEmail      = parsers.AutoLinkEmail,
10339     AutoLinkRelativeReference
10340                         = parsers.AutoLinkRelativeReference,
10341     InlineHtml         = parsers.InlineHtml,
10342     HtmlEntity         = parsers.HtmlEntity,
10343     EscapedChar        = parsers.EscapedChar,
10344     Smart              = parsers.Smart,
10345     Symbol             = parsers.Symbol,
10346     SpecialChar        = parsers.fail,
10347     InitializeState    = parsers.succeed,
10348 }

```

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]`.

```

10349     self.update_rule = function(rule_name, get_pattern)
10350         assert(current_extension_name ~= nil)
10351         assert(syntax[rule_name] ~= nil,
10352             [[Rule ]] .. rule_name
10353             .. [[ -> ... does not exist in markdown grammar]])
10354         local previous_pattern
10355         local extension_name
10356         if walkable_syntax[rule_name] then
10357             local previous_accountable_pattern
10358             = walkable_syntax[rule_name][1]
10359             previous_pattern = previous_accountable_pattern[1]
10360             extension_name
10361             = previous_accountable_pattern[2]
10362             .. ", " .. current_extension_name
10363         else
10364             previous_pattern = nil
10365             extension_name = current_extension_name
10366         end
10367         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

```

```

10368     if type(get_pattern) == "function" then
10369         pattern = get_pattern(previous_pattern)
10370     else
10371         assert(previous_pattern == nil,
10372             [[Rule ]] .. rule_name ..
10373             [[ has already been updated by ]] .. extension_name)
10374         pattern = get_pattern
10375     end
10376     local accountable_pattern = { pattern, extension_name, rule_name }
10377     walkable_syntax[rule_name] = { accountable_pattern }
10378 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.

```

10379     local special_characters = {}
10380     self.add_special_character = function(c)
10381         table.insert(special_characters, c)
10382         syntax.SpecialChar = S(table.concat(special_characters, ""))
10383     end
10384
10385     self.add_special_character("*")
10386     self.add_special_character("[")
10387     self.add_special_character("]")
10388     self.add_special_character("<")
10389     self.add_special_character("!")
10390     self.add_special_character("\\")

```

Add method `reader->initialize_named_group` that defines named groups with a default capture value.

```

10391     self.initialize_named_group = function(name, value)
10392         local pattern = Ct("")
10393         if value ~= nil then
10394             pattern = pattern / value
10395         end
10396         syntax.InitializeState = syntax.InitializeState
10397                                 * Cg(pattern, name)
10398     end

```

Add a named group for indentation.

```

10399     self.initialize_named_group("indent_info")

```

Apply syntax extensions.

```

10400     for _, extension in ipairs(extensions) do
10401         current_extension_name = extension.name
10402         extension.extend_writer(writer)
10403         extension.extend_reader(self)
10404     end
10405     current_extension_name = nil

```

If the `debugExtensions` option is enabled, serialize `walkable_syntax` to a JSON for debugging purposes.

```

10406     if options.debugExtensions then
10407         local sorted_lhs = {}
10408         for lhs, _ in pairs(walkable_syntax) do
10409             table.insert(sorted_lhs, lhs)
10410         end
10411         table.sort(sorted_lhs)
10412
10413         local output_lines = {"{"}

```

```

10414     for lhs_index, lhs in ipairs(sorted_lhs) do
10415         local encoded_lhs = util.encode_json_string(lhs)
10416         table.insert(output_lines, [[      ]] .. encoded_lhs .. [[(: []]])
10417         local rule = walkable_syntax[lhs]
10418         for rhs_index, rhs in ipairs(rule) do
10419             local human_readable_rhs
10420             if type(rhs) == "string" then
10421                 human_readable_rhs = rhs
10422             else
10423                 local pattern_name
10424                 if rhs[3] then
10425                     pattern_name = rhs[3]
10426                 else
10427                     pattern_name = "Anonymous Pattern"
10428                 end
10429                 local extension_name = rhs[2]
10430                 human_readable_rhs = pattern_name .. [[ (]]
10431                     .. extension_name .. [[]]]
10432             end
10433             local encoded_rhs
10434                 = util.encode_json_string(human_readable_rhs)
10435             local output_line = [[      ]] .. encoded_rhs
10436             if rhs_index < #rule then
10437                 output_line = output_line .. ","
10438             end
10439             table.insert(output_lines, output_line)
10440         end
10441         local output_line = "    ]"
10442         if lhs_index < #sorted_lhs then
10443             output_line = output_line .. ","
10444         end
10445         table.insert(output_lines, output_line)
10446     end
10447     table.insert(output_lines, "}")
10448
10449     local output = table.concat(output_lines, "\n")
10450     local output_filename = options.debugExtensionsFileName
10451     local output_file = assert(io.open(output_filename, "w"),
10452         [[Could not open file ]] .. output_filename
10453         .. [[ for writing]])
10454     assert(output_file:write(output))
10455     assert(output_file:close())
10456 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](#).

```

10457     for lhs, rule in pairs(walkable_syntax) do
10458         syntax[lhs] = parsers.fail
10459         for _, rhs in ipairs(rule) do
10460             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.

```

10461         if type(rhs) == "string" then
10462             pattern = V(rhs)
10463         else
10464             pattern = rhs[1]
10465             if type(pattern) == "string" then
10466                 pattern = V(pattern)
10467             end
10468         end
10469         syntax[lhs] = syntax[lhs] + pattern
10470     end
10471 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.

```

10472     if options.underscores then
10473         self.add_special_character("_")
10474     end
10475
10476     if not options.codeSpans then
10477         syntax.Code = parsers.fail
10478     else
10479         self.add_special_character("`")
10480     end
10481
10482     if not options.html then
10483         syntax.DisplayHtml = parsers.fail
10484         syntax.InlineHtml = parsers.fail
10485         syntax.HtmlEntity = parsers.fail
10486     else
10487         self.add_special_character("&")
10488     end
10489
10490     if options.preserveTabs then
10491         options.stripIndent = false
10492     end
10493
10494     if not options.smartEllipses then

```



```

10495     syntax.Smart = parsers.fail
10496 else
10497     self.add_special_character(".")
10498 end
10499
10500 if not options.relativeReferences then
10501     syntax.AutoLinkRelativeReference = parsers.fail
10502 end
10503
10504 if options.contentLevel == "inline" then
10505     syntax[1] = "Inlines"
10506     syntax.Inlines = V("InitializeState")
10507         * parsers.Inline^0
10508         * ( parsers.spacing^0
10509             * parsers.eof / "" )
10510     syntax.Space = parsers.Space + parsers.blankline / writer.space
10511 end
10512
10513 local blocks_nested_t = util.table_copy(syntax)
10514 blocks_nested_t.ExpectedJekyllData = parsers.succeed
10515 parsers.blocks_nested = Ct(blocks_nested_t)
10516
10517 parsers.blocks = Ct(syntax)
10518
10519 local inlines_t = util.table_copy(syntax)
10520 inlines_t[1] = "Inlines"
10521 inlines_t.Inlines = V("InitializeState")
10522     * parsers.Inline^0
10523     * ( parsers.spacing^0
10524         * parsers.eof / "" )
10525 parsers.inlines = Ct(inlines_t)
10526
10527 local inlines_no_inline_note_t = util.table_copy(inlines_t)
10528 inlines_no_inline_note_t.InlineNote = parsers.fail
10529 parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
10530
10531 local inlines_no_html_t = util.table_copy(inlines_t)
10532 inlines_no_html_t.DisplayHtml = parsers.fail
10533 inlines_no_html_t.InlineHtml = parsers.fail
10534 inlines_no_html_t.HtmlEntity = parsers.fail
10535 parsers.inlines_no_html = Ct(inlines_no_html_t)
10536
10537 local inlines_nbsp_t = util.table_copy(inlines_t)
10538 inlines_nbsp_t.Endline = parsers.NonbreakingEndline
10539 inlines_nbsp_t.Space = parsers.NonbreakingSpace
10540 parsers.inlines_nbsp = Ct(inlines_nbsp_t)
10541

```

```

10542     local inlines_no_link_or_emphasis_t = util.table_copy(inlines_t)
10543     inlines_no_link_or_emphasis_t.LinkAndEmph = parsers.fail
10544     inlines_no_link_or_emphasis_t.EndlineExceptions
10545         = parsers.EndlineExceptions - parsers.eof
10546     parsers.inlines_no_link_or_emphasis
10547         = Ct(inlines_no_link_or_emphasis_t)

```

Return a function that converts markdown string `input` into a plain TeX output and returns it..

```

10548     return function(input)

```

Unicode-normalize the input.

```

10549         if options.unicodeNormalization then
10550             local form = options.unicodeNormalizationForm
10551             if form == "nfc" then
10552                 input = uni_algos.normalize.NFC(input)
10553             elseif form == "nfd" then
10554                 input = uni_algos.normalize.NFD(input)
10555             elseif form == "nfkc" then
10556                 input = uni_algos.normalize.NFKC(input)
10557             elseif form == "nfkd" then
10558                 input = uni_algos.normalize.NFKD(input)
10559             else
10560                 return writer.error(
10561                     format("Unknown normalization form %s.", form))
10562             end
10563         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.

```

10564         input = input:gsub("\r\n?", "\n")
10565         if input:sub(-1) ~= "\n" then
10566             input = input .. "\n"
10567         end

```

Clear the table of references.

```

10568         references = {}
10569         local document = self.parser_functions.parse_blocks(input)
10570         local output = util.rope_to_string(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`.

```

10571         local undosep_start, undosep_end
10572         local potential_secend_start, secend_start
10573         local potential_sep_start, sep_start
10574         while true do
10575             -- find a `writer->undosep`
10576             undosep_start, undosep_end

```

```

10577         = output:find(writer.undosep_text, 1, true)
10578     if undosep_start == nil then break end
10579     -- skip any preceding section ends
10580     secend_start = undosep_start
10581     while true do
10582         potential_secend_start = secend_start - #writer.secend_text
10583         if potential_secend_start < 1
10584             or output:sub(potential_secend_start,
10585                 secend_start - 1) ~= writer.secend_text
10586             then
10587                 break
10588             end
10589             secend_start = potential_secend_start
10590         end
10591         -- find an immediately preceding
10592         -- block element / paragraph separator
10593         sep_start = secend_start
10594         potential_sep_start = sep_start - #writer.interblocksep_text
10595         if potential_sep_start >= 1
10596             and output:sub(potential_sep_start,
10597                 sep_start - 1) == writer.interblocksep_text
10598             then
10599                 sep_start = potential_sep_start
10600             else
10601                 potential_sep_start = sep_start - #writer.paragraphsep_text
10602                 if potential_sep_start >= 1
10603                     and output:sub(potential_sep_start,
10604                         sep_start - 1) == writer.paragraphsep_text
10605                     then
10606                         sep_start = potential_sep_start
10607                     end
10608                 end
10609                 -- remove `writer->undosep` and immediately preceding
10610                 -- block element / paragraph separator
10611                 output = output:sub(1, sep_start - 1)
10612                 .. output:sub(secend_start, undosep_start - 1)
10613                 .. output:sub(undosep_end + 1)
10614             end
10615             return output
10616         end
10617     end
10618     return self
10619 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.

```
10620 M.extensions = {}
```

### 3.1.7.1 Bracketed Spans

The `extensions.bracketed_spans` function implements the Pandoc bracketed span syntax extension.

```
10621 M.extensions.bracketed_spans = function()
10622   return {
10623     name = "built-in bracketed_spans syntax extension",
10624     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.

```
10625       function self.span(s, attr)
10626         if self.flatten_inlines then return s end
10627         return {"\\markdownRendererBracketedSpanAttributeContextBegin",
10628               self.attributes(attr),
10629               s,
10630               "\\markdownRendererBracketedSpanAttributeContextEnd{}}"}
10631       end
10632   end, extend_reader = function(self)
10633     local parsers = self.parsers
10634     local writer = self.writer
10635
10636     local span_label = parsers.lbracket
10637                       * (Cs((parsers.alphanumeric^1
10638                             + parsers.inticks
10639                             + parsers.autolink
10640                             + V("InlineHtml")
10641                             + ( parsers.backslash * parsers.backslash)
10642                             + ( parsers.backslash
10643                               * (parsers.lbracket + parsers.rbracket)
10644                               + V("Space") + V("Endline")
10645                               + (parsers.any
10646                                 - ( parsers.newline
10647                                   + parsers.lbracket
10648                                   + parsers.rbracket
10649                                   + parsers.blankline^2))))^1)
10650                       / self.parser_functions.parse_inlines)
10651                       * parsers.rbracket
10652
10653     local Span = span_label
10654               * Ct(parsers.attributes)
```

```

10655             / writer.span
10656
10657         self.insert_pattern("Inline before LinkAndEmph",
10658                             Span, "Span")
10659     end
10660 }
10661 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.

```

10662 M.extensions.citations = function(citation_nbsps)
10663     return {
10664         name = "built-in citations syntax extension",
10665         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.

```

10666     function self.citations(text_cites, cites)
10667         local buffer = {}
10668         if self.flatten_inlines then
10669             for _,cite in ipairs(cites) do
10670                 if cite.prenote then
10671                     table.insert(buffer, {cite.prenote, " "})
10672                 end
10673                 table.insert(buffer, cite.name)
10674                 if cite.postnote then
10675                     table.insert(buffer, {" ", cite.postnote})
10676                 end
10677             end
10678         else

```

```

10679         table.insert(buffer,
10680             {"\\markdownRenderer",
10681             text_cites and "TextCite" or "Cite",
10682             "{", #cites, "}"})
10683     for _,cite in ipairs(cites) do
10684         table.insert(buffer,
10685             {cite.suppress_author and "-" or "+", "{",
10686             cite.prenote or "", "}{" ,
10687             cite.postnote or "", "}{" , cite.name, "}"})
10688     end
10689 end
10690 return buffer
10691 end
10692 end, extend_reader = function(self)
10693     local parsers = self.parsers
10694     local writer = self.writer
10695
10696     local citation_chars
10697         = parsers.alphanumeric
10698         + S("#$%&-+<>~/_")
10699
10700     local citation_name
10701         = Cs(parsers.dash~1) * parsers.at
10702         * Cs(citation_chars
10703             * ((( citation_chars
10704                 + parsers.internal_punctuation
10705                 - parsers.comma - parsers.semicolon)
10706             * -#(( parsers.internal_punctuation
10707                 - parsers.comma
10708                 - parsers.semicolon)^0
10709             * -( citation_chars
10710                 + parsers.internal_punctuation
10711                 - parsers.comma
10712                 - parsers.semicolon)))^0
10713             * citation_chars)^-1)
10714
10715     local citation_body_prenote
10716         = Cs((parsers.alphanumeric^1
10717             + parsers.bracketed
10718             + parsers.inticks
10719             + parsers.autolink
10720             + V("InlineHtml")
10721             + V("Space") + V("EndlineNoSub")
10722             + (parsers.anyescaped
10723                 - ( parsers.newline
10724                     + parsers.rbracket
10725                     + parsers.blankline^2))

```

```

10726         - ( parsers.spnl
10727           * parsers.dash~-1
10728           * parsers.at))^1)
10729
10730     local citation_body_postnote
10731       = Cs((parsers.alphanumeric~1
10732         + parsers.bracketed
10733         + parsers.inticks
10734         + parsers.autolink
10735         + V("InlineHtml")
10736         + V("Space") + V("EndlineNoSub")
10737         + (parsers.anyescaped
10738           - ( parsers.newline
10739             + parsers.rbracket
10740             + parsers.semicolon
10741             + parsers.blankline~2))
10742         - (parsers.spnl * parsers.rbracket))^1)
10743
10744     local citation_body_chunk
10745       = ( citation_body_prenote
10746         * parsers.spnlc_sep
10747         + Cc("")
10748         * parsers.spnlc
10749       )
10750       * citation_name
10751       * ( parsers.internal_punctuation
10752         - parsers.semicolon)^-1
10753       * ( parsers.spnlc / function(_) return end
10754         * citation_body_postnote
10755         + Cc("")
10756         * parsers.spnlc
10757       )
10758
10759     local citation_body
10760       = citation_body_chunk
10761       * ( parsers.semicolon
10762         * parsers.spnlc
10763         * citation_body_chunk
10764       )^0
10765
10766     local citation_headless_body_postnote
10767       = Cs((parsers.alphanumeric~1
10768         + parsers.bracketed
10769         + parsers.inticks
10770         + parsers.autolink
10771         + V("InlineHtml")
10772         + V("Space") + V("Endline")

```

```

10773         + (parsers.anyescaped
10774         - ( parsers.newline
10775         + parsers.rbracket
10776         + parsers.at
10777         + parsers.semicolon + parsers.blankline^2))
10778         - (parsers.spnl * parsers.rbracket))^0)
10779
10780     local citation_headless_body
10781         = citation_headless_body_postnote
10782         * ( parsers.semicolon
10783         * parsers.spnlc
10784         * citation_body_chunk
10785         )^0
10786
10787     local citations
10788         = function(text_cites, raw_cites)
10789         local function normalize(str)
10790             if str == "" then
10791                 str = nil
10792             else
10793                 str = (citation_nbsps and
10794                     self.parser_functions.parse_inlines_nbsp or
10795                     self.parser_functions.parse_inlines)(str)
10796             end
10797             return str
10798         end
10799
10800         local cites = {}
10801         for i = 1,#raw_cites,4 do
10802             cites[#cites+1] = {
10803                 prenote = normalize(raw_cites[i]),
10804                 suppress_author = raw_cites[i+1] == "-",
10805                 name = writer.identifier(raw_cites[i+2]),
10806                 postnote = normalize(raw_cites[i+3]),
10807             }
10808         end
10809         return writer.citations(text_cites, cites)
10810     end
10811
10812     local TextCitations
10813         = Ct((parsers.spnlc
10814         * Cc("")
10815         * citation_name
10816         * ((parsers.spnlc
10817         * parsers.lbracket
10818         * citation_headless_body
10819         * parsers.rbracket) + Cc("")))^1)

```



```

10820             / function(raw_cites)
10821                 return citations(true, raw_cites)
10822             end
10823
10824     local ParenthesizedCitations
10825         = Ct((parsers.spnlc
10826             * parsers.lbracket
10827             * citation_body
10828             * parsers.rbracket)^1)
10829     / function(raw_cites)
10830         return citations(false, raw_cites)
10831     end
10832
10833     local Citations = TextCitations + ParenthesizedCitations
10834
10835     self.insert_pattern("Inline before LinkAndEmph",
10836         Citations, "Citations")
10837
10838     self.add_special_character("@")
10839     self.add_special_character("-")
10840 end
10841 }
10842 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.

```

10843 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.

```

10844     local languages_json = (function()
10845         local base, prev, curr
10846         for _, pathname in ipairs{kpse.lookup(language_map,
10847             {all=true})} do
10848             local file = io.open(pathname, "r")
10849             if not file then goto continue end
10850             local input = assert(file:read("*a"))
10851             assert(file:close())
10852             local json = input:gsub('"[^\\n]-"', '[%1]=')
10853             curr = load("_ENV = {}; return "..json")()
10854             if type(curr) == "table" then
10855                 if base == nil then

```

```

10856         base = curr
10857     else
10858         setmetatable(prev, { __index = curr })
10859     end
10860     prev = curr
10861 end
10862 ::continue::
10863 end
10864 return base or {}
10865 end)()
10866
10867 return {
10868     name = "built-in content_blocks syntax extension",
10869     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.

```

10870     function self.contentblock(src,suf,type,tit)
10871         if not self.is_writing then return "" end
10872         src = src.." "..suf
10873         suf = suf:lower()
10874         if type == "onlineimage" then
10875             return {"\\markdownRendererContentBlockOnlineImage{" ,suf,"} ",
10876                 "{" ,self.string(src),"} ",
10877                 "{" ,self.uri(src),"} ",
10878                 "{" ,self.string(tit or ""),"} "}
10879         elseif languages_json[suf] then
10880             return {"\\markdownRendererContentBlockCode{" ,suf,"} ",
10881                 "{" ,self.string(languages_json[suf]),"} ",
10882                 "{" ,self.string(src),"} ",
10883                 "{" ,self.uri(src),"} ",
10884                 "{" ,self.string(tit or ""),"} "}
10885         else
10886             return {"\\markdownRendererContentBlock{" ,suf,"} ",
10887                 "{" ,self.string(src),"} ",
10888                 "{" ,self.uri(src),"} ",
10889                 "{" ,self.string(tit or ""),"} "}
10890         end
10891     end
10892 end, extend_reader = function(self)
10893     local parsers = self.parsers
10894     local writer = self.writer
10895
10896     local contentblock_tail
10897         = parsers.optionalttitle

```

```

10898             * (parsers.newline + parsers.eof)
10899
10900 -- case insensitive online image suffix:
10901 local onlineimagesuffix
10902     = (function(...)
10903         local parser = nil
10904         for _, suffix in ipairs({...}) do
10905             local pattern=nil
10906             for i=1,#suffix do
10907                 local char=suffix:sub(i,i)
10908                 char = S(char:lower()..char:upper())
10909                 if pattern == nil then
10910                     pattern = char
10911                 else
10912                     pattern = pattern * char
10913                 end
10914             end
10915             if parser == nil then
10916                 parser = pattern
10917             else
10918                 parser = parser + pattern
10919             end
10920         end
10921         return parser
10922     end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
10923
10924 -- online image url for iA Writer content blocks with
10925 -- mandatory suffix, allowing nested brackets:
10926 local onlineimageurl
10927     = (parsers.less
10928         * Cs((parsers.anyescaped
10929             - parsers.more
10930             - parsers.spacing
10931             - #(parsers.period
10932                 * onlineimagesuffix
10933                 * parsers.more
10934                 * contentblock_tail))^0)
10935         * parsers.period
10936         * Cs(onlineimagesuffix)
10937         * parsers.more
10938         + (Cs((parsers.inparens
10939             + (parsers.anyescaped
10940                 - parsers.spacing
10941                 - parsers.rparent
10942                 - #(parsers.period
10943                     * onlineimagesuffix
10944                     * contentblock_tail))))^0)

```

```

10945             * parsers.period
10946             * Cs(onlineimagesuffix))
10947         ) * Cc("onlineimage")
10948
10949     -- filename for iA Writer content blocks with mandatory suffix:
10950     local localfilepath
10951         = parsers.slash
10952         * Cs((parsers.anyescaped
10953             - parsers.tab
10954             - parsers.newline
10955             - #(parsers.period
10956                 * parsers.alphanumeric^1
10957                 * contentblock_tail))^1)
10958         * parsers.period
10959         * Cs(parsers.alphanumeric^1)
10960         * Cc("localfile")
10961
10962     local ContentBlock
10963         = parsers.check_trail_no_rem
10964         * (localfilepath + onlineimageurl)
10965         * contentblock_tail
10966         / writer.contentblock
10967
10968     self.insert_pattern("Block before Blockquote",
10969                       ContentBlock, "ContentBlock")
10970 end
10971 }
10972 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.

```

10973 M.extensions.definition_lists = function(tight_lists)
10974     return {
10975         name = "built-in definition_lists syntax extension",
10976         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.

```

10977         local function dliem(term, defs)
10978             local retVal = {"\\markdownRendererDlItem{",term,""}
10979             for _, def in ipairs(defs) do
10980                 retVal[#retVal+1]

```

```

10981         = {"\\markdownRendererDlDefinitionBegin ",def,
10982             "\\markdownRendererDlDefinitionEnd "}
10983     end
10984     retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
10985     return retVal
10986 end
10987
10988 function self.definitionlist(items,tight)
10989     if not self.is_writing then return "" end
10990     local buffer = {}
10991     for _,item in ipairs(items) do
10992         buffer[#buffer + 1] = dliitem(item.term, item.definitions)
10993     end
10994     if tight and tight_lists then
10995         return {"\\markdownRendererDlBeginTight\n", buffer,
10996             "\n\\markdownRendererDlEndTight"}
10997     else
10998         return {"\\markdownRendererDlBegin\n", buffer,
10999             "\n\\markdownRendererDlEnd"}
11000     end
11001 end
11002 end, extend_reader = function(self)
11003     local parsers = self.parsers
11004     local writer = self.writer
11005
11006     local defstartchar = S("~:")
11007
11008     local defstart
11009     = parsers.check_trail_length(0) * defstartchar
11010     * #parsers.spacing
11011     * (parsers.tab + parsers.space~-3)
11012     + parsers.check_trail_length(1)
11013     * defstartchar * #parsers.spacing
11014     * (parsers.tab + parsers.space~-2)
11015     + parsers.check_trail_length(2)
11016     * defstartchar * #parsers.spacing
11017     * (parsers.tab + parsers.space~-1)
11018     + parsers.check_trail_length(3)
11019     * defstartchar * #parsers.spacing
11020
11021     local indented_line
11022     = (parsers.check_minimal_indent / "")
11023     * parsers.check_code_trail * parsers.line
11024
11025     local blank
11026     = parsers.check_minimal_blank_indent_and_any_trail
11027     * parsers.optionalspace * parsers.newline

```

```

11028
11029     local dlchunk = Cs(parsers.line * (indented_line - blank)^0)
11030
11031     local indented_blocks = function(bl)
11032         return Cs( bl
11033             * (blank^1 * (parsers.check_minimal_indent / ""))
11034             * parsers.check_code_trail * -parsers.blankline * bl)^0
11035             * (blank^1 + parsers.eof))
11036     end
11037
11038     local function definition_list_item(term, defs, _)
11039         return { term = self.parser_functions.parse_inlines(term),
11040             definitions = defs }
11041     end
11042
11043     local DefinitionListItemLoose
11044     = C(parsers.line) * blank^0
11045     * Ct((parsers.check_minimal_indent * (defstart
11046         * indented_blocks(dlchunk)
11047         / self.parser_functions.parse_blocks_nested))^1)
11048     * Cc(false) / definition_list_item
11049
11050     local DefinitionListItemTight
11051     = C(parsers.line)
11052     * Ct((parsers.check_minimal_indent * (defstart * dlchunk
11053         / self.parser_functions.parse_blocks_nested))^1)
11054     * Cc(true) / definition_list_item
11055
11056     local DefinitionList
11057     = ( Ct(DefinitionListItemLoose^1) * Cc(false)
11058         + Ct(DefinitionListItemTight^1)
11059         * (blank^0
11060             * -DefinitionListItemLoose * Cc(true))
11061         ) / writer.definitionlist
11062
11063     self.insert_pattern("Block after Heading",
11064         DefinitionList, "DefinitionList")
11065 end
11066 }
11067 end

```

### 3.1.7.5 Fancy Lists

The `extensions.fancy_lists` function implements the Pandoc fancy list syntax extension.

```

11068 M.extensions.fancy_lists = function()
11069     return {

```

```

11070     name = "built-in fancy_lists syntax extension",
11071     extend_writer = function(self)
11072         local options = self.options
11073

```

Define `writer->fancylis`t 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.

```

11074     function self.fancylis(items,tight,startnum,numstyle,numdelim)
11075         if not self.is_writing then return "" end
11076         local buffer = {}
11077         local num = startnum
11078         for _,item in ipairs(items) do
11079             if item ~= "" then
11080                 buffer[#buffer + 1] = self.fancyitem(item,num)
11081             end
11082             if num ~= nil and item ~= "" then
11083                 num = num + 1
11084             end
11085         end
11086         local contents = util.intersperse(buffer,"\n")
11087         if tight and options.tightLists then
11088             return {"\\markdownRenderFancy01BeginTight{",
11089                 numstyle,"}{",numdelim,"}",contents,
11090                 "\\markdownRenderFancy01EndTight "}
11091         else
11092             return {"\\markdownRenderFancy01Begin{",
11093                 numstyle,"}{",numdelim,"}",contents,
11094                 "\\markdownRenderFancy01End "}

```

```

11095         end
11096     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.

```

11097     function self.fancyitem(s,num)
11098         if num ~= nil then
11099             return {"\\markdownRendererFancyOliItemWithNumber{",num,"}",s,
11100                 "\\markdownRendererFancyOliItemEnd "}
11101         else
11102             return {"\\markdownRendererFancyOliItem ",s,
11103                 "\\markdownRendererFancyOliItemEnd "}
11104         end
11105     end
11106 end, extend_reader = function(self)
11107     local parsers = self.parsers
11108     local options = self.options
11109     local writer = self.writer
11110
11111     local function combine_markers_and_delims(markers, delims)
11112         local markers_table = {}
11113         for _,marker in ipairs(markers) do
11114             local start_marker
11115             local continuation_marker
11116             if type(marker) == "table" then
11117                 start_marker = marker[1]
11118                 continuation_marker = marker[2]
11119             else
11120                 start_marker = marker
11121                 continuation_marker = marker
11122             end
11123             for _,delim in ipairs(delims) do
11124                 table.insert(markers_table,
11125                     {start_marker, continuation_marker, delim})
11126             end
11127         end
11128         return markers_table
11129     end
11130
11131     local function join_table_with_func(func, markers_table)
11132         local pattern = func(table.unpack(markers_table[1]))
11133         for i = 2, #markers_table do
11134             pattern = pattern + func(table.unpack(markers_table[i]))
11135         end
11136         return pattern
11137     end

```



```

11138
11139     local lowercase_letter_marker = R("az")
11140     local uppercase_letter_marker = R("AZ")
11141
11142     local roman_marker = function(chars)
11143         local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])
11144         local l, x, v, i
11145             = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])
11146         return  m^-3
11147             * (c*m + c*d + d^-1 * c^-3)
11148             * (x*c + x*l + l^-1 * x^-3)
11149             * (i*x + i*v + v^-1 * i^-3)
11150     end
11151
11152     local lowercase_roman_marker
11153         = roman_marker({"m", "d", "c", "l", "x", "v", "i"})
11154     local uppercase_roman_marker
11155         = roman_marker({"M", "D", "C", "L", "X", "V", "I"})
11156
11157     local lowercase_opening_roman_marker = P("i")
11158     local uppercase_opening_roman_marker = P("I")
11159
11160     local digit_marker = parsers.dig * parsers.dig^-8
11161
11162     local markers = {
11163         {lowercase_opening_roman_marker, lowercase_roman_marker},
11164         {uppercase_opening_roman_marker, uppercase_roman_marker},
11165         lowercase_letter_marker,
11166         uppercase_letter_marker,
11167         lowercase_roman_marker,
11168         uppercase_roman_marker,
11169         digit_marker
11170     }
11171
11172     local delims = {
11173         parsers.period,
11174         parsers.rparent
11175     }
11176
11177     local markers_table = combine_markers_and_delims(markers, delims)
11178
11179     local function enumerator(start_marker, _,
11180                               delimiter_type, interrupting)
11181         local delimiter_range
11182         local allowed_end
11183         if interrupting then
11184             delimiter_range = P("1")

```

```

11185         allowed_end = C(parsers.spacechar~1) * #parsers.linechar
11186     else
11187         delimiter_range = start_marker
11188         allowed_end = C(parsers.spacechar~1)
11189             + #(parsers.newline + parsers.eof)
11190     end
11191
11192     return parsers.check_trail
11193         * Ct(C(delimiter_range) * C(delimiter_type))
11194         * allowed_end
11195 end
11196
11197 local starter = join_table_with_func(enumerator, markers_table)
11198
11199 local TightListItem = function(starter)
11200     return parsers.add_indent(starter, "li")
11201         * parsers.indented_content_tight
11202 end
11203
11204 local LooseListItem = function(starter)
11205     return parsers.add_indent(starter, "li")
11206         * parsers.indented_content_loose
11207         * remove_indent("li")
11208 end
11209
11210 local function roman2number(roman)
11211     local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100,
11212         ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
11213     local numeral = 0
11214
11215     local i = 1
11216     local len = string.len(roman)
11217     while i < len do
11218         local z1, z2 = romans[ string.sub(roman, i, i) ],
11219             romans[ string.sub(roman, i+1, i+1) ]
11220         if z1 < z2 then
11221             numeral = numeral + (z2 - z1)
11222             i = i + 2
11223         else
11224             numeral = numeral + z1
11225             i = i + 1
11226         end
11227     end
11228     if i <= len then
11229         numeral = numeral + romans[ string.sub(roman,i,i) ]
11230     end
11231     return numeral

```

```

11232     end
11233
11234     local function sniffstyle(numstr, delimend)
11235         local numdelim
11236         if delimend == ")" then
11237             numdelim = "OneParen"
11238         elseif delimend == "." then
11239             numdelim = "Period"
11240         else
11241             numdelim = "Default"
11242         end
11243
11244         local num
11245         num = numstr:match("^([I])$")
11246         if num then
11247             return roman2number(num), "UpperRoman", numdelim
11248         end
11249         num = numstr:match("^([i])$")
11250         if num then
11251             return roman2number(string.upper(num)), "LowerRoman", numdelim
11252         end
11253         num = numstr:match("^([A-Z])$")
11254         if num then
11255             return string.byte(num) - string.byte("A") + 1,
11256                "UpperAlpha", numdelim
11257         end
11258         num = numstr:match("^([a-z])$")
11259         if num then
11260             return string.byte(num) - string.byte("a") + 1,
11261                "LowerAlpha", numdelim
11262         end
11263         num = numstr:match("^([IVXLCDM]+)")
11264         if num then
11265             return roman2number(num), "UpperRoman", numdelim
11266         end
11267         num = numstr:match("^([ivxlc dm]+)")
11268         if num then
11269             return roman2number(string.upper(num)), "LowerRoman", numdelim
11270         end
11271         return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
11272     end
11273
11274     local function fancylist(items,tight,start)
11275         local startnum, numstyle, numdelim
11276         = sniffstyle(start[2][1], start[2][2])
11277         return writer.fancylist(items,tight,
11278                                options.startNumber and startnum or 1,

```

```

11279             numstyle or "Decimal",
11280             numdelim or "Default")
11281     end
11282
11283     local FancyListOfType
11284     = function(start_marker, continuation_marker, delimiter_type)
11285         local enumerator_start
11286         = enumerator(start_marker, continuation_marker,
11287                     delimiter_type)
11288         local enumerator_cont
11289         = enumerator(continuation_marker, continuation_marker,
11290                     delimiter_type)
11291         return Cg(enumerator_start, "listtype")
11292             * (Ct( TightListItem(Cb("listtype"))
11293                 * ((parsers.check_minimal_indent / "")
11294                   * TightListItem(enumerator_cont))^0)
11295             * Cc(true)
11296             * -#((parsers.conditionally_indented_blankline^0 / "")
11297                 * parsers.check_minimal_indent * enumerator_cont)
11298         + Ct( LooseListItem(Cb("listtype"))
11299             * ((parsers.conditionally_indented_blankline^0 / "")
11300               * (parsers.check_minimal_indent / "")
11301               * LooseListItem(enumerator_cont))^0)
11302             * Cc(false)
11303             ) * Ct(Cb("listtype")) / fancylist
11304     end
11305
11306     local FancyList
11307     = join_table_with_func(FancyListOfType, markers_table)
11308
11309     local ListStarter = starter
11310
11311     self.update_rule("OrderedList", FancyList)
11312     self.update_rule("ListStarter", ListStarter)
11313 end
11314 }
11315 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.

```

11316 M.extensions.fenced_code = function(blank_before_code_fence,
11317                                       allow_attributes,
11318                                       allow_raw_blocks)
11319   return {
11320     name = "built-in fenced_code syntax extension",
11321     extend_writer = function(self)
11322       local options = self.options
11323

```

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.

```

11324   function self.fencedCode(s, i, attr)
11325     if not self.is_writing then return "" end
11326     s = s:gsub("\n$", "")
11327     local buf = {}
11328     if attr ~= nil then
11329       table.insert(buf,
11330         {"\\markdownRendererFencedCodeAttributeContextBegin",
11331          self.attributes(attr)})
11332     end
11333     local name = util.cache_verbatim(options.cacheDir, s)
11334     table.insert(buf,
11335       {"\\markdownRendererInputFencedCode{",
11336        name,"}{",self.string(i),"}{",self.infostring(i),"}")})
11337     if attr ~= nil then
11338       table.insert(buf,
11339         "\\markdownRendererFencedCodeAttributeContextEnd{")
11340     end
11341     return buf
11342   end
11343

```

Define `writer->rawBlock` as a function that will transform an input raw block `s` with the raw attribute `attr` to the output format.

```

11344   if allow_raw_blocks then
11345     function self.rawBlock(s, attr)
11346       if not self.is_writing then return "" end
11347       s = s:gsub("\n$", "")
11348       local name = util.cache_verbatim(options.cacheDir, s)
11349       return {"\\markdownRendererInputRawBlock{",
11350        name,"}{", self.string(attr),"}")
11351     end
11352   end
11353   end, extend_reader = function(self)
11354     local parsers = self.parsers

```

```

11355     local writer = self.writer
11356
11357     local function captures_geq_length(_,i,a,b)
11358         return #a >= #b and i
11359     end
11360
11361     local function strip_enclosing_whitespaces(str)
11362         return str:gsub("^%s*(.)%s*$", "%1")
11363     end
11364
11365     local tilde_infostring = Cs(Cs((V("HtmlEntity")
11366                                     + parsers.anyescaped
11367                                     - parsers.newline)^0)
11368                                / strip_enclosing_whitespaces)
11369
11370     local backtick_infostring
11371     = Cs( Cs((V("HtmlEntity")
11372               + ( -(parsers.backslash * parsers.backtick)
11373                   * parsers.anyescaped)
11374                   - parsers.newline
11375                   - parsers.backtick)^0)
11376          / strip_enclosing_whitespaces)
11377
11378     local fenceindent
11379
11380     local function has_trail(indent_table)
11381         return indent_table ~= nil and
11382                indent_table.trail ~= nil and
11383                next(indent_table.trail) ~= nil
11384     end
11385
11386     local function has_indents(indent_table)
11387         return indent_table ~= nil and
11388                indent_table.indents ~= nil and
11389                next(indent_table.indents) ~= nil
11390     end
11391
11392     local function get_last_indent_name(indent_table)
11393         if has_indents(indent_table) then
11394             return indent_table.indents[#indent_table.indents].name
11395         end
11396     end
11397
11398     local count_fenced_start_indent =
11399     function(_, _, indent_table, trail)
11400         local last_indent_name = get_last_indent_name(indent_table)
11401         fenceindent = 0

```

```

11402         if last_indent_name ~= "li" then
11403             fenceindent = #trail
11404         end
11405         return true
11406     end
11407
11408     local fencehead = function(char, infostring)
11409         return Cmt( Cb("indent_info")
11410             * parsers.check_trail, count_fenced_start_indent)
11411             * Cg(char^3, "fencelength")
11412             * parsers.optionalspace
11413             * infostring
11414             * (parsers.newline + parsers.eof)
11415     end
11416
11417     local fencetail = function(char)
11418         return parsers.check_trail_no_rem
11419             * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
11420             * parsers.optionalspace * (parsers.newline + parsers.eof)
11421             + parsers.eof
11422     end
11423
11424     local process_fenced_line =
11425         function(s, i, -- luacheck: ignore s i
11426             indent_table, line_content, is_blank)
11427         local remainder = ""
11428         if has_trail(indent_table) then
11429             remainder = indent_table.trail.internal_remainder
11430         end
11431
11432         if is_blank
11433             and get_last_indent_name(indent_table) == "li" then
11434             remainder = ""
11435         end
11436
11437         local str = remainder .. line_content
11438         local index = 1
11439         local remaining = fenceindent
11440
11441         while true do
11442             local c = str:sub(index, index)
11443             if c == " " and remaining > 0 then
11444                 remaining = remaining - 1
11445                 index = index + 1
11446             elseif c == "\t" and remaining > 3 then
11447                 remaining = remaining - 4
11448                 index = index + 1

```

```

11449         else
11450             break
11451         end
11452     end
11453
11454     return true, str:sub(index)
11455 end
11456
11457 local fencedline = function(char)
11458     return Cmt( Cb("indent_info")
11459         * C(parsers.line - fencetail(char))
11460         * Cc(false), process_fenced_line)
11461 end
11462
11463 local blankfencedline
11464     = Cmt( Cb("indent_info")
11465         * C(parsers.blankline)
11466         * Cc(true), process_fenced_line)
11467
11468 local TildeFencedCode
11469     = fencehead(parsers.tilde, tilde_infostring)
11470     * Cs(( (parsers.check_minimal_blank_indent / "")
11471         * blankfencedline
11472         + ( parsers.check_minimal_indent / "")
11473         * fencedline(parsers.tilde))^0)
11474     * ( (parsers.check_minimal_indent / "")
11475         * fencetail(parsers.tilde) + parsers.succeed)
11476
11477 local BacktickFencedCode
11478     = fencehead(parsers.backtick, backtick_infostring)
11479     * Cs(( (parsers.check_minimal_blank_indent / "")
11480         * blankfencedline
11481         + (parsers.check_minimal_indent / "")
11482         * fencedline(parsers.backtick))^0)
11483     * ( (parsers.check_minimal_indent / "")
11484         * fencetail(parsers.backtick) + parsers.succeed)
11485
11486 local infostring_with_attributes
11487     = Ct(C((parsers.linechar
11488         - ( parsers.optionalspace
11489         * parsers.attributes))^0)
11490         * parsers.optionalspace
11491         * Ct(parsers.attributes))
11492
11493 local FencedCode
11494     = ((TildeFencedCode + BacktickFencedCode)
11495     / function(infostring, code)

```



```

11496         local expanded_code = self.expandtabs(code)
11497
11498         if allow_raw_blocks then
11499             local raw_attr = lpeg.match(parsers.raw_attribute,
11500                                         infostring)
11501             if raw_attr then
11502                 return writer.rawBlock(expanded_code, raw_attr)
11503             end
11504         end
11505
11506         local attr = nil
11507         if allow_attributes then
11508             local match = lpeg.match(infostring_with_attributes,
11509                                     infostring)
11510             if match then
11511                 infostring, attr = table.unpack(match)
11512             end
11513         end
11514         return writer.fencedCode(expanded_code, infostring, attr)
11515     end)
11516
11517     self.insert_pattern("Block after Verbatim",
11518                       FencedCode, "FencedCode")
11519
11520     local fencestart
11521     if blank_before_code_fence then
11522         fencestart = parsers.fail
11523     else
11524         fencestart = fencehead(parsers.backtick, backtick_infostring)
11525                     + fencehead(parsers.tilde, tilde_infostring)
11526     end
11527
11528     self.update_rule("EndlineExceptions", function(previous_pattern)
11529         if previous_pattern == nil then
11530             previous_pattern = parsers.EndlineExceptions
11531         end
11532         return previous_pattern + fencestart
11533     end)
11534
11535     self.add_special_character("`")
11536     self.add_special_character("~")
11537 end
11538 }
11539 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.

```
11540 M.extensions.fenced_divs = function(blank_before_div_fence)
11541   return {
11542     name = "built-in fenced_divs syntax extension",
11543     extend_writer = function(self)
```

Define `writer->div_begin` as a function that will transform the beginning of an input fenced div with attributes `attributes` to the output format.

```
11544     function self.div_begin(attributes)
11545       local start_output
11546       = {"\\markdownRendererFencedDivAttributeContextBegin\n",
11547         self.attributes(attributes)}
11548       local end_output
11549       = {"\\markdownRendererFencedDivAttributeContextEnd{}}"}
11550       return self.push_attributes(
11551         "div", attributes, start_output, end_output)
11552     end
```

Define `writer->div_end` as a function that will produce the end of a fenced div in the output format.

```
11553     function self.div_end()
11554       return self.pop_attributes("div")
11555     end
11556   end, extend_reader = function(self)
11557     local parsers = self.parsers
11558     local writer = self.writer
```

Define basic patterns for matching the opening and the closing tag of a div.

```
11559     local fenced_div_infostring
11560     = C((parsers.linechar
11561       - ( parsers.spacechar^1
11562         * parsers.colon^1))^1)
11563
11564     local fenced_div_begin = parsers.nonindentspace
11565     * parsers.colon^3
11566     * parsers.optionalspace
11567     * fenced_div_infostring
11568     * ( parsers.spacechar^1
11569       * parsers.colon^1)^0
11570     * parsers.optionalspace
11571     * (parsers.newline + parsers.eof)
11572
11573     local fenced_div_end = parsers.nonindentspace
11574     * parsers.colon^3
```

```

11575             * parsers.optionalspace
11576             * (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.

```

11577     self.initialize_named_group("fenced_div_level", "0")
11578     self.initialize_named_group("fenced_div_num_opening_indents")
11579
11580     local function increment_div_level()
11581         local push_indent_table =
11582             function(s, i, indent_table, -- luacheck: ignore s i
11583                 fenced_div_num_opening_indents, fenced_div_level)
11584                 fenced_div_level = tonumber(fenced_div_level) + 1
11585                 local num_opening_indents = 0
11586                 if indent_table.indents ~= nil then
11587                     num_opening_indents = #indent_table.indents
11588                 end
11589                 fenced_div_num_opening_indents[fenced_div_level]
11590                 = num_opening_indents
11591                 return true, fenced_div_num_opening_indents
11592             end
11593
11594     local increment_level =
11595         function(s, i, fenced_div_level) -- luacheck: ignore s i
11596             fenced_div_level = tonumber(fenced_div_level) + 1
11597             return true, tostring(fenced_div_level)
11598         end
11599
11600     return Cg( Cmt( Cb("indent_info")
11601         * Cb("fenced_div_num_opening_indents")
11602         * Cb("fenced_div_level"), push_indent_table)
11603         , "fenced_div_num_opening_indents")
11604         * Cg( Cmt( Cb("fenced_div_level"), increment_level)
11605         , "fenced_div_level")
11606     end
11607
11608     local function decrement_div_level()
11609         local pop_indent_table =
11610             function(s, i, -- luacheck: ignore s i
11611                 fenced_div_indent_table, fenced_div_level)
11612                 fenced_div_level = tonumber(fenced_div_level)
11613                 fenced_div_indent_table[fenced_div_level] = nil
11614                 return true, tostring(fenced_div_level - 1)

```

```

11615         end
11616
11617         return Cg( Cmt( Cb("fenced_div_num_opening_indents")
11618             * Cb("fenced_div_level"), pop_indent_table)
11619             , "fenced_div_level")
11620     end
11621
11622
11623     local non_fenced_div_block
11624     = parsers.check_minimal_indent * V("Block")
11625     - parsers.check_minimal_indent_and_trail * fenced_div_end
11626
11627     local non_fenced_div_paragraph
11628     = parsers.check_minimal_indent * V("Paragraph")
11629     - parsers.check_minimal_indent_and_trail * fenced_div_end
11630
11631     local blank = parsers.minimally_indented_blank
11632
11633     local block_separated = parsers.block_sep_group(blank)
11634     * non_fenced_div_block
11635
11636     local loop_body_pair
11637     = parsers.create_loop_body_pair(block_separated,
11638                                     non_fenced_div_paragraph,
11639                                     parsers.block_sep_group(blank),
11640                                     parsers.par_sep_group(blank))
11641
11642     local content_loop = ( non_fenced_div_block
11643     * loop_body_pair.block^0
11644     + non_fenced_div_paragraph
11645     * block_separated
11646     * loop_body_pair.block^0
11647     + non_fenced_div_paragraph
11648     * loop_body_pair.par^0)
11649     * blank^0
11650
11651     local FencedDiv = fenced_div_begin
11652     / function (infostring)
11653     local attr
11654     = lpeg.match(Ct(parsers.attributes),
11655                 infostring)
11656     if attr == nil then
11657         attr = {"." .. infostring}
11658     end
11659     return attr
11660     end
11661     / writer.div_begin

```

```

11662             * increment_div_level()
11663             * parsers.skipblanklines
11664             * Ct(content_loop)
11665             * parsers.minimally_indented_blank~0
11666             * parsers.check_minimal_indent_and_trail
11667             * fenced_div_end
11668             * decrement_div_level()
11669             * (Cc("") / writer.div_end)
11670
11671         self.insert_pattern("Block after Verbatim",
11672                             FencedDiv, "FencedDiv")
11673
11674         self.add_special_character(":".")
11675

```

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.

```

11676         local function is_inside_div()
11677             local check_div_level =
11678                 function(s, i, fenced_div_level) -- luacheck: ignore s i
11679                     fenced_div_level = tonumber(fenced_div_level)
11680                     return fenced_div_level > 0
11681                 end
11682
11683             return Cmt(Cb("fenced_div_level"), check_div_level)
11684         end
11685
11686         local function check_indent()
11687             local compare_indent =
11688                 function(s, i, indent_table, -- luacheck: ignore s i
11689                     fenced_div_num_opening_indents, fenced_div_level)
11690                     fenced_div_level = tonumber(fenced_div_level)
11691                     local num_current_indents
11692                     = ( indent_table.current_line_indents ~= nil and
11693                         #indent_table.current_line_indents) or 0
11694                     local num_opening_indents
11695                     = fenced_div_num_opening_indents[fenced_div_level]
11696                     return num_current_indents == num_opening_indents
11697                 end
11698
11699             return Cmt( Cb("indent_info")
11700                 * Cb("fenced_div_num_opening_indents")
11701                 * Cb("fenced_div_level"), compare_indent)
11702         end
11703
11704         local fencestart = is_inside_div()

```

```

11705             * fenced_div_end
11706             * check_indent()
11707
11708         if not blank_before_div_fence then
11709             self.update_rule("EndlineExceptions", function(previous_pattern)
11710                 if previous_pattern == nil then
11711                     previous_pattern = parsers.EndlineExceptions
11712                 end
11713                 return previous_pattern + fencestart
11714             end)
11715         end
11716     end
11717 }
11718 end

```

### 3.1.7.8 Header Attributes

The `extensions.header_attributes` function implements the Pandoc header attribute syntax extension.

```

11719 M.extensions.header_attributes = function()
11720     return {
11721         name = "built-in header_attributes syntax extension",
11722         extend_writer = function()
11723         end, extend_reader = function(self)
11724             local parsers = self.parsers
11725             local writer = self.writer
11726
11727             local function strip_atx_end(s)
11728                 return s:gsub("%s+##%s*$", "")
11729             end
11730
11731             local AtxHeading = Cg(parsers.heading_start, "level")
11732                 * parsers.optionalspace
11733                 * (C(((parsers.linechar
11734                     - (parsers.attributes
11735                         * parsers.optionalspace
11736                         * parsers.newline))
11737                     * (parsers.linechar
11738                         - parsers.lbrace)^0)^1)
11739                     / strip_atx_end
11740                     / parsers.parse_heading_text)
11741                 * Cg(Ct(parsers.newline
11742                     + (parsers.attributes
11743                         * parsers.optionalspace
11744                         * parsers.newline)), "attributes")
11745                 * Cb("level")
11746                 * Cb("attributes")

```

```

11747             / writer.heading
11748
11749     local function strip_trailing_spaces(s)
11750         return s:gsub("%s*$","")
11751     end
11752
11753     local heading_line = (parsers.linechar
11754                         - (parsers.attributes
11755                           * parsers.optionalspace
11756                           * parsers.newline))^1
11757                         - parsers.thematic_break_lines
11758
11759     local heading_text
11760     = heading_line
11761     * ( (V("Endline") / "\n")
11762       * (heading_line - parsers.heading_level))^0
11763     * parsers.newline^-1
11764
11765     local SettextHeading
11766     = parsers.freeze_trail * parsers.check_trail_no_rem
11767     * #(heading_text
11768       * (parsers.attributes
11769         * parsers.optionalspace
11770         * parsers.newline)^-1
11771       * parsers.check_minimal_indent
11772       * parsers.check_trail
11773       * parsers.heading_level)
11774     * Cs(heading_text) / strip_trailing_spaces
11775     / parsers.parse_heading_text
11776     * Cg(Ct((parsers.attributes
11777       * parsers.optionalspace
11778       * parsers.newline)^-1), "attributes")
11779     * parsers.check_minimal_indent_and_trail * parsers.heading_level
11780     * Cb("attributes")
11781     * parsers.newline
11782     * parsers.unfreeze_trail
11783     / writer.heading
11784
11785     local Heading = AtxHeading + SettextHeading
11786     self.update_rule("Heading", Heading)
11787 end
11788 }
11789 end

```

### 3.1.7.9 Inline Code Attributes

The `extensions.inline_code_attributes` function implements the Pandoc in-line code attribute syntax extension.

```
11790 M.extensions.inline_code_attributes = function()
11791   return {
11792     name = "built-in inline_code_attributes syntax extension",
11793     extend_writer = function()
11794       end, extend_reader = function(self)
11795         local writer = self.writer
11796
11797         local CodeWithAttributes = parsers.inticks
11798                               * Ct(parsers.attributes)
11799                               / writer.code
11800
11801         self.insert_pattern("Inline before Code",
11802                               CodeWithAttributes,
11803                               "CodeWithAttributes")
11804       end
11805     }
11806 end
```

#### 3.1.7.10 Line Blocks

The `extensions.line_blocks` function implements the Pandoc line block syntax extension.

```
11807 M.extensions.line_blocks = function()
11808   return {
11809     name = "built-in line_blocks syntax extension",
11810     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.

```
11811       function self.lineblock(lines)
11812         if not self.is_writing then return "" end
11813         local buffer = {}
11814         for i = 1, #lines - 1 do
11815           buffer[#buffer + 1] = { lines[i], self.hard_line_break }
11816         end
11817         buffer[#buffer + 1] = lines[#lines]
11818
11819         return {"\\markdownRendererLineBlockBegin\n"
11820               ,buffer,
11821               "\n\\markdownRendererLineBlockEnd "}
11822       end
11823     end, extend_reader = function(self)
11824       local parsers = self.parsers
11825       local writer = self.writer
11826
```



```

11827     local LineBlock
11828         = Ct((Cs(( (parsers.pipe * parsers.space) / ""
11829                     * ((parsers.space)/entities.char_entity("nbsp"))^0
11830                     * parsers.linechar^0 * (parsers.newline/"")
11831                     * (-parsers.pipe
11832                       * (parsers.space^1/" ")
11833                       * parsers.linechar^1
11834                       * (parsers.newline/"")
11835                       )^0
11836                     * (parsers.blankline/"")^0)
11837           / self.parser_functions.parse_inlines)^1)
11838     / writer.lineblock
11839
11840     self.insert_pattern("Block after Blockquote",
11841                       LineBlock, "LineBlock")
11842 end
11843 }
11844 end

```

### 3.1.7.11 Marked text

The `extensions.mark` function implements the Pandoc mark syntax extension.

```

11845 M.extensions.mark = function()
11846     return {
11847         name = "built-in mark syntax extension",
11848         extend_writer = function(self)

```

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

```

11849         function self.mark(s)
11850             if self.flatten_inlines then return s end
11851             return {"\\markdownRendererMark{", s, "}"}
11852         end
11853     end, extend_reader = function(self)
11854         local parsers = self.parsers
11855         local writer = self.writer
11856
11857         local doubleequals = P("==")
11858
11859         local Mark
11860             = parsers.between(V("Inline"), doubleequals, doubleequals)
11861             / function (inlines) return writer.mark(inlines) end
11862
11863         self.add_special_character("=")
11864         self.insert_pattern("Inline before LinkAndEmph",
11865                           Mark, "Mark")
11866     end
11867 }

```

11868 end

### 3.1.7.12 Link Attributes

The `extensions.link_attributes` function implements the Pandoc link attribute syntax extension.

```
11869 M.extensions.link_attributes = function()
11870   return {
11871     name = "built-in link_attributes syntax extension",
11872     extend_writer = function()
11873     end, extend_reader = function(self)
11874       local parsers = self.parsers
11875       local options = self.options
11876
```

The following patterns define link reference definitions with attributes.

```
11877     local define_reference_parser
11878     = (parsers.check_trail / "")
11879     * parsers.link_label
11880     * parsers.colon
11881     * parsers.spnlc * parsers.url
11882     * ( parsers.spnlc_sep * parsers.title
11883     * (parsers.spnlc * Ct(parsers.attributes))
11884     * parsers.only_blank
11885     + parsers.spnlc_sep * parsers.title * parsers.only_blank
11886     + Cc("") * (parsers.spnlc * Ct(parsers.attributes))
11887     * parsers.only_blank
11888     + Cc("") * parsers.only_blank)
11889
11890     local ReferenceWithAttributes = define_reference_parser
11891     / self.register_link
11892
11893     self.update_rule("Reference", ReferenceWithAttributes)
11894
```

The following patterns define direct and indirect links with attributes.

```
11895
11896     local LinkWithAttributesAndEmph
11897     = Ct(parsers.link_and_emph_table * Cg(Cc(true),
11898     "match_link_attributes"))
11899     / self.defer_link_and_emphasis_processing
11900
11901     self.update_rule("LinkAndEmph", LinkWithAttributesAndEmph)
11902
```

The following patterns define autolinks with attributes.

```
11903     local AutoLinkUrlWithAttributes
11904     = parsers.auto_link_url
```

```

11905             * Ct(parsers.attributes)
11906             / self.auto_link_url
11907
11908         self.insert_pattern("Inline before AutoLinkUrl",
11909                             AutoLinkUrlWithAttributes,
11910                             "AutoLinkUrlWithAttributes")
11911
11912         local AutoLinkEmailWithAttributes
11913             = parsers.auto_link_email
11914             * Ct(parsers.attributes)
11915             / self.auto_link_email
11916
11917         self.insert_pattern("Inline before AutoLinkEmail",
11918                             AutoLinkEmailWithAttributes,
11919                             "AutoLinkEmailWithAttributes")
11920
11921         if options.relativeReferences then
11922
11923             local AutoLinkRelativeReferenceWithAttributes
11924                 = parsers.auto_link_relative_reference
11925                 * Ct(parsers.attributes)
11926                 / self.auto_link_url
11927
11928             self.insert_pattern(
11929                 "Inline before AutoLinkRelativeReference",
11930                 AutoLinkRelativeReferenceWithAttributes,
11931                 "AutoLinkRelativeReferenceWithAttributes")
11932
11933         end
11934
11935     end
11936 }
11937 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.

```

11938 M.extensions.notes = function(notes, inline_notes)
11939     assert(notes or inline_notes)
11940     return {
11941         name = "built-in notes syntax extension",
11942         extend_writer = function(self)

```

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

```

11943     function self.note(s)
11944         if self.flatten_inlines then return "" end
11945         return {"\\markdownRendererNote{" ,s,""} }
11946     end
11947 end, extend_reader = function(self)
11948     local parsers = self.parsers
11949     local writer = self.writer
11950
11951     local rawnotes = parsers.rawnotes
11952
11953     if inline_notes then
11954         local InlineNote
11955         = parsers.circumflex
11956         * ( parsers.link_label
11957           / self.parser_functions.parse_inlines_no_inline_note)
11958         / writer.note
11959
11960         self.insert_pattern("Inline after LinkAndEmph",
11961                             InlineNote, "InlineNote")
11962     end
11963     if notes then
11964         local function strip_first_char(s)
11965             return s:sub(2)
11966         end
11967
11968         local RawNoteRef
11969             = #(parsers.lbracket * parsers.circumflex)
11970             * parsers.link_label / strip_first_char
11971
11972         -- like indirect_link
11973         local function lookup_note(ref)
11974             return writer.defer_call(function()
11975                 local found = rawnotes[self.normalize_tag(ref)]
11976                 if found then
11977                     return writer.note(
11978                         self.parser_functions.parse_blocks_nested(found))
11979                 else
11980                     return {"[",
11981                             self.parser_functions.parse_inlines("^" .. ref), "]" }
11982                 end
11983             end)
11984         end
11985
11986         local function register_note(ref,rawnote)
11987             local normalized_tag = self.normalize_tag(ref)

```

```

11988         if rawnotes[normalized_tag] == nil then
11989             rawnotes[normalized_tag] = rawnote
11990         end
11991         return ""
11992     end
11993
11994     local NoteRef = RawNoteRef / lookup_note
11995
11996     local optionally_indented_line
11997         = parsers.check_optional_indent_and_any_trail * parsers.line
11998
11999     local blank
12000         = parsers.check_optional_blank_indent_and_any_trail
12001         * parsers.optionalspace * parsers.newline
12002
12003     local chunk
12004         = Cs(parsers.line
12005             * (optionally_indented_line - blank)^0)
12006
12007     local indented_blocks = function(bl)
12008         return Cs( bl
12009             * ( blank^1 * (parsers.check_optional_indent / "")
12010             * parsers.check_code_trail
12011             * -parsers.blankline * bl)^0)
12012     end
12013
12014     local NoteBlock
12015         = parsers.check_trail_no_rem
12016         * RawNoteRef * parsers.colon
12017         * parsers.spnlc * indented_blocks(chunk)
12018         / register_note
12019
12020     self.update_rule("Reference", function(previous_pattern)
12021         if previous_pattern == nil then
12022             previous_pattern = parsers.Reference
12023         end
12024         return NoteBlock + previous_pattern
12025     end)
12026
12027     self.insert_pattern("Inline before LinkAndEmph",
12028         NoteRef, "NoteRef")
12029 end
12030
12031 self.add_special_character("^")
12032 end
12033 }
12034 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.

```
12035 M.extensions.pipe_tables = function(table_captions, table_attributes)
12036
12037   local function make_pipe_table_rectangular(rows)
12038     local num_columns = #rows[2]
12039     local rectangular_rows = {}
12040     for i = 1, #rows do
12041       local row = rows[i]
12042       local rectangular_row = {}
12043       for j = 1, num_columns do
12044         rectangular_row[j] = row[j] or ""
12045       end
12046       table.insert(rectangular_rows, rectangular_row)
12047     end
12048     return rectangular_rows
12049   end
12050
12051   local function pipe_table_row(allow_empty_first_column
12052                                , nonempty_column
12053                                , column_separator
12054                                , column)
12055     local row_beginning
12056     if allow_empty_first_column then
12057       row_beginning = -- empty first column
12058                       #(parsers.spacechar^4
12059                        * column_separator)
12060                       * parsers.optionalspace
12061                       * column
12062                       * parsers.optionalspace
12063                       -- non-empty first column
12064                       + parsers.nonindentSPACE
12065                       * nonempty_column~-1
12066                       * parsers.optionalspace
12067     else
12068       row_beginning = parsers.nonindentSPACE
12069                       * nonempty_column~-1
12070                       * parsers.optionalspace
12071     end
12072
12073   return Ct(row_beginning
```

```

12074         * (-- single column with no leading pipes
12075         #(column_separator
12076         * parsers.optionalspace
12077         * parsers.newline)
12078         * column_separator
12079         * parsers.optionalspace
12080         -- single column with leading pipes or
12081         -- more than a single column
12082         + (column_separator
12083         * parsers.optionalspace
12084         * column
12085         * parsers.optionalspace)^1
12086         * (column_separator
12087         * parsers.optionalspace)^-1))
12088     end
12089
12090     return {
12091         name = "built-in pipe_tables syntax extension",
12092         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.

```

12093         function self.table(rows, caption, attributes)
12094             if not self.is_writing then return "" end
12095             local buffer = {}
12096             if attributes ~= nil then
12097                 table.insert(buffer,
12098                     "\\markdownRendererTableAttributeContextBegin\n")
12099                 table.insert(buffer, self.attributes(attributes))
12100             end
12101             table.insert(buffer,
12102                 {"\\markdownRendererTable{",
12103                 caption or "", "}{" , #rows - 1, "}{" ,
12104                 #rows[1], "}"}))
12105             local temp = rows[2] -- put alignments on the first row
12106             rows[2] = rows[1]
12107             rows[1] = temp
12108             for i, row in ipairs(rows) do
12109                 table.insert(buffer, "{")
12110                 for _, column in ipairs(row) do
12111                     if i > 1 then -- do not use braces for alignments
12112                         table.insert(buffer, "{")
12113                     end
12114                     table.insert(buffer, column)
12115                     if i > 1 then
12116                         table.insert(buffer, "}")
12117                     end

```

```

12118         end
12119         table.insert(buffer, "}")
12120     end
12121     if attributes ~= nil then
12122         table.insert(buffer,
12123             "\\markdownRendererTableAttributeContextEnd{}")
12124     end
12125     return buffer
12126 end
12127 end, extend_reader = function(self)
12128     local parsers = self.parsers
12129     local writer = self.writer
12130
12131     local table_hline_separator = parsers.pipe + parsers.plus
12132
12133     local table_hline_column = (parsers.dash
12134         - #(parsers.dash
12135             * (parsers.spacechar
12136                 + table_hline_separator
12137                 + parsers.newline)))^1
12138         * (parsers.colon * Cc("r")
12139             + parsers.dash * Cc("d"))
12140         + parsers.colon
12141         * (parsers.dash
12142             - #(parsers.dash
12143                 * (parsers.spacechar
12144                     + table_hline_separator
12145                     + parsers.newline)))^1
12146         * (parsers.colon * Cc("c")
12147             + parsers.dash * Cc("l"))
12148
12149     local table_hline = pipe_table_row(false
12150         , table_hline_column
12151         , table_hline_separator
12152         , table_hline_column)
12153
12154     local table_caption_beginning
12155     = ( parsers.check_minimal_blank_indent_and_any_trail_no_rem
12156         * parsers.optionalspace * parsers.newline)^0
12157         * parsers.check_minimal_indent_and_trail
12158         * (P("Table")^-1 * parsers.colon)
12159         * parsers.optionalspace
12160
12161     local function strip_trailing_spaces(s)
12162         return s:gsub("%s*$", "")
12163     end
12164

```



```

12165     local table_row
12166     = pipe_table_row(true
12167         , (C((parsers.linechar - parsers.pipe)^1)
12168           / strip_trailing_spaces
12169           / self.parser_functions.parse_inlines)
12170         , parsers.pipe
12171         , (C((parsers.linechar - parsers.pipe)^0)
12172           / strip_trailing_spaces
12173           / self.parser_functions.parse_inlines))
12174
12175     local table_caption
12176     if table_captions then
12177         table_caption = #table_caption_beginning
12178             * table_caption_beginning
12179         if table_attributes then
12180             table_caption = table_caption
12181                 * (C(((( parsers.linechar
12182                   - (parsers.attributes
12183                     * parsers.optionalspace
12184                     * parsers.newline
12185                     * -( parsers.optionalspace
12186                       * parsers.linechar))))
12187                   + ( parsers.newline
12188                     * #( parsers.optionalspace
12189                       * parsers.linechar)
12190                     * C(parsers.optionalspace)
12191                       / writer.space))
12192                   * (parsers.linechar
12193                     - parsers.lbrace)^0)^1)
12194                   / self.parser_functions.parse_inlines)
12195                 * (parsers.newline
12196                   + ( Ct(parsers.attributes)
12197                     * parsers.optionalspace
12198                     * parsers.newline))
12199         else
12200             table_caption = table_caption
12201                 * C(( parsers.linechar
12202                   + ( parsers.newline
12203                     * #( parsers.optionalspace
12204                       * parsers.linechar)
12205                     * C(parsers.optionalspace)
12206                       / writer.space))^1)
12207                   / self.parser_functions.parse_inlines
12208                 * parsers.newline
12209         end
12210     else
12211         table_caption = parsers.fail

```

```

12212     end
12213
12214     local PipeTable
12215     = Ct( table_row * parsers.newline
12216         * (parsers.check_minimal_indent_and_trail / {})
12217         * table_hline * parsers.newline
12218         * ( (parsers.check_minimal_indent / {})
12219           * table_row * parsers.newline)^0)
12220     / make_pipe_table_rectangular
12221     * table_caption^-1
12222     / writer.table
12223
12224     self.insert_pattern("Block after Blockquote",
12225                       PipeTable, "PipeTable")
12226   end
12227 }
12228 end

```

### 3.1.7.15 Raw Attributes

The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```

12229 M.extensions.raw_inline = function()
12230   return {
12231     name = "built-in raw_inline syntax extension",
12232     extend_writer = function(self)
12233       local options = self.options
12234

```

Define `writer->rawInline` as a function that will transform an input inline raw span `s` with the raw attribute `attr` to the output format.

```

12235     function self.rawInline(s, attr)
12236       if not self.is_writing then return "" end
12237       if self.flatten_inlines then return s end
12238       local name = util.cache_verbatim(options.cacheDir, s)
12239       return {"\\markdownRendererInputRawInline{",
12240             name,"}{" , self.string(attr),"}" }
12241     end
12242   end, extend_reader = function(self)
12243     local writer = self.writer
12244
12245     local RawInline = parsers.inticks
12246                       * parsers.raw_attribute
12247                       / writer.rawInline
12248
12249     self.insert_pattern("Inline before Code",
12250                       RawInline, "RawInline")
12251   end

```

```

12252   }
12253 end

```

### 3.1.7.16 Strike-Through

The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```

12254 M.extensions.strike_through = function()
12255   return {
12256     name = "built-in strike_through syntax extension",
12257     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.

```

12258       function self.strike_through(s)
12259         if self.flatten_inlines then return s end
12260         return {"\\markdownRendererStrikeThrough{" ,s,"}"}
12261       end
12262     end, extend_reader = function(self)
12263       local parsers = self.parsers
12264       local writer = self.writer
12265
12266       local StrikeThrough = (
12267         parsers.between(parsers.Inline, parsers.doubletildes,
12268           parsers.doubletildes)
12269       ) / writer.strike_through
12270
12271       self.insert_pattern("Inline after LinkAndEmph",
12272         StrikeThrough, "StrikeThrough")
12273
12274       self.add_special_character("~")
12275     end
12276   }
12277 end

```

### 3.1.7.17 Subscripts

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

```

12278 M.extensions.subscripts = function()
12279   return {
12280     name = "built-in subscripts syntax extension",
12281     extend_writer = function(self)

```

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

```

12282       function self.subscript(s)
12283         if self.flatten_inlines then return s end

```

```

12284         return {"\\markdownRendererSubscript{"s,"}"}
12285     end
12286 end, extend_reader = function(self)
12287     local parsers = self.parsers
12288     local writer = self.writer
12289
12290     local Subscript = (
12291         parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
12292     ) / writer.subscript
12293
12294     self.insert_pattern("Inline after LinkAndEmph",
12295                         Subscript, "Subscript")
12296
12297     self.add_special_character("~")
12298 end
12299 }
12300 end

```

### 3.1.7.18 Superscripts

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

```

12301 M.extensions.superscripts = function()
12302     return {
12303         name = "built-in superscripts syntax extension",
12304         extend_writer = function(self)

```

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

```

12305         function self.superscript(s)
12306             if self.flatten_inlines then return s end
12307             return {"\\markdownRendererSuperscript{"s,"}"}
12308         end
12309     end, extend_reader = function(self)
12310         local parsers = self.parsers
12311         local writer = self.writer
12312
12313         local Superscript = (
12314             parsers.between(parsers.Str, parsers.circumflex,
12315                             parsers.circumflex)
12316         ) / writer.superscript
12317
12318         self.insert_pattern("Inline after LinkAndEmph",
12319                             Superscript, "Superscript")
12320
12321         self.add_special_character("^")
12322     end
12323 }

```

12324 end

### 3.1.7.19 T<sub>E</sub>X Math

The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```
12325 M.extensions.tex_math = function(tex_math_dollars,  
12326                                     tex_math_single_backslash,  
12327                                     tex_math_double_backslash)  
12328   return {  
12329     name = "built-in tex_math syntax extension",  
12330     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.

```
12331       function self.display_math(s)  
12332         if self.flatten_inlines then return s end  
12333         return {"\\markdownRendererDisplayMath{",self.math(s),"}"}  
12334       end
```

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

```
12335       function self.inline_math(s)  
12336         if self.flatten_inlines then return s end  
12337         return {"\\markdownRendererInlineMath{",self.math(s),"}"}  
12338       end  
12339     end, extend_reader = function(self)  
12340       local parsers = self.parsers  
12341       local writer = self.writer  
12342  
12343       local function between(p, starter, ender)  
12344         return (starter * Cs(p * (p - ender)^0) * ender)  
12345       end  
12346  
12347       local function strip_preceding_whitespaces(str)  
12348         return str:gsub("^%s*(.-)$", "%1")  
12349       end  
12350  
12351       local allowed_before_closing  
12352       = B( parsers.backslash * parsers.any  
12353         + parsers.any * (parsers.any - parsers.backslash))  
12354  
12355       local allowed_before_closing_no_space  
12356       = B( parsers.backslash * parsers.any  
12357         + parsers.any * (parsers.nonspacechar - parsers.backslash))  
12358     end
```

The following patterns implement the Pandoc dollar math syntax extension.

```

12359     local dollar_math_content
12360         = (parsers.newline * (parsers.check_optional_indent / ""))
12361         + parsers.backslash^-1
12362         * parsers.linechar)
12363         - parsers.blankline^2
12364         - parsers.dollar
12365
12366     local inline_math_opening_dollars = parsers.dollar
12367                                         * #(parsers.nonspacechar)
12368
12369     local inline_math_closing_dollars
12370         = allowed_before_closing_no_space
12371         * parsers.dollar
12372         * -#(parsers.digit)
12373
12374     local inline_math_dollars = between(Cs( dollar_math_content),
12375                                         inline_math_opening_dollars,
12376                                         inline_math_closing_dollars)
12377
12378     local display_math_opening_dollars = parsers.dollar
12379                                         * parsers.dollar
12380
12381     local display_math_closing_dollars = parsers.dollar
12382                                         * parsers.dollar
12383
12384     local display_math_dollars = between(Cs( dollar_math_content),
12385                                         display_math_opening_dollars,
12386                                         display_math_closing_dollars)

```

The following patterns implement the Pandoc single and double backslash math syntax extensions.

```

12387     local backslash_math_content
12388         = (parsers.newline * (parsers.check_optional_indent / ""))
12389         + parsers.linechar)
12390         - parsers.blankline^2

```

The following patterns implement the Pandoc double backslash math syntax extension.

```

12391     local inline_math_opening_double = parsers.backslash
12392                                         * parsers.backslash
12393                                         * parsers.lparent
12394
12395     local inline_math_closing_double = allowed_before_closing
12396                                         * parsers.spacechar^0
12397                                         * parsers.backslash
12398                                         * parsers.backslash
12399                                         * parsers.rparent
12400

```

```

12401     local inline_math_double = between(Cs( backslash_math_content),
12402                                         inline_math_opening_double,
12403                                         inline_math_closing_double)
12404                                         / strip_preceding_whitespace
12405
12406     local display_math_opening_double = parsers.backslash
12407                                         * parsers.backslash
12408                                         * parsers.lbracket
12409
12410     local display_math_closing_double = allowed_before_closing
12411                                         * parsers.spacechar^0
12412                                         * parsers.backslash
12413                                         * parsers.backslash
12414                                         * parsers.rbracket
12415
12416     local display_math_double = between(Cs( backslash_math_content),
12417                                         display_math_opening_double,
12418                                         display_math_closing_double)
12419                                         / strip_preceding_whitespace

```

The following patterns implement the Pandoc single backslash math syntax extension.

```

12420     local inline_math_opening_single = parsers.backslash
12421                                         * parsers.lparent
12422
12423     local inline_math_closing_single = allowed_before_closing
12424                                         * parsers.spacechar^0
12425                                         * parsers.backslash
12426                                         * parsers.rparent
12427
12428     local inline_math_single = between(Cs( backslash_math_content),
12429                                         inline_math_opening_single,
12430                                         inline_math_closing_single)
12431                                         / strip_preceding_whitespace
12432
12433     local display_math_opening_single = parsers.backslash
12434                                         * parsers.lbracket
12435
12436     local display_math_closing_single = allowed_before_closing
12437                                         * parsers.spacechar^0
12438                                         * parsers.backslash
12439                                         * parsers.rbracket
12440
12441     local display_math_single = between(Cs( backslash_math_content),
12442                                         display_math_opening_single,
12443                                         display_math_closing_single)
12444                                         / strip_preceding_whitespace
12445
12446     local display_math = parsers.fail

```

```

12447
12448     local inline_math = parsers.fail
12449
12450     if tex_math_dollars then
12451         display_math = display_math + display_math_dollars
12452         inline_math = inline_math + inline_math_dollars
12453     end
12454
12455     if tex_math_double_backslash then
12456         display_math = display_math + display_math_double
12457         inline_math = inline_math + inline_math_double
12458     end
12459
12460     if tex_math_single_backslash then
12461         display_math = display_math + display_math_single
12462         inline_math = inline_math + inline_math_single
12463     end
12464
12465     local TexMath = display_math / writer.display_math
12466                   + inline_math / writer.inline_math
12467
12468     self.insert_pattern("Inline after LinkAndEmph",
12469                       TexMath, "TexMath")
12470
12471     if tex_math_dollars then
12472         self.add_special_character("$")
12473     end
12474
12475     if tex_math_single_backslash or tex_math_double_backslash then
12476         self.add_special_character("\\")
12477         self.add_special_character("[")
12478         self.add_special_character("]")
12479         self.add_special_character("(")
12480         self.add_special_character("(")
12481     end
12482 end
12483 }
12484 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.

```

12485 M.extensions.jekyll_data = function(expect_jekyll_data,
12486                                         ensure_jekyll_data)
12487   return {
12488     name = "built-in jekyll_data syntax extension",
12489     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.

```

12490     function self.jekyllData(d, t, p)
12491       if not self.is_writing then return "" end
12492
12493       local buf = {}
12494
12495       local keys = {}
12496       for k, _ in pairs(d) do
12497         table.insert(keys, k)
12498       end

```

For reproducibility, sort the keys. For mixed string-and-numeric keys, sort numeric keys before string keys.

```

12499       table.sort(keys, function(first, second)
12500         if type(first) ~= type(second) then
12501           return type(first) < type(second)
12502         else
12503           return first < second
12504         end
12505       end)
12506
12507       if not p then
12508         table.insert(buf, "\\markdownRendererJekyllDataBegin")
12509       end
12510
12511       local is_sequence = false
12512       if #d > 0 and #d == #keys then
12513         for i=1, #d do
12514           if d[i] == nil then
12515             goto not_a_sequence
12516           end
12517         end
12518         is_sequence = true
12519       end

```

```

12520      ::not_a_sequence::
12521
12522      if is_sequence then
12523          table.insert(buf,
12524              "\\markdownRendererJekyllDataSequenceBegin{")
12525          table.insert(buf, self.identifier(p or "null"))
12526          table.insert(buf, "{")
12527          table.insert(buf, #keys)
12528          table.insert(buf, "}")
12529      else
12530          table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
12531          table.insert(buf, self.identifier(p or "null"))
12532          table.insert(buf, "{")
12533          table.insert(buf, #keys)
12534          table.insert(buf, "}")
12535      end
12536
12537      for _, k in ipairs(keys) do
12538          local v = d[k]
12539          local typ = type(v)
12540          k = tostring(k or "null")
12541          if typ == "table" and next(v) ~= nil then
12542              table.insert(
12543                  buf,
12544                  self.jekyllData(v, t, k)
12545              )
12546          else
12547              k = self.identifier(k)
12548              v = tostring(v)
12549              if typ == "boolean" then
12550                  table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
12551                  table.insert(buf, k)
12552                  table.insert(buf, "{")
12553                  table.insert(buf, v)
12554                  table.insert(buf, "}")
12555              elseif typ == "number" then
12556                  table.insert(buf, "\\markdownRendererJekyllDataNumber{")
12557                  table.insert(buf, k)
12558                  table.insert(buf, "{")
12559                  table.insert(buf, v)
12560                  table.insert(buf, "}")
12561              elseif typ == "string" then
12562                  table.insert(buf,
12563                      "\\markdownRendererJekyllDataProgrammaticString{")
12564                  table.insert(buf, k)
12565                  table.insert(buf, "{")
12566                  table.insert(buf, self.identifier(v))

```

```

12567         table.insert(buf, "}")
12568         table.insert(buf,
12569             "\\markdownRendererJekyllDataTypographicString{")
12570         table.insert(buf, k)
12571         table.insert(buf, "{")
12572         table.insert(buf, t(v))
12573         table.insert(buf, "}")
12574     elseif typ == "table" then
12575         table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
12576         table.insert(buf, k)
12577         table.insert(buf, "}")
12578     else
12579         local error = self.error(format(
12580             "Unexpected type %s for value of "
12581             .. "YAML key %s.", typ, k))
12582         table.insert(buf, error)
12583     end
12584 end
12585 end
12586
12587 if is_sequence then
12588     table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
12589 else
12590     table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
12591 end
12592
12593 if not p then
12594     table.insert(buf, "\\markdownRendererJekyllDataEnd")
12595 end
12596
12597 return buf
12598 end
12599 end, extend_reader = function(self)
12600     local parsers = self.parsers
12601     local writer = self.writer
12602
12603     local JekyllData
12604     = Cmt( C((parsers.line - P("---") - P("..."))^0)
12605         , function(s, i, text) -- luacheck: ignore s i
12606             local data
12607             local ran_ok, _ = pcall(function()
12608                 local tinyyaml = require("tinyyaml")
12609                 data = tinyyaml.parse(text, {timestamps=false})
12610             end)
12611             if ran_ok and data ~= nil then
12612                 return true, writer.jekyllData(data, function(s)
12613                     return self.parser_functions.parse_blocks_nested(s)

```

```

12614         end, nil)
12615     else
12616         return false
12617     end
12618 end
12619 )
12620
12621 local UnexpectedJekyllData
12622 = P("----")
12623 * parsers.blankline / 0
12624 -- if followed by blank, it's thematic break
12625 * #(-parsers.blankline)
12626 * JekyllData
12627 * (P("----") + P("..."))
12628
12629 local ExpectedJekyllData
12630 = ( P("----")
12631 * parsers.blankline / 0
12632 -- if followed by blank, it's thematic break
12633 * #(-parsers.blankline)
12634 )^-1
12635 * JekyllData
12636 * (P("----") + P("..."))^-1
12637
12638 if ensure_jekyll_data then
12639     ExpectedJekyllData = ExpectedJekyllData
12640 * parsers.eof
12641 else
12642     ExpectedJekyllData = ( ExpectedJekyllData
12643 * (V("Blank")^0 / writer.interblocksep)
12644 )^-1
12645 end
12646
12647 self.insert_pattern("Block before Blockquote",
12648     UnexpectedJekyllData, "UnexpectedJekyllData")
12649 if expect_jekyll_data then
12650     self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
12651 end
12652 end
12653 }
12654 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 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`.

```
12655 function M.new(options)
```

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

```
12656 options = options or {}
12657 setmetatable(options, { __index = function (_, key)
12658     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.

```
12659 local parser_convert = nil
12660 return function(input, include_flat_output)
12661     local function convert(input)
12662         if parser_convert == nil then
```

Lazy-load `markdown-parser.lua` and check that it originates from the same version of the Markdown package.

```
12663         local parser = require("markdown-parser")
12664         if metadata.version ~= parser.metadata.version then
12665             warn("markdown.lua " .. metadata.version .. " used with " ..
12666                 "markdown-parser.lua " .. parser.metadata.version .. ".")
12667         end
12668         parser_convert = parser.new(options)
12669     end
12670     return parser_convert(input)
12671 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).

```
12672 local raw_output, flat_output
12673 if options.eagerCache or options.finalizeCache then
12674     local salt = util.salt(options)
12675     local name, result = util.cache(options.cacheDir, input, salt,
12676                                     convert, ".md.tex")
12677     raw_output = [[\input{}}] .. name .. [[}\relax]]
12678     flat_output = function()
12679         if result == nil then
12680             local input_file = assert(io.open(name, "r"),
12681                                     [[Could not open file ]] .. name .. [[ for reading]])
12682             result = assert(input_file:read("*a"))
12683             assert(input_file:close())
12684         end
```

```

12685         return result
12686     end

```

Otherwise, return the result of the conversion directly.

```

12687     else
12688         raw_output = convert(input)
12689         flat_output = function()
12690             return raw_output
12691         end
12692     end

```

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

```

12693     if options.finalizeCache then
12694         local file, mode
12695         if options.frozenCacheCounter > 0 then
12696             mode = "a"
12697         else
12698             mode = "w"
12699         end
12700         file = assert(io.open(options.frozenCacheFileName, mode),
12701             [[Could not open file ]] .. options.frozenCacheFileName
12702             .. [[ for writing]])
12703         assert(file:write(
12704             [[\expandafter\global\expandafter\def\csname ]]
12705             .. [[markdownFrozenCache]] .. options.frozenCacheCounter
12706             .. [[\endcsname{}]] .. raw_output .. [[]]] .. "\n"))
12707         assert(file:close())
12708     end

```

Besides the canonical output of the conversion, which may contain cached files behind `\input`, also return a function that always produces a flat output regardless of caching as the second return value.

```

12709     if include_flat_output then
12710         return raw_output, flat_output
12711     else
12712         return raw_output
12713     end
12714 end
12715 end

```

The `new` function from file `markdown-parser.lua` returns a conversion function that takes a markdown string and turns it into a plain T<sub>E</sub>X output. See Section 2.1.1.

```

12716 function M.new(options)

```

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

```

12717     options = options or {}

```

```

12718 setmetatable(options, { __index = function (_, key)
12719     return defaultOptions[key] end })

```

If the singleton cache contains a conversion function for the same `options`, reuse it.

```

12720 if options.singletonCache and singletonCache.convert then
12721     for k, v in pairs(defaultOptions) do
12722         if type(v) == "table" then
12723             for i = 1, math.max(#singletonCache.options[k], #options[k]) do
12724                 if singletonCache.options[k][i] ~= options[k][i] then
12725                     goto miss
12726                 end
12727             end

```

The `cacheDir` option is disregarded.

```

12728         elseif k ~= "cacheDir"
12729             and singletonCache.options[k] ~= options[k] then
12730             goto miss
12731         end
12732     end
12733     return singletonCache.convert
12734 end
12735 ::miss::

```

Apply built-in syntax extensions based on `options`.

```

12736 local extensions = {}
12737
12738 if options.bracketedSpans then
12739     local bracketed_spans_extension = M.extensions.bracketed_spans()
12740     table.insert(extensions, bracketed_spans_extension)
12741 end
12742
12743 if options.contentBlocks then
12744     local content_blocks_extension = M.extensions.content_blocks(
12745         options.contentBlocksLanguageMap)
12746     table.insert(extensions, content_blocks_extension)
12747 end
12748
12749 if options.definitionLists then
12750     local definition_lists_extension = M.extensions.definition_lists(
12751         options.tightLists)
12752     table.insert(extensions, definition_lists_extension)
12753 end
12754
12755 if options.fencedCode then
12756     local fenced_code_extension = M.extensions.fenced_code(
12757         options.blankBeforeCodeFence,
12758         options.fencedCodeAttributes,
12759         options.rawAttribute)

```

```

12760     table.insert(extensions, fenced_code_extension)
12761 end
12762
12763 if options.fencedDivs then
12764     local fenced_div_extension = M.extensions.fenced_divs(
12765         options.blankBeforeDivFence)
12766     table.insert(extensions, fenced_div_extension)
12767 end
12768
12769 if options.headerAttributes then
12770     local header_attributes_extension = M.extensions.header_attributes()
12771     table.insert(extensions, header_attributes_extension)
12772 end
12773
12774 if options.inlineCodeAttributes then
12775     local inline_code_attributes_extension =
12776         M.extensions.inline_code_attributes()
12777     table.insert(extensions, inline_code_attributes_extension)
12778 end
12779
12780 if options.jekyllData then
12781     local jekyll_data_extension = M.extensions.jekyll_data(
12782         options.expectJekyllData, options.ensureJekyllData)
12783     table.insert(extensions, jekyll_data_extension)
12784 end
12785
12786 if options.linkAttributes then
12787     local link_attributes_extension =
12788         M.extensions.link_attributes()
12789     table.insert(extensions, link_attributes_extension)
12790 end
12791
12792 if options.lineBlocks then
12793     local line_block_extension = M.extensions.line_blocks()
12794     table.insert(extensions, line_block_extension)
12795 end
12796
12797 if options.mark then
12798     local mark_extension = M.extensions.mark()
12799     table.insert(extensions, mark_extension)
12800 end
12801
12802 if options.pipeTables then
12803     local pipe_tables_extension = M.extensions.pipe_tables(
12804         options.tableCaptions, options.tableAttributes)
12805     table.insert(extensions, pipe_tables_extension)
12806 end

```



```

12807
12808   if options.rawAttribute then
12809       local raw_inline_extension = M.extensions.raw_inline()
12810       table.insert(extensions, raw_inline_extension)
12811   end
12812
12813   if options.strikeThrough then
12814       local strike_through_extension = M.extensions.strike_through()
12815       table.insert(extensions, strike_through_extension)
12816   end
12817
12818   if options.subscripts then
12819       local subscript_extension = M.extensions.subscripts()
12820       table.insert(extensions, subscript_extension)
12821   end
12822
12823   if options.superscripts then
12824       local superscript_extension = M.extensions.superscripts()
12825       table.insert(extensions, superscript_extension)
12826   end
12827
12828   if options.texMathDollars or
12829       options.texMathSingleBackslash or
12830       options.texMathDoubleBackslash then
12831       local tex_math_extension = M.extensions.tex_math(
12832           options.texMathDollars,
12833           options.texMathSingleBackslash,
12834           options.texMathDoubleBackslash)
12835       table.insert(extensions, tex_math_extension)
12836   end
12837
12838   if options.notes or options.inlineNotes then
12839       local notes_extension = M.extensions.notes(
12840           options.notes, options.inlineNotes)
12841       table.insert(extensions, notes_extension)
12842   end
12843
12844   if options.citations then
12845       local citations_extension
12846       = M.extensions.citations(options.citationNbsps)
12847       table.insert(extensions, citations_extension)
12848   end
12849
12850   if options.fancyLists then
12851       local fancy_lists_extension = M.extensions.fancy_lists()
12852       table.insert(extensions, fancy_lists_extension)
12853   end

```

Apply user-defined syntax extensions based on `options.extensions`.

```
12854   for _, user_extension_filename in ipairs(options.extensions) do
12855       local user_extension = (function(filename)
```

First, load and compile the contents of the user-defined syntax extension.

```
12856         local pathname = assert(kpse.find_file(filename),
12857             [[Could not locate user-defined syntax extension "]]
12858             .. filename)
12859         local input_file = assert(io.open(pathname, "r"),
12860             [[Could not open user-defined syntax extension "]]
12861             .. pathname .. [[" for reading]])
12862         local input = assert(input_file:read("*a"))
12863         assert(input_file:close())
12864         local user_extension, err = load([[
12865             local sandbox = {}
12866             setmetatable(sandbox, {__index = _G})
12867             _ENV = sandbox
12868         ]] .. input)()
12869         assert(user_extension,
12870             [[Failed to compile user-defined syntax extension "]]
12871             .. pathname .. [[:]] .. (err or [[]]))
```

Then, validate the user-defined syntax extension.

```
12872         assert(user_extension.api_version ~= nil,
12873             [[User-defined syntax extension "]] .. pathname
12874             .. [[" does not specify mandatory field "api_version"]])
12875         assert(type(user_extension.api_version) == "number",
12876             [[User-defined syntax extension "]] .. pathname
12877             .. [[" specifies field "api_version" of type "]]
12878             .. type(user_extension.api_version)
12879             .. [[" but "number" was expected]])
12880         assert(user_extension.api_version > 0
12881             and user_extension.api_version
12882             <= metadata.user_extension_api_version,
12883             [[User-defined syntax extension "]] .. pathname
12884             .. [[" uses syntax extension API version "]]
12885             .. user_extension.api_version .. [[" but markdown.lua ]]
12886             .. metadata.version .. [[" uses API version ]]
12887             .. metadata.user_extension_api_version
12888             .. [[" which is incompatible]])
12889
12890         assert(user_extension.grammar_version ~= nil,
12891             [[User-defined syntax extension "]] .. pathname
12892             .. [[" does not specify mandatory field "grammar_version"]])
12893         assert(type(user_extension.grammar_version) == "number",
12894             [[User-defined syntax extension "]] .. pathname
12895             .. [[" specifies field "grammar_version" of type "]]
12896             .. type(user_extension.grammar_version)
```

```

12897     .. [" but "number" was expected]])
12898     assert(user_extension.grammar_version == metadata.grammar_version,
12899           [[User-defined syntax extension "]] .. pathname
12900           .. [" uses grammar version "]]
12901           .. user_extension.grammar_version
12902           .. [[ but markdown.lua ]] .. metadata.version
12903           .. [[ uses grammar version ]] .. metadata.grammar_version
12904           .. [[, which is incompatible]])
12905
12906     assert(user_extension.finalize_grammar ~= nil,
12907           [[User-defined syntax extension "]] .. pathname
12908           .. [" does not specify mandatory "finalize_grammar" field]])
12909     assert(type(user_extension.finalize_grammar) == "function",
12910           [[User-defined syntax extension "]] .. pathname
12911           .. [" specifies field "finalize_grammar" of type "]]
12912           .. type(user_extension.finalize_grammar)
12913           .. [" 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.)

```

12914     local extension = {
12915       name = [[user-defined "]] .. pathname .. [" syntax extension]],
12916       extend_reader = user_extension.finalize_grammar,
12917       extend_writer = function() end,
12918     }
12919     return extension
12920   end)(user_extension_filename)
12921   table.insert(extensions, user_extension)
12922 end

```

Produce a conversion function from markdown to plain TeX.

```

12923 local writer = M.writer.new(options)
12924 local reader = M.reader.new(writer, options)
12925 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`.

```

12926 collectgarbage("collect")

```

Update the singleton cache.

```

12927 if options.singletonCache then
12928   local singletonCacheOptions = {}
12929   for k, v in pairs(options) do
12930     singletonCacheOptions[k] = v
12931   end
12932   setmetatable(singletonCacheOptions,
12933     { __index = function (_, key)
12934       return defaultOptions[key] end })

```

```

12935     singletonCache.options = singletonCacheOptions
12936     singletonCache.convert = convert
12937 end

```

Return the conversion function from markdown to plain T<sub>E</sub>X.

```

12938     return convert
12939 end

12940 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.

```

12941
12942 local input
12943 if input_filename then
12944     local input_file = assert(io.open(input_filename, "r"),
12945         [[Could not open file ]] .. input_filename .. [[ for reading]])
12946     input = assert(input_file:read("*a"))
12947     assert(input_file:close())
12948 else
12949     input = assert(io.read("*a"))
12950 end
12951

```

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

```

12952 local lfs = require("lfs")
12953 if options.cacheDir and not lfs.isdir(options.cacheDir) then
12954     assert(lfs.mkdir(options["cacheDir"]))
12955 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.

```

12956 local kpse
12957 (function()
12958     local should_initialize = package.loaded.kpse == nil
12959                             or tex.initialize ~= nil
12960     kpse = require("kpse")
12961     if should_initialize then
12962         kpse.set_program_name("luatex")
12963     end
12964 end)()
12965 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.

```

12966 if metadata.version ~= md.metadata.version then

```

```

12967   warn("markdown-cli.lua " .. metadata.version .. " used with " ..
12968         "markdown.lua " .. md.metadata.version .. ".")
12969 end
12970
12971 local convert = md.new(options)
12972 local raw_output, flat_output = convert(input, true)
12973 local output
12974 if flat_output == nil then
12975   if options.eagerCache then
12976     warn("markdown.lua has not produced flat output, so I am using " ..
12977         "backwards-compatible raw output instead. This may cause " ..
12978         "the conversion result to be hidden behind "\\input'.')
12979   end
12980   output = raw_output
12981 else
12982   output = flat_output()
12983 end
12984
12985 if output_filename then
12986   local output_file = assert(io.open(output_filename, "w"),
12987     [[Could not open file ]] .. output_filename .. [[ for writing]])
12988   assert(output_file:write(output))
12989   assert(output_file:close())
12990 else
12991   assert(io.write(output))
12992 end

```

Remove the `options.cacheDir` directory if it is empty.

```

12993 if options.cacheDir then
12994   lfs.rmdir(options.cacheDir)
12995 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

```

12996 \ExplSyntaxOn
12997 \cs_if_free:NT
12998   \markdownInfo
12999   {
13000     \cs_new:Npn
13001       \markdownInfo #1

```

```

13002     {
13003         \msg_info:nne
13004         { markdown }
13005         { generic-message }
13006         { #1 }
13007     }
13008 }
13009 \cs_if_free:NT
13010 \markdownWarning
13011 {
13012     \cs_new:Npn
13013     \markdownWarning #1
13014     {
13015         \msg_warning:nne
13016         { markdown }
13017         { generic-message }
13018         { #1 }
13019     }
13020 }
13021 \cs_if_free:NT
13022 \markdownError
13023 {
13024     \cs_new:Npn
13025     \markdownError #1 #2
13026     {
13027         \msg_error:nnee
13028         { markdown }
13029         { generic-message-with-help-text }
13030         { #1 }
13031         { #2 }
13032     }
13033 }
13034 \msg_new:nnn
13035 { markdown }
13036 { generic-message }
13037 { #1 }
13038 \msg_new:nnnn
13039 { markdown }
13040 { generic-message-with-help-text }
13041 { #1 }
13042 { #2 }
13043 \cs_generate_variant:Nn
13044 \msg_info:nnn
13045 { nne }
13046 \cs_generate_variant:Nn
13047 \msg_warning:nnn
13048 { nne }

```

```

13049 \cs_generate_variant:Nn
13050   \msg_error:nnnn
13051   { nnee }
13052 \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.

```

13053 \ExplSyntaxOn
13054 \prop_new:N \g_@@_plain_tex_loaded_themes_linenos_prop
13055 \prop_new:N \g_@@_plain_tex_loaded_themes_versions_prop
13056 \cs_new:Nn
13057   \@@_plain_tex_load_theme:nnn
13058   {
13059     \prop_get:NnNTF
13060       \g_@@_plain_tex_loaded_themes_linenos_prop
13061       { #1 }
13062       \l_tmpa_tl
13063       {
13064         \prop_get:NnN
13065           \g_@@_plain_tex_loaded_themes_versions_prop
13066           { #1 }
13067           \l_tmpb_tl
13068         \str_if_eq:nVTF
13069           { #2 }
13070           \l_tmpb_tl
13071           {
13072             \msg_warning:nnnVn
13073               { markdown }
13074               { repeatedly-loaded-plain-tex-theme }
13075               { #1 }
13076               \l_tmpa_tl
13077               { #2 }
13078           }
13079           {
13080             \msg_error:nnnnVV
13081               { markdown }
13082               { different-versions-of-plain-tex-theme }
13083               { #1 }
13084               { #2 }
13085               \l_tmpb_tl
13086               \l_tmpa_tl
13087           }
13088       }
13089   {

```

```

13090 \prop_gput:Nnx
13091 \g_@@_plain_tex_loaded_themes_linenos_prop
13092 { #1 }
13093 { \tex_the:D \tex_inputlineno:D } % noqa: W200
13094 \prop_gput:Nnn
13095 \g_@@_plain_tex_loaded_themes_versions_prop
13096 { #1 }
13097 { #2 }

```

Load built-in plain TeX themes from the prop `\g_@@_plain_tex_built_in_themes_prop` and from the filesystem otherwise.

```

13098 \prop_if_in:NnTF
13099 \g_@@_plain_tex_built_in_themes_prop
13100 { #1 }
13101 {
13102   \msg_info:nnnn
13103   { markdown }
13104   { loading-built-in-plain-tex-theme }
13105   { #1 }
13106   { #2 }
13107   \prop_item:Nn
13108   \g_@@_plain_tex_built_in_themes_prop
13109   { #1 }
13110 }
13111 {
13112   \msg_info:nnnn
13113   { markdown }
13114   { loading-plain-tex-theme }
13115   { #1 }
13116   { #2 }
13117   \file_input:n
13118   { markdown theme #3 }
13119 }
13120 }
13121 }
13122 \msg_new:nnn
13123 { markdown }
13124 { loading-plain-tex-theme }
13125 { Loading-version~#2~of~plain~TeX~Markdown~theme~#1 }
13126 \msg_new:nnn
13127 { markdown }
13128 { loading-built-in-plain-tex-theme }
13129 { Loading-version~#2~of~built-in~plain~TeX~Markdown~theme~#1 }
13130 \msg_new:nnn
13131 { markdown }
13132 { repeatedly-loaded-plain-tex-theme }
13133 {

```



```

13134     Version~#3~of~plain~TeX~Markdown~theme~#1~was~previously~
13135     loaded~on~line~#2,~not~loading~it~again
13136   }
13137   \msg_new:nnn
13138   { markdown }
13139   { different-versions-of-plain-tex-theme }
13140   {
13141     Tried~to~load~version~#2~of~plain~TeX~Markdown~theme~#1~
13142     but~version~#3~has~already~been~loaded~on~line~#4
13143   }
13144   \cs_generate_variant:Nn
13145   \prop_gput:Nnn
13146   { Nnx }
13147   \cs_gset_eq:NN
13148   \@@_load_theme:nnn
13149   \@@_plain_tex_load_theme:nnn
13150   \cs_generate_variant:Nn
13151   \@@_load_theme:nnn
13152   { VeV }
13153   \cs_generate_variant:Nn
13154   \msg_error:nnnnnn
13155   { nnnnVV }
13156   \cs_generate_variant:Nn
13157   \msg_warning:nnnnn
13158   { nnnVn }

```

Developers can use the `\markdownLoadPlainTeXTheme` macro to load a corresponding plain T<sub>E</sub>X theme from within themes for 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.

```

13159   \cs_new:Npn
13160   \markdownLoadPlainTeXTheme
13161   {

```

First, we extract the name of the current theme from the `\g_@@_current_theme_tl` macro.

```

13162     \tl_set:NV
13163     \l_tmpa_tl
13164     \g_@@_current_theme_tl
13165     \tl_reverse:N
13166     \l_tmpa_tl
13167     \tl_set:Ne
13168     \l_tmpb_tl
13169     {
13170       \tl_tail:V
13171       \l_tmpa_tl
13172     }
13173     \tl_reverse:N
13174     \l_tmpb_tl

```

Next, we munge the theme name.

```
13175 \str_set:NV
13176 \l_tmpa_str
13177 \l_tmpb_tl
13178 \str_replace_all:Nnn
13179 \l_tmpa_str
13180 { / }
13181 { _ }
```

Finally, we load the plain TeX theme.

```
13182 \@@_plain_tex_load_theme:VeV
13183 \l_tmpb_tl
13184 { \markdownThemeVersion }
13185 \l_tmpa_str
13186 }
13187 \cs_generate_variant:Nn
13188 \tl_set:Nn
13189 { Ne }
13190 \cs_generate_variant:Nn
13191 \@@_plain_tex_load_theme:nnn
13192 { VeV }
```

The `witiko/dot` theme nags users that they should use the name `witiko/diagrams@v1` instead.

```
13193 \prop_gput:Nnn
13194 \g_@@_plain_tex_built_in_themes_prop
13195 { witiko / dot }
13196 {
13197 \str_if_eq:enF
13198 { \markdownThemeVersion }
13199 { silent }
13200 {
13201 \markdownWarning
13202 {
13203 The~theme~name~"witiko/dot"~has~been~soft-deprecated.
13204 \iow_newline:
13205 Consider~changing~the~name~to~"witiko/diagrams@v1".
13206 }
13207 }
```

We enable the `fencedCode` Lua option.

```
13208 \markdownSetup { fencedCode }
```

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

```
13209 \cs_set_eq:NN
13210 \@@_dot_previous_definition:nnn
13211 \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:

```

13212 \regex_const:Nn
13213 \c_@@_dot_infostring_regex
13214 { ^dot(\s+(.+)?)? }
13215 \seq_new:N
13216 \l_@@_dot_matches_seq
13217 \markdownSetup {
13218   rendererPrototypes = {
13219     inputFencedCode = {
13220       \regex_extract_once:NnNTF
13221       \c_@@_dot_infostring_regex
13222       { #2 }
13223       \l_@@_dot_matches_seq
13224       {
13225         \@@_if_option:nF
13226         { frozenCache }
13227         {
13228           \sys_shell_now:n
13229           {
13230             if!~test~-e~#1.pdf.source~
13231             ||!~diff~#1~#1.pdf.source;
13232             then~
13233             dot~-Tpdf~-o~#1.pdf~#1;
13234             cp~#1~#1.pdf.source;
13235             fi
13236           }
13237         }

```

We include the typeset image using the image token renderer:

```

13238 \exp_args:NNne
13239 \exp_last_unbraced:No
13240 \markdownRendererImage
13241 {
13242   { Graphviz~image }
13243   { #1.pdf }
13244   { #1.pdf }
13245 }
13246 {
13247   \seq_item:Nn
13248   \l_@@_dot_matches_seq
13249   { 3 }
13250 }
13251 }

```

If the infostring does not start with `dot ...`, we use the previous definition of the fenced code token renderer prototype:

```

13252         {
13253             \@@_dot_previous_definition:nnn
13254             { #1 }
13255             { #2 }
13256             { #3 }
13257         }
13258     },
13259 },
13260 }
13261 }
```

The theme `witiko/diagrams` loads either the theme `witiko/dot` for version `v1` or the theme `witiko/diagrams/v2` for version `v2`.

```

13262 \prop_gput:Nnn
13263 \g_@@_plain_tex_built_in_themes_prop
13264 { witiko / diagrams }
13265 {
13266     \str_case:enF
13267     { \markdownThemeVersion }
13268     {
13269         { latest }
13270         {
13271             \markdownWarning
13272             {
13273                 Write~"witiko/diagrams@v2"~to~pin~version~"v2"~of~the~
13274                 theme~"witiko/diagrams".~This~will~keep~your~documents~
13275                 from~suddenly~breaking~when~we~have~released~future~
13276                 versions~of~the~theme~with~backwards~incompatible~
13277                 syntax~and~behavior.
13278             }
13279             \markdownSetup
13280             {
13281                 import = witiko/diagrams/v2,
13282             }
13283         }
13284     { v2 }
13285     {
13286         \markdownSetup
13287         {
13288             import = witiko/diagrams/v2,
13289         }
13290     }
13291     { v1 }
13292     {
13293         \markdownSetup
```

```

13294         {
13295             import = witiko/dot@silent,
13296         }
13297     }
13298 }
13299 {
13300     \msg_error:nnnen
13301     { markdown }
13302     { unknown-theme-version }
13303     { witiko/diagrams }
13304     { \markdownThemeVersion }
13305     { v1 }
13306 }
13307 }
13308 \cs_generate_variant:Nn
13309 \msg_error:nnnnn
13310 { nnnen }
13311 \msg_new:nnnn
13312 { markdown }
13313 { unknown-theme-version }
13314 { Unknown~version~"#2"~of~theme~"#1"~has~been~requested. }
13315 { Known~versions~are:~#3 }

```

Next, we implement the theme `witiko/diagrams/v2`.

```

13316 \prop_gput:Nnn
13317 \g_@@_plain_tex_built_in_themes_prop
13318 { witiko / diagrams / v2 }
13319 {

```

We enable the `fencedCode` and `fencedCodeAttributes` Lua option.

```

13320 \@@_setup:n
13321 {
13322     fencedCode = true,
13323     fencedCodeAttributes = true,
13324 }

```

Store the previous fenced code token renderer prototype.

```

13325 \cs_set_eq:NN
13326 \@@_diagrams_previous_fenced_code:nnn
13327 \markdownRendererInputFencedCodePrototype

```

Store the caption and the desired format of the diagram.

```

13328 \tl_new:N
13329 \l_@@_diagrams_caption_tl
13330 \tl_new:N
13331 \l_@@_diagrams_format_tl
13332 \tl_set:Nn
13333 \l_@@_diagrams_format_tl
13334 { pdf }

```

```

13335 \@@_setup:n
13336 {
13337     rendererPrototypes = {
Route attributes on fenced code blocks to the image attribute renderer prototypes.
13338     fencedCodeAttributeContextBegin = {
13339         \group_begin:
13340         \markdownRendererImageAttributeContextBegin
13341         \cs_set_eq:NN
13342         \@@_diagrams_previous_key_value:nn
13343         \markdownRendererAttributeKeyValuePrototype
13344         \@@_setup:n
13345         {
13346             rendererPrototypes = {
13347                 attributeKeyValue = {
13348                     \str_case:nnF
13349                     { ##1 }
13350                     {
13351                         { caption }
13352                         {
13353                             \tl_set:Nn
13354                             \l_@@_diagrams_caption_tl
13355                             { ##2 }
13356                         }
13357                         { format }
13358                         {
13359                             \tl_set:Nn
13360                             \l_@@_diagrams_format_tl
13361                             { ##2 }
13362                         }
13363                     }
13364                     {
13365                         \@@_diagrams_previous_key_value:nn
13366                         { ##1 }
13367                         { ##2 }
13368                     }
13369                 },
13370             },
13371         }
13372     },
13373     fencedCodeAttributeContextEnd = {
13374         \markdownRendererImageAttributeContextEnd
13375         \group_end:
13376     },
13377 },
13378 }
13379 \cs_new:Nn
13380 \@@_diagrams_render_diagram:nnnn

```

```

13381 {
13382   \@@_if_option:nF
13383   { frozenCache }
13384   {
13385     \sys_shell_now:n
13386     {
13387       if~!~test~-e~#2.source~
13388       ||~!~diff~#1~#2.source;
13389       then~
13390         (#3);
13391         cp~#1~#2.source;
13392       fi
13393     }
13394     \exp_args:NNnV
13395     \exp_last_unbraced:No
13396     \markdownRendererImage
13397     {
13398       { #4 }
13399       { #2 }
13400       { #2 }
13401     }
13402     \l_@@_diagrams_caption_tl
13403   }
13404 }

```

Use the prop `\g_markdown_diagrams_infostrings_prop` to determine how the code with a given infostring should be processed and routed to the token renderer prototype(s) for images.

```

13405   \prop_new:N
13406   \g_markdown_diagrams_infostrings_prop

```

If we know a processing function for a given infostring, use it.

```

13407   \@@_setup:n
13408   {
13409     rendererPrototypes = {
13410       inputFencedCode = {
13411         \prop_get:NnNTF
13412         \g_markdown_diagrams_infostrings_prop
13413         { #2 }
13414         \l_tmpa_tl
13415         {
13416           \cs_set:NV
13417           \@@_diagrams_infostrings_current:n
13418           \l_tmpa_tl
13419           \@@_diagrams_infostrings_current:n
13420           { #1 }
13421         }

```

Otherwise, use the previous fenced code token renderer prototype.

```

13422         {
13423             \@@_diagrams_previous_fenced_code:nmm
13424             { #1 }
13425             { #2 }
13426             { #3 }
13427         }
13428     },
13429 },
13430 }
13431 \cs_generate_variant:Nn
13432 \cs_set:Nn
13433 { NV }

```

Typeset fenced code with infostring `dot` using the command `dot` from the package `Graphviz`.

```

13434 \cs_set:Nn
13435 \@@_diagrams_infostrings_current:n
13436 {
13437     \@@_diagrams_render_diagram:nnnn
13438     { #1 }
13439     { #1.pdf }
13440     { dot~--Tpdf~-o~#1.pdf~#1 }
13441     { Graphviz~image }
13442 }
13443 \@@_tl_set_from_cs:NNn
13444 \l_tmpa_tl
13445 \@@_diagrams_infostrings_current:n
13446 { 1 }
13447 \prop_gput:NnV
13448 \g_markdown_diagrams_infostrings_prop
13449 { dot }
13450 \l_tmpa_tl

```

Typeset fenced code with infostring `mermaid` using the command `mmdc` from the npm package `@mermaid-js/mermaid-cli`.

```

13451 \cs_set:Nn
13452 \@@_diagrams_infostrings_current:n
13453 {
13454     \@@_diagrams_render_diagram:nnnn
13455     { #1 }
13456     { #1.pdf }
13457     { mmdc~--pdfFit~-i~#1~-o~#1.pdf }
13458     { Mermaid~image }
13459 }
13460 \@@_tl_set_from_cs:NNn
13461 \l_tmpa_tl

```



```

13462     \@@_diagrams_infostrings_current:n
13463     { 1 }
13464     \prop_gput:NnV
13465     \g_markdown_diagrams_infostrings_prop
13466     { mermaid }
13467     \l_tmpa_tl

```

Typeset fenced code with infostring `plantuml` using the command `plantuml` from the package PlantUML.

```

13468     \regex_const:Nn
13469     \c_@@_diagrams_filename_suffix_regex
13470     { \.[^\.]*$ }
13471     \cs_set:Nn
13472     \@@_diagrams_infostrings_current:n
13473     {

```

Use the output format provided by the user.

```

13474     \tl_set:Nn
13475     \l_tmpa_tl
13476     { #1 }
13477     \regex_replace_once:NxN
13478     \c_@@_diagrams_filename_suffix_regex
13479     {
13480     .
13481     \tl_use:N
13482     \l_@@_diagrams_format_tl
13483     }
13484     \l_tmpa_tl
13485     \tl_set:Nn
13486     \l_tmpb_tl
13487     { plantuml~-t }
13488     \tl_put_right:NV
13489     \l_tmpb_tl
13490     \l__markdown_diagrams_format_tl
13491     \tl_put_right:Nn
13492     \l_tmpb_tl
13493     { ~#1 }

```

For the SVG format, use Inkscape to convert the resulting image to PDF.

```

13494     \str_if_eq:VnT
13495     \l_@@_diagrams_format_tl
13496     { svg }
13497     {
13498     \tl_put_right:Nn
13499     \l_tmpb_tl
13500     { ;~inkscape~ }
13501     \tl_put_right:NV
13502     \l_tmpb_tl

```

```

13503         \l_tmpa_tl
13504     \tl_put_right:Nn
13505         \l_tmpb_tl
13506         { ---export-area-drawing---export-dpi=300~-o~ }
13507     \tl_set:Nn
13508         \l_tmpa_tl
13509         { #1 }
13510     \regex_replace_once:NnN
13511         \c_@@_diagrams_filename_suffix_regex
13512         { .pdf }
13513         \l_tmpa_tl
13514     \tl_put_right:NV
13515         \l_tmpb_tl
13516         \l_tmpa_tl
13517 }
13518 \@@_diagrams_render_diagram:nVVn
13519 { #1 }
13520     \l_tmpa_tl
13521     \l_tmpb_tl
13522     { PlantUML~image }
13523 }
13524 \cs_generate_variant:Nn
13525     \@@_diagrams_render_diagram:nnnn
13526     { nVVn }
13527 \cs_generate_variant:Nn
13528     \regex_replace_once:NnN
13529     { NxN }
13530 \@@_tl_set_from_cs:NNn
13531     \l_tmpa_tl
13532     \@@_diagrams_infostrings_current:n
13533     { 1 }
13534 \prop_gput:NnV
13535     \g_markdown_diagrams_infostrings_prop
13536     { plantuml }
13537     \l_tmpa_tl
13538 }

```

We locally change the category code of percent signs, so that we can use them in the shell code:

```

13539 \group_begin:
13540 \char_set_catcode_other:N \%

```

The [witiko/graphicx/http](https://github.com/witiko/graphicx/http) theme stores the previous definition of the image token renderer prototype:

```

13541 \prop_gput:Nnn
13542     \g_@@_plain_tex_built_in_themes_prop
13543     { witiko / graphicx / http }
13544     {

```

```

13545 \cs_set_eq:NN
13546 \@@_graphicx_http_previous_definition:nnnn
13547 \markdownRendererImagePrototype

```

We define variables and functions to enumerate the images for caching and to store the pathname of the file containing the pathname of the downloaded image file.

```

13548 \int_new:N
13549 \g_@@_graphicx_http_image_number_int
13550 \int_gset:Nn
13551 \g_@@_graphicx_http_image_number_int
13552 { 0 }
13553 \cs_new:Nn
13554 \@@_graphicx_http_filename:
13555 {
13556 \markdownOptionCacheDir
13557 / witiko_graphicx_http .
13558 \int_use:N
13559 \g_@@_graphicx_http_image_number_int
13560 }

```

We define a function that 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 function produces a shell command that tries to download the online image to the pathname.

```

13561 \cs_new:Nn
13562 \@@_graphicx_http_download:nn
13563 {
13564 wget~-0~#2~#1~
13565 ||~curl~--location~-o~#2~#1~
13566 ||~rm~-f~#2
13567 }

```

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

```

13568 \str_new:N
13569 \l_@@_graphicx_http_filename_str
13570 \ior_new:N
13571 \g_@@_graphicx_http_filename_ior
13572 \markdownSetup {
13573 rendererPrototypes = {
13574 image = {
13575 \@@_if_option:nF
13576 { frozenCache }
13577 {

```

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:

```

13578 \sys_shell_now:e

```

```

13579         {
13580             mkdir~-p~" \markdownOptionCacheDir ";
13581             if~printf~'%s'~"#3"~|~grep~-q~-E~'^https?:';
13582             then~

```

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

```

13583             OUTPUT_PREFIX=" \markdownOptionCacheDir ";
13584             OUTPUT_BODY="$(printf~'%s'~'#3'
13585                             |~md5sum~|~cut~-d'~'-f1)";
13586             OUTPUT_SUFFIX="$(printf~'%s'~'#3'
13587                             |~sed~'s/.*[.]//')";
13588             OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";

```

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

```

13589             if~!~[~-e~"$OUTPUT"~];
13590             then~
13591                 \@@_graphicx_http_download:nn
13592                 { '#3' }
13593                 { "$OUTPUT" } ;
13594                 printf~'%s'~"$OUTPUT"~
13595                 >~" \@@_graphicx_http_filename: ";
13596             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:

```

13597             else~
13598                 printf~'%s'~'#3'~
13599                 >~" \@@_graphicx_http_filename: ";
13600             fi
13601         }
13602     }

```

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

```

13603         \ior_open:Ne
13604         \g_@@_graphicx_http_filename_ior
13605         { \@@_graphicx_http_filename: }
13606         \ior_str_get:NN
13607         \g_@@_graphicx_http_filename_ior
13608         \l_@@_graphicx_http_filename_str
13609         \ior_close:N
13610         \g_@@_graphicx_http_filename_ior
13611         \@@_graphicx_http_previous_definition:nnVn
13612         { #1 }
13613         { #2 }
13614         \l_@@_graphicx_http_filename_str
13615         { #4 }

```

```

13616         \int_gincr:N
13617         \g_@@_graphicx_http_image_number_int
13618     }
13619 }
13620 }
13621 \cs_generate_variant:Nn
13622   \ior_open:Nn
13623   { Ne }
13624 \cs_generate_variant:Nn
13625   \@@_graphicx_http_previous_definition:nnnn
13626   { nnVn }
13627 }
13628 \group_end:

```

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

```

13629 \prop_gput:Nnn
13630   \g_@@_plain_tex_built_in_themes_prop
13631   { witiko / tilde }
13632   {
13633     \markdownSetup {
13634       rendererPrototypes = {
13635         tilde = {~},
13636       },
13637     }
13638   }

```

The themes [witiko/example/foo](#) and [witiko/example/bar](#) are supposed to be used in code examples. They don't do anything.

```

13639 \clist_map_inline:nn
13640   { foo, bar }
13641   {
13642     \prop_gput:Nnn
13643       \g_@@_plain_tex_built_in_themes_prop
13644       { witiko / example / #1 }
13645       {
13646         \markdownWarning
13647         {
13648           The~theme~witiko/example/#1~is~supposed~to~be~used~in~code~
13649           examples.~Using~it~in~actual~code~has~no~effect,~except~
13650           this~warning~message,~and~is~usually~a~mistake.
13651         }
13652       }
13653   }
13654 \ExplSyntaxOff

```

The [witiko/markdown/defaults](#) plain T<sub>E</sub>X 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.

```
13655 \def\markdownRendererInterblockSeparatorPrototype{\par}%
13656 \def\markdownRendererParagraphSeparatorPrototype{%
13657   \markdownRendererInterblockSeparator}%
13658 \def\markdownRendererHardLineBreakPrototype{\hfil\break}%
13659 \def\markdownRendererSoftLineBreakPrototype{ }%
13660 \let\markdownRendererEllipsisPrototype\dots
13661 \def\markdownRendererNbspPrototype{~}%
13662 \def\markdownRendererLeftBracePrototype{\char`\{}%
13663 \def\markdownRendererRightBracePrototype{\char`\}%
13664 \def\markdownRendererDollarSignPrototype{\char`$}%
13665 \def\markdownRendererPercentSignPrototype{\char`\}%
13666 \def\markdownRendererAmpersandPrototype{\&}%
13667 \def\markdownRendererUnderscorePrototype{\char`_}%
13668 \def\markdownRendererHashPrototype{\char`\#}%
13669 \def\markdownRendererCircumflexPrototype{\char`^}%
13670 \def\markdownRendererBackslashPrototype{\char`\}%
13671 \def\markdownRendererTildePrototype{\char`~}%
13672 \def\markdownRendererPipePrototype{|}%
13673 \def\markdownRendererCodeSpanPrototype#1{\tt#1}%
13674 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
13675 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
13676   \markdownInput{#3}}%
13677 \def\markdownRendererContentBlockOnlineImagePrototype{%
13678   \markdownRendererImage}%
13679 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
13680   \markdownRendererInputFencedCode{#3}{#2}{#2}}%
13681 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
13682 \def\markdownRendererUlBeginPrototype{}%
13683 \def\markdownRendererUlBeginTightPrototype{}%
13684 \def\markdownRendererUlItemPrototype{}%
13685 \def\markdownRendererUlItemEndPrototype{}%
13686 \def\markdownRendererUlEndPrototype{}%
13687 \def\markdownRendererUlEndTightPrototype{}%
13688 \def\markdownRendererOlBeginPrototype{}%
13689 \def\markdownRendererOlBeginTightPrototype{}%
13690 \def\markdownRendererFancyOlBeginPrototype#1#2{%
13691   \markdownRendererOlBegin}%
13692 \def\markdownRendererFancyOlBeginTightPrototype#1#2{%
13693   \markdownRendererOlBeginTight}%
13694 \def\markdownRendererOlItemPrototype{}%
13695 \def\markdownRendererOlItemWithNumberPrototype#1{}%
13696 \def\markdownRendererOlItemEndPrototype{}%
13697 \def\markdownRendererFancyOlItemPrototype{\markdownRendererOlItem}%
13698 \def\markdownRendererFancyOlItemWithNumberPrototype{%
```

```

13699 \markdownRendererOlItemWithNumber}%
13700 \def\markdownRendererFancyOlItemEndPrototype{}%
13701 \def\markdownRendererOlEndPrototype{}%
13702 \def\markdownRendererOlEndTightPrototype{}%
13703 \def\markdownRendererFancyOlEndPrototype{\markdownRendererOlEnd}%
13704 \def\markdownRendererFancyOlEndTightPrototype{%
13705 \markdownRendererOlEndTight}%
13706 \def\markdownRendererDlBeginPrototype{}%
13707 \def\markdownRendererDlBeginTightPrototype{}%
13708 \def\markdownRendererDlItemPrototype#1{#1}%
13709 \def\markdownRendererDlItemEndPrototype{}%
13710 \def\markdownRendererDlDefinitionBeginPrototype{}%
13711 \def\markdownRendererDlDefinitionEndPrototype{\par}%
13712 \def\markdownRendererDlEndPrototype{}%
13713 \def\markdownRendererDlEndTightPrototype{}%
13714 \def\markdownRendererEmphasisPrototype#1{\it#1}%
13715 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
13716 \def\markdownRendererBlockQuoteBeginPrototype{\begingroup\it}%
13717 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
13718 \def\markdownRendererLineBlockBeginPrototype{\begingroup\parindent=0pt}%
13719 \def\markdownRendererLineBlockEndPrototype{\endgroup}%
13720 \def\markdownRendererInputVerbatimPrototype#1{%
13721 \par{\tt\input#1\relax}}\par}%
13722 \def\markdownRendererInputFencedCodePrototype#1#2#3{%
13723 \markdownRendererInputVerbatim{#1}}%
13724 \def\markdownRendererHeadingOnePrototype#1{#1}%
13725 \def\markdownRendererHeadingTwoPrototype#1{#1}%
13726 \def\markdownRendererHeadingThreePrototype#1{#1}%
13727 \def\markdownRendererHeadingFourPrototype#1{#1}%
13728 \def\markdownRendererHeadingFivePrototype#1{#1}%
13729 \def\markdownRendererHeadingSixPrototype#1{#1}%
13730 \def\markdownRendererThematicBreakPrototype{}%
13731 \def\markdownRendererNotePrototype#1{#1}%
13732 \def\markdownRendererCitePrototype#1{}%
13733 \def\markdownRendererTextCitePrototype#1{}%
13734 \def\markdownRendererTickedBoxPrototype{[X]}%
13735 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
13736 \def\markdownRendererUntickedBoxPrototype{[ ]}%
13737 \def\markdownRendererStrikeThroughPrototype#1{#1}%
13738 \def\markdownRendererSuperscriptPrototype#1{#1}%
13739 \def\markdownRendererSubscriptPrototype#1{#1}%
13740 \def\markdownRendererDisplayMathPrototype#1{$$#1$$}%
13741 \def\markdownRendererInlineMathPrototype#1{$#1$}%
13742 \ExplSyntaxOn
13743 \cs_gset:Npn
13744 \markdownRendererHeaderAttributeContextBeginPrototype
13745 {

```

```

13746     \group_begin:
13747     \color_group_begin:
13748   }
13749   \cs_gset:Npn
13750     \markdownRendererHeaderAttributeContextEndPrototype
13751   {
13752     \color_group_end:
13753     \group_end:
13754   }
13755   \cs_gset_eq:NN
13756     \markdownRendererBracketedSpanAttributeContextBeginPrototype
13757     \markdownRendererHeaderAttributeContextBeginPrototype
13758   \cs_gset_eq:NN
13759     \markdownRendererBracketedSpanAttributeContextEndPrototype
13760     \markdownRendererHeaderAttributeContextEndPrototype
13761   \cs_gset_eq:NN
13762     \markdownRendererFencedDivAttributeContextBeginPrototype
13763     \markdownRendererHeaderAttributeContextBeginPrototype
13764   \cs_gset_eq:NN
13765     \markdownRendererFencedDivAttributeContextEndPrototype
13766     \markdownRendererHeaderAttributeContextEndPrototype
13767   \cs_gset_eq:NN
13768     \markdownRendererFencedCodeAttributeContextBeginPrototype
13769     \markdownRendererHeaderAttributeContextBeginPrototype
13770   \cs_gset_eq:NN
13771     \markdownRendererFencedCodeAttributeContextEndPrototype
13772     \markdownRendererHeaderAttributeContextEndPrototype
13773   \cs_gset:Npn
13774     \markdownRendererReplacementCharacterPrototype
13775   { \codepoint_str_generate:n { fffd } }
13776   \ExplSyntaxOff
13777   \def\markdownRendererSectionBeginPrototype{}%
13778   \def\markdownRendererSectionEndPrototype{}%
13779   \ExplSyntaxOn
13780   \cs_gset:Npn
13781     \markdownRendererWarningPrototype
13782     #1#2#3#4
13783   {
13784     \tl_set:Nn
13785       \l_tmpa_tl
13786       { #2 }
13787     \tl_if_empty:nF
13788       { #4 }
13789     {
13790       \tl_put_right:Nn
13791         \l_tmpa_tl
13792         { \iow_newline: #4 }

```



```

13793     }
13794     \exp_args:NV
13795     \markdownWarning
13796     \l_tmpa_tl
13797   }
13798   \ExplSyntaxOff
13799   \def\markdownRendererErrorPrototype#1#2#3#4{%
13800     \markdownError{#2}{#4}}%

```

### 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.

```

13801 \ExplSyntaxOn
13802 \cs_new:Nn
13803   \@@_plain_tex_default_input_raw_inline:nn
13804   {
13805     \str_case:nn
13806       { #2 }
13807       {
13808         { md } { \markdownInput{#1} }
13809         { tex } { \markdownEscape{#1} \unskip }
13810       }
13811   }
13812 \cs_new:Nn
13813   \@@_plain_tex_default_input_raw_block:nn
13814   {
13815     \str_case:nn
13816       { #2 }
13817       {
13818         { md } { \markdownInput{#1} }
13819         { tex } { \markdownEscape{#1} }
13820       }
13821   }
13822 \cs_gset:Npn
13823   \markdownRendererInputRawInlinePrototype#1#2
13824   {
13825     \@@_plain_tex_default_input_raw_inline:nn
13826       { #1 }
13827       { #2 }
13828   }
13829 \cs_gset:Npn
13830   \markdownRendererInputRawBlockPrototype#1#2
13831   {
13832     \@@_plain_tex_default_input_raw_block:nn
13833       { #1 }

```

```

13834         { #2 }
13835     }
13836 \ExplSyntaxOff

```

### 3.2.3.2 Simple YAML Metadata Renderer Prototypes

In this section, we implement the simple high-level interface for processing simple YAML metadata using the key-value [markdown/jekyllData](#). See also Section 2.2.6.1.

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  $p$ .

```

13837 \ExplSyntaxOn
13838 \seq_new:N \g_@@_jekyll_data_datatypes_seq
13839 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
13840 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
13841 \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 `\@@_jekyll_data_push_address_segment:n` macro.

```

13842 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
13843 \cs_new:Nn \@@_jekyll_data_push_address_segment:n
13844 {
13845     \seq_if_empty:NF
13846     \g_@@_jekyll_data_datatypes_seq
13847     {
13848         \seq_get_right:NN
13849         \g_@@_jekyll_data_datatypes_seq
13850         \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.

```

13851     \str_if_eq:NNTF

```

```

13852     \l_tmpa_tl
13853     \c_@@_jekyll_data_sequence_tl
13854     {
13855         \seq_put_right:Nn
13856         \g_@@_jekyll_data_wildcard_absolute_address_seq
13857         { * }
13858     }
13859     {
13860         \seq_put_right:Nn
13861         \g_@@_jekyll_data_wildcard_absolute_address_seq
13862         { #1 }
13863     }
13864 }
13865 }

```

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 `\@@_jekyll_data_concatenate_address:NN` macro and we will construct both token lists using the `\@@_jekyll_data_update_address_tls:` macro.

```

13866 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
13867 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
13868 \cs_new:Nn \@@_jekyll_data_concatenate_address:NN
13869 {
13870     \seq_pop_left:NN #1 \l_tmpa_tl
13871     \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }

```

```

13872     \seq_put_left:NV #1 \l_tmpa_tl
13873   }
13874 \cs_new:Nn \@@_jekyll_data_update_address_tls:
13875 {
13876   \@@_jekyll_data_concatenate_address:NN
13877   \g_@@_jekyll_data_wildcard_absolute_address_seq
13878   \g_@@_jekyll_data_wildcard_absolute_address_tl
13879   \seq_get_right:NN
13880   \g_@@_jekyll_data_wildcard_absolute_address_seq
13881   \g_@@_jekyll_data_wildcard_relative_address_tl
13882 }

```

To make sure that the stacks and token lists stay in sync, we will use the `\@@_jekyll_data_push:nN` and `\@@_jekyll_data_pop:` macros.

```

13883 \cs_new:Nn \@@_jekyll_data_push:nN
13884 {
13885   \@@_jekyll_data_push_address_segment:n
13886   { #1 }
13887   \seq_put_right:NV
13888   \g_@@_jekyll_data_datatypes_seq
13889   #2
13890   \@@_jekyll_data_update_address_tls:
13891 }
13892 \cs_new:Nn \@@_jekyll_data_pop:
13893 {
13894   \seq_pop_right:NN
13895   \g_@@_jekyll_data_wildcard_absolute_address_seq
13896   \l_tmpa_tl
13897   \seq_pop_right:NN
13898   \g_@@_jekyll_data_datatypes_seq
13899   \l_tmpa_tl
13900   \@@_jekyll_data_update_address_tls:
13901 }

```

To set a single key–value, we will use the `\@@_jekyll_data_set_keyval_known:nn` macro, ignoring unknown keys. To set key–values for both absolute and relative wildcards, we will use the `\@@_jekyll_data_set_keyvals_known:nn` macro.

```

13902 \cs_new:Nn \@@_jekyll_data_set_keyval_known:nn
13903 {
13904   \keys_set_known:nn
13905   { markdown/jekyllData }
13906   { { #1 } = { #2 } }
13907 }
13908 \cs_generate_variant:Nn
13909   \@@_jekyll_data_set_keyval_known:nn
13910   { Vn }
13911 \cs_new:Nn \@@_jekyll_data_set_keyvals_known:nn
13912 {

```

```

13913     \@@_jekyll_data_push:nN
13914     { #1 }
13915     \c_@@_jekyll_data_scalar_tl
13916     \@@_jekyll_data_set_keyval_known:Vn
13917     \g_@@_jekyll_data_wildcard_absolute_address_tl
13918     { #2 }
13919     \@@_jekyll_data_set_keyval_known:Vn
13920     \g_@@_jekyll_data_wildcard_relative_address_tl
13921     { #2 }
13922     \@@_jekyll_data_pop:
13923 }

```

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

```

13924 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
13925     \@@_jekyll_data_push:nN
13926     { #1 }
13927     \c_@@_jekyll_data_sequence_tl
13928 }
13929 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
13930     \@@_jekyll_data_push:nN
13931     { #1 }
13932     \c_@@_jekyll_data_mapping_tl
13933 }
13934 \def\markdownRendererJekyllDataSequenceEndPrototype{
13935     \@@_jekyll_data_pop:
13936 }
13937 \def\markdownRendererJekyllDataMappingEndPrototype{
13938     \@@_jekyll_data_pop:
13939 }
13940 \def\markdownRendererJekyllDataBooleanPrototype#1#2{
13941     \@@_jekyll_data_set_keyvals_known:nn
13942     { #1 }
13943     { #2 }
13944 }
13945 \def\markdownRendererJekyllDataEmptyPrototype#1{}
13946 \def\markdownRendererJekyllDataNumberPrototype#1#2{
13947     \@@_jekyll_data_set_keyvals_known:nn
13948     { #1 }
13949     { #2 }
13950 }

```

We will process all string scalar values assuming that they may contain markdown markup and are intended for typesetting.

```

13951 \def\markdownRendererJekyllDataProgrammaticStringPrototype#1#2{}
13952 \def\markdownRendererJekyllDataTypographicStringPrototype#1#2{
13953     \@@_jekyll_data_set_keyvals_known:nn
13954     { #1 }

```

```

13955     { #2 }
13956 }
13957 \ExplSyntaxOff

```

### 3.2.3.3 Complex YAML Metadata Renderer Prototypes

In this section, we implement the high-level interface for routing complex YAML metadata to expl3 key-values using the option `jeekyllDataKeyValue=<module>`. See also Section 2.2.6.1.

```

13958 \ExplSyntaxOn
13959 \@@_with_various_cases:nn
13960 { jeekyllDataKeyValue }
13961 {
13962   \keys_define:nn
13963     { markdown/options }
13964     {
13965       #1 .code:n = {
13966         \@@_route_jeekyll_data_to_key_values:n
13967         { ##1 }
13968       },

```

When no `<module>` has been provided, assume that the YAML metadata specify absolute paths to key-values.

```

13969       #1 .default:n = { },
13970     }
13971   }
13972 \seq_new:N
13973 \l_@@_jeekyll_data_current_position_seq
13974 \tl_new:N
13975 \l_@@_jeekyll_data_current_position_tl
13976 \cs_new:Nn
13977   \@@_route_jeekyll_data_to_key_values:n
13978   {
13979     \markdownSetup
13980     {
13981       renderers = {
13982         jeekyllData(Sequence|Mapping)Begin = {
13983           \bool_lazy_and:nnTF
13984             {
13985               \seq_if_empty_p:N
13986                 \l_@@_jeekyll_data_current_position_seq
13987             }
13988             {
13989               \str_if_eq_p:nn
13990                 { ##1 }
13991                 { null }
13992             }

```

```

13993         {
13994             \tl_if_empty:nF
13995             { #1 }
13996             {
13997                 \seq_put_right:Nn
13998                 \l_@@_jekyll_data_current_position_seq
13999                 { #1 }
14000             }
14001         }
14002         {
14003             \seq_put_right:Nn
14004             \l_@@_jekyll_data_current_position_seq
14005             { ##1 }
14006         }
14007     },
14008     jekyllData(Sequence|Mapping)End = {
14009         \seq_pop_right:NN
14010         \l_@@_jekyll_data_current_position_seq
14011         \l_tmpa_tl
14012     },

```

For every YAML key `path.to.<key>` with a value of type *<non-string type>*, set the key *<non-string type>* of the key–value *<module>/path/to/<key>* if it is known and the key *<key>* of the key–value *<module>/path/to* otherwise. *<Non-string type>* is one of `boolean`, `number`, and `empty`.

```

14013     jekyllDataBoolean = {
14014         \tl_set:Nx
14015         \l_@@_jekyll_data_current_position_tl
14016         {
14017             \seq_use:Nn
14018             \l_@@_jekyll_data_current_position_seq
14019             { / }
14020         }
14021         \keys_if_exist:VnTF
14022         \l_@@_jekyll_data_current_position_tl
14023         { ##1 / boolean }
14024         {
14025             \@@_keys_set:xn
14026             {
14027                 \tl_use:N
14028                 \l_@@_jekyll_data_current_position_tl
14029                 / ##1 / boolean
14030             }
14031             { ##2 }
14032         }
14033         {
14034             \@@_keys_set:xn

```

```

14035         {
14036             \tl_use:N
14037             \l_@@_jekyll_data_current_position_tl
14038             / ##1
14039         }
14040         { ##2 }
14041     }
14042 },
14043 jekyllDataNumber = {
14044     \tl_set:Nx
14045     \l_@@_jekyll_data_current_position_tl
14046     {
14047         \seq_use:Nn
14048         \l_@@_jekyll_data_current_position_seq
14049         { / }
14050     }
14051     \keys_if_exist:VnTF
14052     \l_@@_jekyll_data_current_position_tl
14053     { ##1 / number }
14054     {
14055         \@@_keys_set:xn
14056         {
14057             \tl_use:N
14058             \l_@@_jekyll_data_current_position_tl
14059             / ##1 / number
14060         }
14061         { ##2 }
14062     }
14063     {
14064         \@@_keys_set:xn
14065         {
14066             \tl_use:N
14067             \l_@@_jekyll_data_current_position_tl
14068             / ##1
14069         }
14070         { ##2 }
14071     }
14072 },

```

For the  $\langle non-string\ type \rangle$  of `empty`, no value is passed to the key–value. Therefore, a default value should always be defined for nullable keys using the key property `.default:n`.

```

14073 jekyllDataEmpty = {
14074     \tl_set:Nx
14075     \l_@@_jekyll_data_current_position_tl
14076     {
14077         \seq_use:Nn

```



```

14078         \l_@@_jekyll_data_current_position_seq
14079         { / }
14080     }
14081     \keys_if_exist:VnTF
14082     \l_@@_jekyll_data_current_position_tl
14083     { ##1 / empty }
14084     {
14085         \keys_set:xn
14086         {
14087             \tl_use:N
14088             \l_@@_jekyll_data_current_position_tl
14089             / ##1
14090         }
14091         { empty }
14092     }
14093     {
14094         \keys_set:Vn
14095         \l_@@_jekyll_data_current_position_tl
14096         { ##1 }
14097     }
14098 },

```

For every YAML key `path.to.<key>` with a value of type `string`, set the keys `typographicString` and `programmaticString` of the key–value `<module>/path/to/<key>` if they are known with the typographic and programmatic strings of the value, respectively. Furthermore, set the key `<key>` of the key–value `<module>/path/to` with the typographic string of the value unless the key `typographicString` is known. If the key `programmaticString` is known, only set the key `<key>` if it is known. In contrast, if neither `typographicString` nor `programmaticString` are known, set `<key>` normally, i.e. regardless of whether it is known or unknown.

```

14099     jekyllDataTypographicString = {
14100         \tl_set:Nx
14101         \l_@@_jekyll_data_current_position_tl
14102         {
14103             \seq_use:Nn
14104             \l_@@_jekyll_data_current_position_seq
14105             { / }
14106         }
14107         \keys_if_exist:VnTF
14108         \l_@@_jekyll_data_current_position_tl
14109         { ##1 / typographicString }
14110         {
14111             \@@_keys_set:xn
14112             {
14113                 \tl_use:N

```

```

14114         \l_@@_jekyll_data_current_position_tl
14115         / ##1 / typographicString
14116     }
14117     { ##2 }
14118 }
14119 {
14120     \keys_if_exist:VnTF
14121     \l_@@_jekyll_data_current_position_tl
14122     { ##1 / programmaticString }
14123     {
14124         \@@_keys_set_known:xn
14125         {
14126             \tl_use:N
14127             \l_@@_jekyll_data_current_position_tl
14128             / ##1
14129         }
14130         { ##2 }
14131     }
14132     {
14133         \@@_keys_set:xn
14134         {
14135             \tl_use:N
14136             \l_@@_jekyll_data_current_position_tl
14137             / ##1
14138         }
14139         { ##2 }
14140     }
14141 }
14142 },
14143 jekyllDataProgrammaticString = {
14144     \tl_set:Nx
14145     \l_@@_jekyll_data_current_position_tl
14146     {
14147         \seq_use:Nn
14148         \l_@@_jekyll_data_current_position_seq
14149         { / }
14150     }
14151     \keys_if_exist:VnT
14152     \l_@@_jekyll_data_current_position_tl
14153     { ##1 / programmaticString }
14154     {
14155         \@@_keys_set:xn
14156         {
14157             \tl_use:N
14158             \l_@@_jekyll_data_current_position_tl
14159             / ##1 / programmaticString
14160         }

```

```

14161             { ##2 }
14162         }
14163     },
14164 },
14165 }
14166 }
14167 \cs_new:Nn
14168   \@@_keys_set:nn
14169   {
14170     \keys_set:nn
14171     { }
14172     { { #1 } = { #2 } }
14173   }
14174 \cs_new:Nn
14175   \@@_keys_set_known:nn
14176   {
14177     \keys_set_known:nn
14178     { }
14179     { { #1 } = { #2 } }
14180   }
14181 \cs_generate_variant:Nn
14182   \@@_keys_set:nn
14183   { xn }
14184 \cs_generate_variant:Nn
14185   \@@_keys_set_known:nn
14186   { xn }
14187 \cs_generate_variant:Nn
14188   \keys_set:nn
14189   { xn, Vn }
14190 \prg_generate_conditional_variant:Nnn
14191   \keys_if_exist:nn
14192   { Vn }
14193   { T, TF }
14194 \ExplSyntaxOff

```

If plain T<sub>E</sub>X is the top layer, we load the [witiko/markdown/defaults](#) plain T<sub>E</sub>X theme with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```

14195 \ExplSyntaxOn
14196 \str_if_eq:VVT
14197   \c_@@_top_layer_tl
14198   \c_@@_option_layer_plain_tex_tl
14199   {
14200     \use:c
14201     { ExplSyntaxOff }
14202     \@@_if_option:nF
14203     { noDefaults }

```

```

14204     {
14205         \@@_if_option:nTF
14206         { experimental }
14207         {
14208             \@@_setup:n
14209             { theme = witiko/markdown/defaults@experimental }
14210         }
14211         {
14212             \@@_setup:n
14213             { theme = witiko/markdown/defaults }
14214         }
14215     }
14216     \use:c
14217     { ExplSyntaxOn }
14218 }
14219 \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  $\text{\TeX}$  options (see Section 2.2.2) in a format recognized by Lua (see Section 2.1.3).

```

14220 \ExplSyntaxOn
14221 \tl_new:N \g_@@_formatted_lua_options_tl
14222 \cs_new:Nn \@@_format_lua_options:
14223 {
14224     \tl_gclear:N
14225     \g_@@_formatted_lua_options_tl
14226     \seq_map_function:NN
14227     \g_@@_lua_options_seq
14228     \@@_format_lua_option:n
14229 }
14230 \cs_new:Nn \@@_format_lua_option:n
14231 {
14232     \@@_typecheck_option:n
14233     { #1 }
14234     \@@_get_option_type:nN
14235     { #1 }
14236     \l_tmpa_tl
14237     \bool_case_true:nF
14238     {
14239         {
14240             \str_if_eq_p:VV
14241             \l_tmpa_tl
14242             \c_@@_option_type_boolean_tl ||
14243             \str_if_eq_p:VV

```

```

14244         \l_tmpa_tl
14245         \c_@@_option_type_number_tl ||
14246     \str_if_eq_p:VV
14247         \l_tmpa_tl
14248         \c_@@_option_type_counter_tl
14249     }
14250     {
14251         \@@_get_option_value:nN
14252         { #1 }
14253         \l_tmpa_tl
14254         \tl_gput_right:Nx
14255         \g_@@_formatted_lua_options_tl
14256         { #1~~~ \l_tmpa_tl ,~ }
14257     }
14258     {
14259         \str_if_eq_p:VV
14260         \l_tmpa_tl
14261         \c_@@_option_type_clist_tl
14262     }
14263     {
14264         \@@_get_option_value:nN
14265         { #1 }
14266         \l_tmpa_tl
14267         \tl_gput_right:Nx
14268         \g_@@_formatted_lua_options_tl
14269         { #1~~~\c_left_brace_str }
14270         \clist_map_inline:Vn
14271         \l_tmpa_tl
14272         {
14273             \@@_lua_escape:xN
14274             { ##1 }
14275             \l_tmpb_tl
14276             \tl_gput_right:Nn
14277             \g_@@_formatted_lua_options_tl
14278             { " }
14279             \tl_gput_right:NV
14280             \g_@@_formatted_lua_options_tl
14281             \l_tmpb_tl
14282             \tl_gput_right:Nn
14283             \g_@@_formatted_lua_options_tl
14284             { " ,~ }
14285         }
14286         \tl_gput_right:Nx
14287         \g_@@_formatted_lua_options_tl
14288         { \c_right_brace_str ,~ }
14289     }
14290 }

```

```

14291     {
14292         \@@_get_option_value:nN
14293         { #1 }
14294         \l_tmpa_tl
14295         \@@_lua_escape:xN
14296         { \l_tmpa_tl }
14297         \l_tmpb_tl
14298         \tl_gput_right:Nn
14299         \g_@@_formatted_lua_options_tl
14300         { #1~~ " }
14301         \tl_gput_right:NV
14302         \g_@@_formatted_lua_options_tl
14303         \l_tmpb_tl
14304         \tl_gput_right:Nn
14305         \g_@@_formatted_lua_options_tl
14306         { " ,~ }
14307     }
14308 }
14309 \cs_generate_variant:Nn
14310 \clist_map_inline:nn
14311 { Vn }
14312 \let
14313 \markdownPrepareLuaOptions
14314 \@@_format_lua_options:
14315 \def
14316 \markdownLuaOptions
14317 {
14318     {
14319         \g_@@_formatted_lua_options_tl
14320     }
14321 }
14322 \sys_if_engine luatex:TF
14323 {
14324     \cs_new:Nn
14325     \@@_lua_escape:nN
14326     {
14327         \tl_set:Nx
14328         #2
14329         {
14330             \lua_escape:n
14331             { #1 }
14332         }
14333     }
14334 }
14335 {
14336     \regex_const:Nn
14337     \c_@@_lua_escape_regex

```

```

14338     { [\\\"'] }
14339   \cs_new:Nn
14340     \@@_lua_escape:nN
14341     {
14342       \tl_set:Nn
14343         #2
14344         { #1 }
14345       \regex_replace_all:NnN
14346         \c_@@_lua_escape_regex
14347         { \u { c_backslash_str } \0 }
14348         #2
14349     }
14350   }
14351 \cs_generate_variant:Nn
14352   \@@_lua_escape:nN
14353   { xN }

```

After the `\markdownPrepareInputFilename` macro has been fully expanded, the `\markdownInputFilename` macro will expand to a Lua string that contains the input filename passed as the first argument.

```

14354 \tl_new:N
14355   \markdownInputFilename
14356 \cs_new:Npn
14357   \markdownPrepareInputFilename
14358   #1
14359   {
14360     \@@_lua_escape:xN
14361     { #1 }
14362     \markdownInputFilename
14363     \tl_gset:Nx
14364     \markdownInputFilename
14365     { " \markdownInputFilename " }
14366   }

```

The `\markdownPrepare` macro contains the Lua code that is executed prior to any conversion from markdown to plain T<sub>E</sub>X. It exposes the `convert` function for the use by any further Lua code.

```

14367 \cs_new:Npn
14368   \markdownPrepare
14369   {

```

First, ensure that the `cacheDir` directory exists.

```

14370     local~lfs = require("lfs")
14371     local~options = \markdownLuaOptions
14372     if~not~lfs.isdir(options.cacheDir) then~
14373       assert(lfs.mkdir(options.cacheDir))
14374     end~

```

Next, load the `markdown` module and create a converter function using the plain  $\text{\TeX}$  options, which were serialized to a Lua table via the `\markdownLuaOptions` macro.

```
14375     local~md = require("markdown")
14376     local~convert = md.new(options)
14377 }
```

The `\markdownConvert` macro contains the Lua code that is executed during the conversion from markdown to plain  $\text{\TeX}$ . It opens the input file, converts it, and prints the conversion result.

```
14378 \cs_new:Npn
14379   \markdownConvert
14380 {
14381   local~filename = \markdownInputFilename
14382   local~file = assert(io.open(filename, "r"),
14383     [[Could~not~open~file~]] .. filename .. [[~for~reading]])
14384   local~input = assert(file:read("*a"))
14385   assert(file:close())
14386   print(convert(input))
14387 }
14388 \ExplSyntaxOff
```

The `\markdownCleanup` macro contains the Lua code that is executed after any conversion from markdown to plain  $\text{\TeX}$ .

```
14389 \def\markdownCleanup{%
Remove the options.cacheDir directory if it is empty.
14390   if options.cacheDir then
14391     lfs.rmdir(options.cacheDir)
14392   end
14393 }%
```

### 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.

```
14394 \csname newread\endcsname\markdownInputFileStream
14395 \csname newwrite\endcsname\markdownOutputFileStream
```

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

```
14396 \begingroup
14397   \catcode\^^I=12%
14398   \gdef\markdownReadAndConvertTab{^^I}%
14399 \endgroup
```

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

```
14400 \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.

```
14401 \catcode\^M=13%
14402 \catcode\^I=13%
14403 \catcode|=0%
14404 \catcode\=12%
14405 |catcode@=14%
14406 |catcode|=12@
14407 |gdef|markdownReadAndConvert#1#2{@
14408   |begingroup@
```

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

```
14409   |markdownIfOption{frozenCache}{-}{@
14410     |immediate|openout|markdownOutputFileStream@
14411     "|markdownOptionInputTempFileName"|relax@
14412   |markdownInfo{@
14413     Buffering block-level markdown input into the temporary @
14414     input file "|markdownOptionInputTempFileName" and scanning @
14415     for the closing token sequence "#1"}@
14416   }@
```

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.

```
14417   |def|do##1{|catcode##1=12}|dospecials@
14418   |catcode|=12@
14419   |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.

```
14420   |def|markdownReadAndConvertStripPercentSign##1{@
14421     |markdownIfOption{stripPercentSigns}{-}{@
14422       |if##1%@
14423         |expandafter|expandafter|expandafter@
14424         |markdownReadAndConvertProcessLine@
14425       |else@
14426         |expandafter|expandafter|expandafter@
14427         |markdownReadAndConvertProcessLine@
14428         |expandafter|expandafter|expandafter##1@
14429       |fi@
14430     }{@
```

```

14431      |expandafter@
14432      |markdownReadAndConvertProcessLine@
14433      |expandafter##1@
14434    }@
14435  }@

```

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.

```

14436    |def|markdownReadAndConvertProcessLine##1#1##2#1##3|relax{@

```

If we are not reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, store the line in the `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.

```

14437      |ifx|relax##3|relax@
14438      |markdownIfOption{frozenCache}{-}{@
14439      |immediate|write|markdownOutputFileStream{##1}@
14440    }@
14441    |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.

```

14442      |def^^M{@
14443      |markdownInfo{The ending token sequence was found}@
14444      |markdownIfOption{frozenCache}{-}{@
14445      |immediate|closeout|markdownOutputFileStream@
14446    }@
14447    |endgroup@
14448    |markdownInput{@
14449      |markdownOptionOutputDir@
14450      /|markdownOptionInputTempFileName@
14451    }@
14452    #2}@
14453  |fi@

```

Repeat with the next line.

```

14454    ^^M}@

```

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

```

14455    |catcode`|^I=13@
14456    |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.

```

14457 |catcode`|^M=13@
14458 |def^^M##1^^M{@
14459 |def^^M###1^^M{@
14460 |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
14461 ^^M}@
14462 ^^M}@

```

Reset the character categories back to the former state.

```
14463 |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.

```

14464 \ExplSyntaxOn
14465 \cs_new:Npn
14466   \markdownLuaExecute
14467   #1
14468   {
14469     \int_compare:nNtT
14470       { \g_luabridge_method_int }
14471       =
14472       { \c_luabridge_method_shell_int }
14473       {
14474         \sys_if_shell_unrestricted:F
14475         {
14476           \sys_if_shell:TF
14477           {
14478             \msg_error:nn
14479               { markdown }
14480               { restricted-shell-access }
14481           }
14482           {
14483             \msg_error:nn
14484               { markdown }
14485               { disabled-shell-access }
14486           }
14487         }
14488       }
14489     \str_gset:NV
14490       \g_luabridge_output_dirname_str
14491       \markdownOptionOutputDir
14492     \luabridge_now:e
14493       { #1 }
14494   }
14495 \cs_generate_variant:Nn
14496   \msg_new:nnnn
14497   { nnnV }
14498 \tl_set:Nn
14499   \l_tmpa_tl

```

```

14500 {
14501     You~may~need~to~run~TeX~with~the~---shell-escape~or~the~
14502     --enable-write18~flag,~or~write~shell_escape=t~in~the~
14503     texmf.cnf~file.
14504 }
14505 \msg_new:nnnV
14506 { markdown }
14507 { restricted-shell-access }
14508 { Shell-escape-is-restricted }
14509 \l_tmpa_tl
14510 \msg_new:nnnV
14511 { markdown }
14512 { disabled-shell-access }
14513 { Shell-escape-is-disabled }
14514 \l_tmpa_tl
14515 \ExplSyntaxOff

```

### 3.2.6 Buffering Inline Markdown Input

This section describes the implementation of the macro `\markinline`.

```

14516 \ExplSyntaxOn
14517 \tl_new:N
14518   \g_@@_after_markinline_tl
14519 \tl_gset:Nn
14520   \g_@@_after_markinline_tl
14521   { \unskip }
14522 \cs_new:Npn
14523   \markinline
14524   {

```

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.

```

14525     \group_begin:
14526     \cctab_select:N
14527     \c_other_cctab

```

Unless we are reading markdown documents from the frozen cache, open the file `inputTempFileName` for writing.

```

14528     \@@_if_option:nF
14529     { frozenCache }
14530     {
14531         \immediate
14532         \openout
14533         \markdownOutputFileStream
14534         "\markdownOptionInputTempFileName"
14535         \relax
14536         \msg_info:nne

```

```

14537         { markdown }
14538         { buffering-markinline }
14539         { \markdownOptionInputTempFileName }
14540     }

```

Peek ahead and extract the inline markdown text.

```

14541     \peek_regex_replace_once:nnF
14542     { { (.*) } }
14543     {

```

Unless we are reading markdown documents from the frozen cache, store the text in the file `inputTempFileName` and close it.

```

14544         \c { @@_if_option:nF }
14545         \cB { frozenCache \cE }
14546         \cB {
14547             \c { immediate }
14548             \c { write }
14549             \c { markdownOutputFileStream }
14550             \cB { \1 \cE }
14551             \c { immediate }
14552             \c { closeout }
14553             \c { markdownOutputFileStream }
14554         \cE }

```

Reset the category codes and `\input` the result of the conversion.

```

14555         \c { group_end: }
14556         \c { group_begin: }
14557         \c { @@_setup:n }
14558         \cB { contentLevel = inline \cE }
14559         \c { markdownInput }
14560         \cB {
14561             \c { markdownOptionOutputDir } /
14562             \c { markdownOptionInputTempFileName }
14563         \cE }
14564         \c { group_end: }
14565         \c { tl_use:N }
14566         \c { g_@@_after_markinline_tl }
14567     }
14568     {
14569         \msg_error:nn
14570         { markdown }
14571         { markinline-peek-failure }
14572         \group_end:
14573         \tl_use:N
14574         \g_@@_after_markinline_tl
14575     }
14576 }
14577 \msg_new:nnn

```

```

14578 { markdown }
14579 { buffering-markinline }
14580 { Buffering~inline~markdown~input~into~
14581   the~temporary~input~file~"#1". }
14582 \msg_new:nnnn
14583 { markdown }
14584 { markinline-peek-failure }
14585 { Use~of~\iow_char:N \\ markinline~doesn't~match~its~definition }
14586 { The~macro~should~be~followed~by~inline~
14587   markdown~text~in~curly~braces }
14588 \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.

```

14589 \ExplSyntaxOn
14590 \cs_new:Npn
14591   \markdownInput
14592   #1
14593   {
14594     \@@_if_option:nTF
14595       { frozenCache }
14596       {
14597         \markdownInputRaw
14598         { #1 }
14599       }
14600   {

```

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`.

```

14601     \tl_set:Nx
14602       \l_tmpa_tl
14603       { #1 }
14604     \file_get_full_name:VNTF
14605       \l_tmpa_tl
14606       \l_tmpb_tl
14607     {
14608       \exp_args:NV
14609         \markdownInputRaw
14610         \l_tmpb_tl
14611     }
14612     {
14613       \msg_error:nnV
14614         { markdown }

```

```

14615             { markdown-file-does-not-exist }
14616             \l_tmpa_tl
14617         }
14618     }
14619 }
14620 \msg_new:nnn
14621 { markdown }
14622 { markdown-file-does-not-exist }
14623 {
14624     Markdown~file~#1~does~not~exist
14625 }
14626 \ExplSyntaxOff
14627 \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.

```

14628 \catcode`\=0%
14629 \catcode`\=12%
14630 \catcode`\&=6%
14631 \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.

```

14632 |begingroup
14633 |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.

```

14634 |catcode`\#=12

```

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

```

14635 |markdownIfOption{frozenCache}{%
14636 |ifnum|markdownOptionFrozenCacheCounter=0|relax
14637 |markdownInfo{Reading frozen cache from
14638 "||markdownOptionFrozenCacheFileName"}%
14639 |input|markdownOptionFrozenCacheFileName|relax
14640 |fi
14641 |markdownInfo{Including markdown document number
14642 "|the|markdownOptionFrozenCacheCounter" from frozen cache}%
14643 |csname markdownFrozenCache%
14644 |the|markdownOptionFrozenCacheCounter|endcsname
14645 |global|advance|markdownOptionFrozenCacheCounter by 1|relax
14646 }{%
14647 |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  $\text{\LaTeX}$ Mk to track changes to the markdown document.

```
14648      |openin|markdownInputFileStream{&1}%
14649      |closein|markdownInputFileStream
14650      |markdownPrepareLuaOptions
14651      |markdownPrepareInputFilename{&1}%
14652      |markdownLuaExecute{%
14653          |markdownPrepare
14654          |markdownConvert
14655          |markdownCleanup}%
```

If we are finalizing the frozen cache, increment `frozenCacheCounter`.

```
14656      |markdownIfOption{finalizeCache}{}%
14657      |global|advance|markdownOptionFrozenCacheCounter by 1|relax}{}%
14658      }%
14659      |endgroup
14660      }%
14661 |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  $\text{\TeX}$  to execute a  $\text{\TeX}$  document in the middle of a markdown document fragment.

```
14662 \gdef\markdownEscape#1{%
14663   \catcode`\%=14\relax
14664   \catcode`\#=6\relax
14665   \input #1\relax
14666   \catcode`\%=12\relax
14667   \catcode`\#=12\relax
14668 }%
```

### 3.3 $\text{\LaTeX}$ Implementation

The  $\text{\LaTeX}$  implementation makes use of the fact that, apart from some subtle differences,  $\text{\LaTeX}$  implements the majority of the plain  $\text{\TeX}$  format [17, Section 9]. As a consequence, we can directly reuse the existing plain  $\text{\TeX}$  implementation.

```
14669 \def\markdownVersionSpace{ }%
14670 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
14671   \markdownVersion\markdownVersionSpace markdown renderer]%
```

#### 3.3.1 Typesetting Markdown

The `\markinlinePlainTeX` macro is used to store the original plain  $\text{\TeX}$  implementation of the `\markinline` macro. The `\markinline` macro is then redefined to



accept an optional argument with options recognized by the L<sup>A</sup>T<sub>E</sub>X interface (see Section 2.3.3).

```

14672 \ExplSyntaxOn
14673 \cs_gset_eq:NN
14674   \markinlinePlainTeX
14675   \markinline
14676 \cs_gset:Npn
14677   \markinline
14678   {
14679     \peek_regex_replace_once:nn
14680     { ( \[ (.*) \] ) ? }
14681     {

```

Apply the options locally.

```

14682       \c { group_begin: }
14683       \c { @@_setup:n }
14684       \cB { \2 \cE }
14685       \c { tl_put_right:Nn }
14686       \c { g_@@_after_markinline_tl }
14687       \cB { \c { group_end: } \cE }
14688       \c { markinlinePlainTeX }
14689     }
14690   }
14691 \ExplSyntaxOff

```

The `\markdownInputPlainTeX` macro is used to store the original plain T<sub>E</sub>X implementation of the `\yamlInput` macro. The `\markdownInput` and `\yamlInput` macros are then redefined to accept an optional argument with options recognized by the L<sup>A</sup>T<sub>E</sub>X interface (see Section 2.3.3).

```

14692 \let\markdownInputPlainTeX\markdownInput
14693 \renewcommand\markdownInput[2][]{%
14694   \begingroup
14695     \markdownSetup{#1}%
14696     \markdownInputPlainTeX{#2}%
14697   \endgroup}%
14698 \renewcommand\yamlInput[2][]{%
14699   \begingroup
14700     \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData, #1}%
14701     \markdownInputPlainTeX{#2}%
14702   \endgroup}%

```

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

```

14703 \ExplSyntaxOn
14704 \renewenvironment
14705   { markdown }
14706   {

```

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.

```
14707 \group_begin:
14708 \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).

```
14709 \char_set_catcode_letter:n { 35 }
```

After we have matched the opening `[` that begins the optional argument, we accept carriage returns as well.

```
14710 \peek_regex_replace_once:nnF
14711 { \ *\[ \r*([~]*)\][^~\r]* }
14712 {
```

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.

```
14713 \c { group_end: }
14714 \c { tl_set_rescan:Nnn } \c { l_tmpa_tl } { } { \1 }
```

Then, we pass the retokenized content to the `\markdownSetup` macro.

```
14715 \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  $\text{\LaTeX}$  environment.

We also make provision for using the `\markdown` command as a part of a different  $\text{\LaTeX}$  environment as follows:

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

```
14716     \c { exp_args:NV }
14717     \c { markdownReadAndConvert@ }
14718     \c { @currenvir }
14719 }
14720 {
14721   \group_end:
14722   \exp_args:NV
14723   \markdownReadAndConvert@
14724   \@currenvir
14725 }
14726 }
14727 { \markdownEnd }
14728 \renewenvironment
14729 { markdown* }
14730 [ 1 ]
14731 {
14732   \@@_if_option:nTF
14733   { experimental }
14734   {
14735     \msg_error:nnn
14736     { markdown }
14737     { latex-markdown-star-deprecated }
14738     { #1 }
14739   }
14740   {
14741     \msg_warning:nnn
14742     { markdown }
14743     { latex-markdown-star-deprecated }
14744     { #1 }
14745   }
14746   \@@_setup:n
14747   { #1 }
14748   \markdownReadAndConvert@
14749   { markdown* }
14750 }
14751 { \markdownEnd }
14752 \renewenvironment
```

```

14753 { yaml }
14754 {
14755   \group_begin:
14756   \yamlSetup
14757     { jekyllData, expectJekyllData, ensureJekyllData }
14758   \markdown
14759 }
14760 { \yamlEnd }
14761 \msg_new:nnn
14762 { markdown }
14763 { latex-markdown-star-deprecated }
14764 {
14765   The~markdown*~LaTeX~environment~has~been~deprecated~and~will~
14766   be~removed~in~the~next~major~version~of~the~Markdown~package.
14767 }
14768 \cs_generate_variant:Nn % noqa: e405, w402
14769 \@@_setup:n
14770 { V }
14771 \ExplSyntaxOff
14772 \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*.

```

14773 \catcode`\|=0\catcode`\<=1\catcode`\>=2%
14774 \catcode`\|=12\catcode`\{=12\catcode`\}=12%
14775 |gdef|markdownReadAndConvert@#1<%
14776   |markdownReadAndConvert<\end{#1}>%
14777   <|end<#1>>>%
14778 |endgroup

```

### 3.3.2 Themes

This section overrides the plain  $\mathrm{T}_{\mathrm{E}}\mathrm{X}$  implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in  $\mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$  themes provided with the Markdown package.

```

14779 \ExplSyntaxOn
14780 \prop_new:N \g_@@_latex_loaded_themes_linenos_prop
14781 \prop_new:N \g_@@_latex_loaded_themes_versions_prop
14782 \cs_gset:Nn % noqa: w401
14783 \@@_load_theme:nnn
14784 {

```

If the Markdown package has not yet been loaded, determine whether either this is a built-in theme according to the prop `\g_@@_latex_built_in_themes_prop` or

a file named `markdowntheme<munged theme name>.sty` exists and whether we are still in the preamble.

```
14785 \ifmarkdownLaTeXLoaded
14786 \ifx\@onlypreamble\@notprerr
```

If both conditions are true, end with an error, since we cannot load L<sup>A</sup>T<sub>E</sub>X themes after the preamble.

```
14787 \bool_if:nTF
14788 {
14789 \bool_lazy_or_p:nn
14790 {
14791 \prop_if_in_p:Nn
14792 \g_@@_latex_built_in_themes_prop
14793 { #1 }
14794 }
14795 {
14796 \file_if_exist_p:n
14797 { markdown theme #3.sty }
14798 }
14799 }
14800 {
14801 \msg_error:nnn
14802 { markdown }
14803 { latex-theme-after-preamble }
14804 { #1 }
14805 }
```

Otherwise, try loading a plain T<sub>E</sub>X theme instead.

```
14806 {
14807 \@@_plain_tex_load_theme:nnn
14808 { #1 }
14809 { #2 }
14810 { #3 }
14811 }
14812 \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.

```
14813 \bool_if:nTF
14814 {
14815 \bool_lazy_or_p:nn
14816 {
14817 \prop_if_in_p:Nn
14818 \g_@@_latex_built_in_themes_prop
14819 { #1 }
14820 }
14821 {
14822 \file_if_exist_p:n
```

```

14823         { markdown theme #3.sty }
14824     }
14825 }
14826 {
14827     \prop_get:NnNTF
14828     \g_@@_latex_loaded_themes_linenos_prop
14829     { #1 }
14830     \l_tmpa_tl
14831     {
14832         \prop_get:NnN
14833         \g_@@_latex_loaded_themes_versions_prop
14834         { #1 }
14835         \l_tmpb_tl
14836         \str_if_eq:nVTF
14837         { #2 }
14838         \l_tmpb_tl
14839         {
14840             \msg_warning:nnnVn
14841             { markdown }
14842             { repeatedly-loaded-latex-theme }
14843             { #1 }
14844             \l_tmpa_tl
14845             { #2 }
14846         }
14847         {
14848             \msg_error:nnnnVV
14849             { markdown }
14850             { different-versions-of-latex-theme }
14851             { #1 }
14852             { #2 }
14853             \l_tmpb_tl
14854             \l_tmpa_tl
14855         }
14856     }
14857 }
14858 \prop_gput:Nnx
14859     \g_@@_latex_loaded_themes_linenos_prop
14860     { #1 }
14861     { \tex_the:D \tex_inputlineno:D } % noqa: W200
14862 \prop_gput:Nnn
14863     \g_@@_latex_loaded_themes_versions_prop
14864     { #1 }
14865     { #2 }

```

Load built-in plain TeX themes from the prop `\g_@@_latex_built_in_themes_prop` and from the filesystem otherwise.

```

14866     \prop_if_in:NnTF

```

```

14867         \g_@@_latex_built_in_themes_prop
14868         { #1 }
14869         {
14870             \msg_info:nnnn
14871             { markdown }
14872             { loading-built-in-latex-theme }
14873             { #1 }
14874             { #2 }
14875             \prop_item:Nn
14876             \g_@@_latex_built_in_themes_prop
14877             { #1 }
14878         }
14879         {
14880             \msg_info:nnnn
14881             { markdown }
14882             { loading-latex-theme }
14883             { #1 }
14884             { #2 }
14885             \RequirePackage
14886             { markdown theme #3 }
14887         }
14888     }
14889 }
14890 {
14891     \@@_plain_tex_load_theme:nnn
14892     { #1 }
14893     { #2 }
14894     { #3 }
14895 }
14896 \fi
14897 \else

```

If the Markdown package has not yet been loaded, postpone the loading until the Markdown package has finished loading.

```

14898     \msg_info:nnnn
14899     { markdown }
14900     { theme-loading-postponed }
14901     { #1 }
14902     { #2 }
14903     \AtEndOfPackage
14904     {
14905         \@@_set_theme:n
14906         { #1 @ #2 }
14907     }
14908 \fi
14909 }
14910 \msg_new:nnn

```

```

14911 { markdown }
14912 { theme-loading-postponed }
14913 {
14914     Postponing~loading~version~#2~of~Markdown~theme~#1~until~
14915     Markdown~package~has~finished~loading
14916 }
14917 \msg_new:nnn
14918 { markdown }
14919 { loading-built-in-latex-theme }
14920 { Loading~version~#2~of~built-in~LaTeX~Markdown~theme~#1 }
14921 \msg_new:nnn
14922 { markdown }
14923 { loading-latex-theme }
14924 { Loading~version~#2~of~LaTeX~Markdown~theme~#1 }
14925 \msg_new:nnn
14926 { markdown }
14927 { repeatedly-loaded-latex-theme }
14928 {
14929     Version~#3~of~LaTeX~Markdown~theme~#1~was~previously~
14930     loaded~on~line~#2,~not~loading~it~again
14931 }
14932 \msg_new:nnn
14933 { markdown }
14934 { different-versions-of-latex-theme }
14935 {
14936     Tried~to~load~version~#2~of~LaTeX~Markdown~theme~#1~
14937     but~version~#3~has~already~been~loaded~on~line~#4
14938 }
14939 \cs_generate_variant:Nn
14940 \msg_new:nnnn
14941 { nnVV }
14942 \tl_set:Nn
14943 \l_tmpa_tl
14944 { Cannot~load~LaTeX~Markdown~theme~#1~after~ }
14945 \tl_put_right:NV
14946 \l_tmpa_tl
14947 \c_backslash_str
14948 \tl_put_right:Nn
14949 \l_tmpa_tl
14950 { begin { document } }
14951 \tl_set:Nn
14952 \l_tmpb_tl
14953 { Load~Markdown~theme~#1~before~ }
14954 \tl_put_right:NV
14955 \l_tmpb_tl
14956 \c_backslash_str
14957 \tl_put_right:Nn

```



```

14958 \l_tmpb_tl
14959 { begin { document } }
14960 \msg_new:nnVV
14961 { markdown }
14962 { latex-theme-after-preamble }
14963 \l_tmpa_tl
14964 \l_tmpb_tl

```

The [witiko/dot](#) and [witiko/graphicx/http](#) L<sup>A</sup>T<sub>E</sub>X themes load the package `graphicx`, see also Section 1.1.3. Then, they load the corresponding plain T<sub>E</sub>X themes.

```

14965 \tl_set:Nn
14966 \l_tmpa_tl
14967 {
14968   \RequirePackage
14969     { graphicx }
14970   \markdownLoadPlainTeXTheme
14971 }
14972 \prop_gput:NnV
14973 \g_@@_latex_built_in_themes_prop
14974 { witiko / dot }
14975 \l_tmpa_tl
14976 \prop_gput:NnV
14977 \g_@@_latex_built_in_themes_prop
14978 { witiko / graphicx / http }
14979 \l_tmpa_tl
14980 \ExplSyntaxOff

```

The [witiko/markdown/defaults](#) L<sup>A</sup>T<sub>E</sub>X theme also loads the corresponding plain T<sub>E</sub>X theme.

```

14981 \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.

### 3.3.3 Options

The supplied package options are processed using the `\markdownSetup` macro.

```

14982 \DeclareOption*{%
14983   \expandafter\markdownSetup\expandafter{\CurrentOption}}%
14984 \ProcessOptions\relax

```

### 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.

```

14985 \markdownIfOption{plain}{\iffalse}{\iftrue}

```

### 3.3.4.1 Lists

If either the `tightLists` or the `fancyLists` Lua option is enabled and the current document class is not beamer, use a package that provides support for tight and fancy lists.

If either the package `paralist` or the package `enumitem` have already been loaded, use them. Otherwise, if the option `experimental` or any test phase has been enabled, use the package `enumitem`. Otherwise, use the package `paralist`.

```
14986 \ExplSyntaxOn
14987 \bool_new:N
14988   \g_@@_tight_or_fancy_lists_bool
14989 \bool_gset_false:N
14990   \g_@@_tight_or_fancy_lists_bool
14991 \@@_if_option:nTF
14992   { tightLists }
14993   {
14994     \bool_gset_true:N
14995       \g_@@_tight_or_fancy_lists_bool
14996   }
14997   {
14998     \@@_if_option:nT
14999     { fancyLists }
15000     {
15001       \bool_gset_true:N
15002         \g_@@_tight_or_fancy_lists_bool
15003     }
15004   }
15005 \bool_new:N
15006   \g_@@_beamer_paralist_or_enumitem_bool
15007 \bool_gset_true:N
15008   \g_@@_beamer_paralist_or_enumitem_bool
15009 \@ifclassloaded
15010   { beamer }
15011   { }
15012   {
15013     \@ifpackageloaded
15014       { paralist }
15015       { }
15016     {
15017       \@ifpackageloaded
15018         { enumitem }
15019         { }
15020     }
15021     \bool_gset_false:N
15022       \g_@@_beamer_paralist_or_enumitem_bool
15023   }
15024 }
```

```

15025 }
15026 \bool_if:nT
15027 {
15028   \g_@@_tight_or_fancy_lists_bool &&
15029   ! \g_@@_beamer_paralist_or_enumitem_bool
15030 }
15031 {
15032   \bool_if:nTF
15033   {
15034     \bool_lazy_or_p:nn
15035     {
15036       \str_if_eq_p:en
15037       { \markdownThemeVersion }
15038       { experimental }
15039     }
15040     {
15041       \bool_lazy_and_p:nn
15042       {
15043         \prop_if_exist_p:N
15044         \g__pdfmanagement_documentproperties_prop
15045       }
15046       {
15047         \bool_lazy_any_p:n
15048         {
15049           {
15050             \prop_if_in_p:Nn
15051             \g__pdfmanagement_documentproperties_prop
15052             { document / testphase / phase-I }
15053           }
15054           {
15055             \prop_if_in_p:Nn
15056             \g__pdfmanagement_documentproperties_prop
15057             { document / testphase / phase-II }
15058           }
15059           {
15060             \prop_if_in_p:Nn
15061             \g__pdfmanagement_documentproperties_prop
15062             { document / testphase / phase-III }
15063           }
15064           {
15065             \prop_if_in_p:Nn
15066             \g__pdfmanagement_documentproperties_prop
15067             { document / testphase / phase-IV }
15068           }
15069           {
15070             \prop_if_in_p:Nn
15071             \g__pdfmanagement_documentproperties_prop

```

```

15072             { document / testphase / phase-V }
15073         }
15074     {
15075         \prop_if_in_p:Nn
15076         \g__pdfmanagement_documentproperties_prop
15077         { document / testphase / phase-VI }
15078     }
15079 }
15080 }
15081 }
15082 }
15083 {
15084     \RequirePackage
15085     { enumitem }
15086 }
15087 {
15088     \RequirePackage
15089     { paralist }
15090 }
15091 }

```

15092 \ExplSyntaxOff

If we loaded the enumitem package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```

15093 \ExplSyntaxOn
15094 \cs_new:Nn
15095   \@@_latex_fancy_list_item_label_number:nn
15096   {
15097     \str_case:nn
15098     { #1 }
15099     {
15100       { Decimal } { #2 }
15101       { LowerRoman } { \int_to_roman:n { #2 } }
15102       { UpperRoman } { \int_to_Roman:n { #2 } }
15103       { LowerAlpha } { \int_to_alph:n { #2 } }
15104       { UpperAlpha } { \int_to_Alph:n { #2 } }
15105     }
15106   }
15107 \cs_new:Nn
15108   \@@_latex_fancy_list_item_label_delimiter:n
15109   {
15110     \str_case:nn
15111     { #1 }
15112     {
15113       { Default } { . }
15114       { OneParen } { ) }
15115       { Period } { . }

```

```

15116     }
15117 }
15118 \cs_new:Nn
15119   \@@_latex_fancy_list_item_label:nnn
15120 {
15121   \@@_latex_fancy_list_item_label_number:nn
15122     { #1 }
15123     { #3 }
15124   \@@_latex_fancy_list_item_label_delimiter:n
15125     { #2 }
15126 }
15127 \cs_generate_variant:Nn
15128   \@@_latex_fancy_list_item_label:nnn
15129   { VVn }
15130 \tl_new:N
15131   \l_@@_latex_fancy_list_item_label_number_style_tl
15132 \tl_new:N
15133   \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15134 \ifpackageloaded { enumitem } {
15135   \markdownSetup { rendererPrototypes = {

```

First, let's define the tight list item renderer prototypes.

```

15136     ulBeginTight = {
15137       \begin
15138         { itemize }
15139         [ noitemsep ]
15140     },
15141     ulEndTight = {
15142       \end
15143         { itemize }
15144     },
15145     olBeginTight = {
15146       \begin
15147         { enumerate }
15148         [ noitemsep ]
15149     },
15150     olEndTight = {
15151       \end
15152         { enumerate }
15153     },
15154     dlBeginTight = {
15155       \begin
15156         { description }
15157         [ noitemsep ]
15158     },
15159     dlEndTight = {
15160       \end
15161         { description }

```

```
15162     },
```

Second, let's define the fancy list item renderer prototypes.

```
15163     fancyOlBegin = {
15164         \group_begin:
15165         \tl_set:Nn
15166             \l_@@_latex_fancy_list_item_label_number_style_tl
15167             { #1 }
15168         \tl_set:Nn
15169             \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15170             { #2 }
15171         \begin
15172             { enumerate }
15173     },
15174     fancyOlBeginTight = {
15175         \group_begin:
15176         \tl_set:Nn
15177             \l_@@_latex_fancy_list_item_label_number_style_tl
15178             { #1 }
15179         \tl_set:Nn
15180             \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15181             { #2 }
15182         \begin
15183             { enumerate }
15184             [ noitemsep ]
15185     },
15186     fancyOlEnd(|Tight) = {
15187         \end { enumerate }
15188         \group_end:
15189     },
15190     fancyOlItemWithNumber = {
15191         \item
15192         [
15193             \@@_latex_fancy_list_item_label:VVn
15194             \l_@@_latex_fancy_list_item_label_number_style_tl
15195             \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15196             { #1 }
15197         ]
15198     },
15199 } }
```

Otherwise, if we loaded the paralist package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```
15200 }
15201 { \ifpackageloaded { paralist } {
15202     \markdownSetup { rendererPrototypes = {
```

Make tight bullet lists a little less compact by adding extra vertical space above and below them.

```

15203     ulBeginTight = {
15204         \group_begin:
15205         \pltopsep=\topsep
15206         \plpartopsep=\partopsep
15207         \begin { compactitem }
15208     },
15209     ulEndTight = {
15210         \end { compactitem }
15211         \group_end:
15212     },
15213     fancyOlBegin = {
15214         \group_begin:
15215         \tl_set:Nn
15216             \l_@@_latex_fancy_list_item_label_number_style_tl
15217             { #1 }
15218         \tl_set:Nn
15219             \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15220             { #2 }
15221         \begin { enumerate }
15222     },
15223     fancyOlEnd = {
15224         \end { enumerate }
15225         \group_end:
15226     },

```

Make tight ordered lists a little less compact by adding extra vertical space above and below them.

```

15227     olBeginTight = {
15228         \group_begin:
15229         \plpartopsep=\partopsep
15230         \pltopsep=\topsep
15231         \begin { compactenum }
15232     },
15233     olEndTight = {
15234         \end { compactenum }
15235         \group_end:
15236     },
15237     fancyOlBeginTight = {
15238         \group_begin:
15239         \tl_set:Nn
15240             \l_@@_latex_fancy_list_item_label_number_style_tl
15241             { #1 }
15242         \tl_set:Nn
15243             \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15244             { #2 }

```

```

15245     \plpartopsep=\partopsep
15246     \pltopsep=\topsep
15247     \begin { compactenum }
15248 },
15249 fancyOlEndTight = {
15250     \end { compactenum }
15251     \group_end:
15252 },
15253 fancyOlItemWithNumber = {
15254     \item
15255     [
15256         \@@_latex_fancy_list_item_label:VVn
15257         \l_@@_latex_fancy_list_item_label_number_style_tl
15258         \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15259         { #1 }
15260     ]
15261 },

```

Make tight definition lists a little less compact by adding extra vertical space above and below them.

```

15262     dlBeginTight = {
15263         \group_begin:
15264         \plpartopsep=\partopsep
15265         \pltopsep=\topsep
15266         \begin { compactdesc }
15267     },
15268     dlEndTight = {
15269         \end { compactdesc }
15270         \group_end:
15271     }
15272 } }
15273 }
15274 {

```

Otherwise, if we loaded neither the enumitem package nor the paralist package, define the tight and fancy list renderer prototypes to fall back on the corresponding renderers for the non-tight lists.

```

15275     \markdownSetup
15276     {
15277         rendererPrototypes = {
15278             ulBeginTight = \markdownRendererUlBegin,
15279             ulEndTight = \markdownRendererUlEnd,
15280             fancyOlBegin = \markdownRendererOlBegin,
15281             fancyOlEnd = \markdownRendererOlEnd,
15282             olBeginTight = \markdownRendererOlBegin,
15283             olEndTight = \markdownRendererOlEnd,
15284             fancyOlBeginTight = \markdownRendererOlBegin,
15285             fancyOlEndTight = \markdownRendererOlEnd,

```



```

15286         dlBeginTight = \markdownRendererDlBegin,
15287         dlEndTight = \markdownRendererDlEnd,
15288     },
15289 }
15290 } }
15291 \ExplSyntaxOff
15292 \RequirePackage{amsmath}

```

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

```

15293 \@ifpackageloaded{unicode-math}{
15294     \markdownSetup{rendererPrototypes={
15295         untickedBox = {\mdlgwhtsquare},
15296     }}
15297 }{
15298     \RequirePackage{amssymb}
15299     \markdownSetup{rendererPrototypes={
15300         untickedBox = {\square},
15301     }}
15302 }
15303 \RequirePackage{csvsimple}
15304 \RequirePackage{fancyvrb}
15305 \RequirePackage{graphicx}
15306 \markdownSetup{rendererPrototypes={
15307     hardLineBreak = {\},
15308     leftBrace = {\textbraceleft},
15309     rightBrace = {\textbraceright},
15310     dollarSign = {\textdollar},
15311     underscore = {\textunderscore},
15312     circumflex = {\textasciicircum},
15313     backslash = {\textbackslash},
15314     tilde = {\textasciitilde},
15315     pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by  $\text{\TeX}$  during the typesetting. Therefore, even if we don't know whether a span of text is part of math formula or not when we are parsing markdown,<sup>36</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.

```

15316     codeSpan = {%
15317         \ifmmode
15318             \text{#1}%
15319         \else

```

---

<sup>36</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.

```

15320      \texttt{#1}%
15321      \fi
15322    }}}
```

### 3.3.4.2 Content Blocks

In content block renderer prototypes, display the content as a table using the package `csvsimple` when the raw attribute is `csv`, display the content using the default templates of the package `luaxml` when the raw attribute is `html`, execute the content with TeX when the raw attribute is `tex`, and display the content as markdown otherwise.

```

15323 \ExplSyntaxOn
15324 \markdownSetup{
15325   rendererPrototypes = {
15326     contentBlock = {
15327       \str_case:nnF
15328         { #1 }
15329         {
15330           { csv }
15331           {
15332             \begin { table }
15333             \begin { center }
15334               \csvautotabular { #3 }
15335             \end { center }
15336             \tl_if_empty:nF
15337               { #4 }
15338             { \caption { #4 } }
15339             \end { table }
15340           }
15341           { html }
15342           {
```

If we are using `TeX4ht`<sup>37</sup>, we will pass HTML elements to the output HTML document unchanged.

```

15343       \cs_if_exist:NTF
15344       \HCode
15345       {
15346         \if_mode_vertical:
15347           \IgnorePar
15348         \fi:
15349         \EndP
15350         \special
15351           { t4ht* < #3 }
15352         \par
15353         \ShowPar
```

---

<sup>37</sup>See <https://tug.org/tex4ht/>.

```

15354         }
15355         {
15356             \@@_luaxml_print_html:n
15357             { #3 }
15358         }
15359     }
15360     { tex }
15361     {
15362         \markdownEscape
15363         { #3 }
15364     }
15365 }
15366 {
15367     \markdownInput
15368     { #3 }
15369 }
15370 },
15371 },
15372 }
15373 \ExplSyntaxOff
15374 \markdownSetup{rendererPrototypes={
15375     ulBegin = {\begin{itemize}},
15376     ulEnd = {\end{itemize}},
15377     olBegin = {\begin{enumerate}},
15378     olItem = {\item{}},
15379     olItemWithNumber = {\item[#1.]},
15380     olEnd = {\end{enumerate}},
15381     dlBegin = {\begin{description}},
15382     dlItem = {\item[#1]},
15383     dlEnd = {\end{description}},
15384     emphasis = {\emph{#1}},
15385     tickedTextBox = {\$\boxtimes$},
15386     halfTickedTextBox = {\$\boxdot$}}}

```

If HTML identifiers appear after a heading, we make them produce `\label` macros.

```

15387 \ExplSyntaxOn
15388 \seq_new:N
15389 \g_@@_header_identifiers_seq
15390 \markdownSetup
15391 {
15392     rendererPrototypes = {
15393         headerAttributeContextBegin = {
15394             \markdownSetup
15395             {
15396                 rendererPrototypes = {
15397                     attributeIdentifier = {
15398                         \seq_gput_right:Nn
15399                         \g_@@_header_identifiers_seq

```

```

15400             { ##1 }
15401         },
15402     },
15403 }
15404 },
15405 headerAttributeContextEnd = {
15406     \seq_map_inline:Nn
15407     \g_@@_header_identifiers_seq
15408     { \label { ##1 } }
15409     \seq_gclear:N
15410     \g_@@_header_identifiers_seq
15411 },
15412 },
15413 }

```

If the `unnumbered` HTML class (or the `{-}` shorthand) appears after a heading the heading and all its subheadings will be unnumbered.

```

15414 \bool_new:N
15415 \l_@@_header_unnumbered_bool
15416 \markdownSetup
15417 {
15418     rendererPrototypes = {
15419         headerAttributeContextBegin += {
15420             \markdownSetup
15421             {
15422                 rendererPrototypes = {
15423                     attributeClassName = {
15424                         \bool_if:nT
15425                         {
15426                             \str_if_eq_p:nn
15427                             { ##1 }
15428                             { unnumbered } &&
15429                             ! \l_@@_header_unnumbered_bool
15430                         }
15431                     }
15432                     \group_begin:
15433                     \bool_set_true:N
15434                     \l_@@_header_unnumbered_bool
15435                     \c@secnumdepth = 0
15436                     \markdownSetup
15437                     {
15438                         rendererPrototypes = {
15439                             sectionBegin = {
15440                                 \group_begin:
15441                             },
15442                             sectionEnd = {
15443                                 \group_end:

```

```

15444         },
15445     },
15446 }
15447 }
15448 },
15449 },
15450 }
15451 },
15452 },
15453 }
15454 \ExplSyntaxOff
15455 \markdownSetup{rendererPrototypes={
15456   superscript = {\textsuperscript{#1}},
15457   subscript = {\textsubscript{#1}},
15458   blockQuoteBegin = {\begin{quotation}},
15459   blockQuoteEnd = {\end{quotation}},
15460   inputVerbatim = {\VerbatimInput{#1}},
15461   thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
15462   note = {\footnote{#1}}}}

```

### 3.3.4.3 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

15463 \RequirePackage{ltxcmds}
15464 \ExplSyntaxOn
15465 \cs_gset_protected:Npn
15466   \markdownRendererInputFencedCodePrototype#1#2#3
15467   {
15468     \tl_if_empty:nTF
15469       { #2 }
15470       { \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.

```

15471   {
15472     \regex_extract_once:nnN
15473       { \w* }
15474       { #2 }
15475       \l_tmpa_seq
15476     \seq_pop_left:NN
15477       \l_tmpa_seq
15478       \l_tmpa_tl

```

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

```

15479   \ltx@ifpackageloaded
15480     { minted }
15481     {
15482       \catcode`\%=14\relax

```

```

15483         \catcode`\#=6\relax
15484         \exp_args:NV
15485         \inputminted
15486         \l_tmpa_tl
15487         { #1 }
15488         \catcode`\%=12\relax
15489         \catcode`\#=12\relax
15490     }
15491 {

```

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

```

15492         \ltx@ifpackageloaded
15493         { listings }
15494         { \lstinputlisting [ language = \l_tmpa_tl ] { #1 } }

```

When neither the listings package nor the minted package is loaded, act as though no infostring were given.

```

15495         { \markdownRendererInputFencedCode { #1 } { } { } }
15496     }
15497 }
15498 }
15499 \ExplSyntaxOff

```

Support the nesting of strong emphasis.

```

15500 \ExplSyntaxOn
15501 \def\markdownLATEXStrongEmphasis#1{
15502     \str_if_in:NnTF
15503     \f@series
15504     { b }
15505     { \textnormal{#1} }
15506     { \textbf{#1} }
15507 }
15508 \ExplSyntaxOff
15509 \markdownSetup{rendererPrototypes={strongEmphasis={%
15510     \protect\markdownLATEXStrongEmphasis{#1}}}}

```

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

```

15511 \@ifundefined{chapter}{%
15512     \markdownSetup{rendererPrototypes = {
15513         headingOne = {\section{#1}},
15514         headingTwo = {\subsection{#1}},
15515         headingThree = {\subsubsection{#1}},
15516         headingFour = {\paragraph{#1}},
15517         headingFive = {\subparagraph{#1}}}}
15518 }{%
15519     \markdownSetup{rendererPrototypes = {
15520         headingOne = {\chapter{#1}},
15521         headingTwo = {\section{#1}},
15522         headingThree = {\subsection{#1}},

```

```

15523     headingFour = {\subsubsection{#1}},
15524     headingFive = {\paragraph{#1}},
15525     headingSix = {\subparagraph{#1}}}}
15526 }%

```

#### 3.3.4.4 Tickboxes

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

```

15527 \markdownSetup{
15528   rendererPrototypes = {
15529     ulItem = {%
15530       \futurelet\markdownLaTeXCheckbox\markdownLaTeXUItem
15531     },
15532   },
15533 }
15534 \def\markdownLaTeXUItem{%
15535   \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
15536     \item[\markdownLaTeXCheckbox]%
15537     \expandafter\@gobble
15538   \else
15539     \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
15540       \item[\markdownLaTeXCheckbox]%
15541       \expandafter\expandafter\expandafter\@gobble
15542     \else
15543       \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
15544         \item[\markdownLaTeXCheckbox]%
15545         \expandafter\expandafter\expandafter\expandafter
15546         \expandafter\expandafter\expandafter\@gobble
15547       \else
15548         \item{}%
15549       \fi
15550     \fi
15551   \fi
15552 }

```

#### 3.3.4.5 HTML elements

If the `html` option is enabled and we are using  $\text{\TeX}4\text{ht}$ <sup>38</sup>, we will pass HTML elements to the output HTML document unchanged.

```

15553 \@ifundefined{HCode}{}{
15554   \markdownSetup{
15555     rendererPrototypes = {
15556       inlineHtmlTag = {%
15557         \ifvmode
15558           \IgnorePar

```

---

<sup>38</sup>See <https://tug.org/tex4ht/>.

```

15559         \EndP
15560     \fi
15561     \HCode{#1}%
15562 },
15563 inputBlockHtmlElement = {%
15564     \ifvmode
15565         \IgnorePar
15566     \fi
15567     \EndP
15568     \special{t4ht*<#1}%
15569     \par
15570     \ShowPar
15571 },
15572 },
15573 }
15574 }

```

### 3.3.4.6 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.

```

15575 \newcount\markdownLaTeXCitationsCounter
15576
15577 % Basic implementation
15578 \long\def@gobblethree#1#2#3{%
15579 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
15580 \advance\markdownLaTeXCitationsCounter by 1\relax
15581 \ifx\relax#4\relax
15582 \ifx\relax#5\relax
15583 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15584 \relax
15585 \cite{#1#2#6}% No prenotes/postnotes, just accumulate cites
15586 \expandafter\expandafter\expandafter
15587 \expandafter\expandafter\expandafter\expandafter
15588 \@gobblethree
15589 \fi
15590 \else% Before a postnote (#5), dump the accumulator
15591 \ifx\relax#1\relax\else
15592 \cite{#1}%
15593 \fi
15594 \cite[#5]{#6}%
15595 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15596 \relax
15597 \else
15598 \expandafter\expandafter\expandafter

```



```

15599     \expandafter\expandafter\expandafter\expandafter
15600     \expandafter\expandafter\expandafter
15601     \expandafter\expandafter\expandafter\expandafter
15602     \markdownLaTeXBasicCitations
15603     \fi
15604     \expandafter\expandafter\expandafter
15605     \expandafter\expandafter\expandafter\expandafter{%
15606     \expandafter\expandafter\expandafter
15607     \expandafter\expandafter\expandafter\expandafter}%
15608     \expandafter\expandafter\expandafter
15609     \expandafter\expandafter\expandafter\expandafter{%
15610     \expandafter\expandafter\expandafter
15611     \expandafter\expandafter\expandafter\expandafter}%
15612     \expandafter\expandafter\expandafter
15613     \@gobblethree
15614     \fi
15615 \else% Before a prenote (#4), dump the accumulator
15616     \ifx\relax#1\relax\else
15617         \cite{#1}%
15618     \fi
15619     \ifnum\markdownLaTeXCitationsCounter>1\relax
15620         \space % Insert a space before the prenote in later citations
15621     \fi
15622     #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
15623     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15624     \relax
15625     \else
15626         \expandafter\expandafter\expandafter
15627         \expandafter\expandafter\expandafter\expandafter
15628         \markdownLaTeXBasicCitations
15629     \fi
15630     \expandafter\expandafter\expandafter{%
15631     \expandafter\expandafter\expandafter}%
15632     \expandafter\expandafter\expandafter{%
15633     \expandafter\expandafter\expandafter}%
15634     \expandafter
15635     \@gobblethree
15636     \fi\markdownLaTeXBasicCitations{#1#2#6},}
15637 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
15638
15639 % Natbib implementation
15640 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
15641     \advance\markdownLaTeXCitationsCounter by 1\relax
15642     \ifx\relax#3\relax
15643         \ifx\relax#4\relax
15644             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15645             \relax

```

```

15646      \citep{#1,#5}% No prenotes/postnotes, just accumulate cites
15647      \expandafter\expandafter\expandafter
15648      \expandafter\expandafter\expandafter\expandafter
15649      \@gobbletwo
15650    \fi
15651  \else% Before a postnote (#4), dump the accumulator
15652    \ifx\relax#1\relax\else
15653      \citep{#1}%
15654    \fi
15655    \citep[] [#4]{#5}%
15656    \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15657      \relax
15658    \else
15659      \expandafter\expandafter\expandafter
15660      \expandafter\expandafter\expandafter\expandafter
15661      \expandafter\expandafter\expandafter
15662      \expandafter\expandafter\expandafter\expandafter
15663      \markdownLaTeXNatbibCitations
15664    \fi
15665    \expandafter\expandafter\expandafter
15666    \expandafter\expandafter\expandafter\expandafter{%
15667    \expandafter\expandafter\expandafter
15668    \expandafter\expandafter\expandafter\expandafter}%
15669    \expandafter\expandafter\expandafter
15670    \@gobbletwo
15671  \fi
15672  \else% Before a prenote (#3), dump the accumulator
15673    \ifx\relax#1\relax\relax\else
15674      \citep{#1}%
15675    \fi
15676    \citep[#3] [#4]{#5}%
15677    \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15678      \relax
15679    \else
15680      \expandafter\expandafter\expandafter
15681      \expandafter\expandafter\expandafter\expandafter
15682      \markdownLaTeXNatbibCitations
15683    \fi
15684    \expandafter\expandafter\expandafter{%
15685    \expandafter\expandafter\expandafter}%
15686    \expandafter
15687    \@gobbletwo
15688    \fi\markdownLaTeXNatbibCitations{#1,#5}}
15689  \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
15690    \advance\markdownLaTeXCitationsCounter by 1\relax
15691    \ifx\relax#3\relax
15692      \ifx\relax#4\relax

```

```

15693 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15694 \relax
15695 \citet{#1,#5}% No prenotes/postnotes, just accumulate cites
15696 \expandafter\expandafter\expandafter
15697 \expandafter\expandafter\expandafter\expandafter
15698 \@gobbletwo
15699 \fi
15700 \else% After a prenote or a postnote, dump the accumulator
15701 \ifx\relax#1\relax\else
15702 \citet{#1}%
15703 \fi
15704 , \citet[#3][#4]{#5}%
15705 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
15706 \relax
15707 ,
15708 \else
15709 \ifnum
15710 \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
15711 \relax
15712 ,
15713 \fi
15714 \fi
15715 \expandafter\expandafter\expandafter
15716 \expandafter\expandafter\expandafter\expandafter
15717 \markdownLaTeXNatbibTextCitations
15718 \expandafter\expandafter\expandafter
15719 \expandafter\expandafter\expandafter\expandafter{%
15720 \expandafter\expandafter\expandafter
15721 \expandafter\expandafter\expandafter\expandafter}%
15722 \expandafter\expandafter\expandafter
15723 \@gobbletwo
15724 \fi
15725 \else% After a prenote or a postnote, dump the accumulator
15726 \ifx\relax#1\relax\relax\else
15727 \citet{#1}%
15728 \fi
15729 , \citet[#3][#4]{#5}%
15730 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
15731 \relax
15732 ,
15733 \else
15734 \ifnum
15735 \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
15736 \relax
15737 ,
15738 \fi
15739 \fi

```

```

15740 \expandafter\expandafter\expandafter
15741 \markdownLaTeXNatbibTextCitations
15742 \expandafter\expandafter\expandafter{%
15743 \expandafter\expandafter\expandafter}%
15744 \expandafter
15745 \@gobbletwo
15746 \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
15747
15748 % BibLaTeX implementation
15749 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
15750 \advance\markdownLaTeXCitationsCounter by 1\relax
15751 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15752 \relax
15753 \autocites#1[#3][#4]{#5}%
15754 \expandafter\@gobbletwo
15755 \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
15756 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
15757 \advance\markdownLaTeXCitationsCounter by 1\relax
15758 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15759 \relax
15760 \textcites#1[#3][#4]{#5}%
15761 \expandafter\@gobbletwo
15762 \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}}
15763
15764 \markdownSetup{rendererPrototypes = {
15765 cite = {%
15766 \markdownLaTeXCitationsCounter=1%
15767 \def\markdownLaTeXCitationsTotal{#1}%
15768 \@ifundefined{autocites}{%
15769 \@ifundefined{citep}{%
15770 \expandafter\expandafter\expandafter
15771 \markdownLaTeXBasicCitations
15772 \expandafter\expandafter\expandafter{%
15773 \expandafter\expandafter\expandafter}%
15774 \expandafter\expandafter\expandafter{%
15775 \expandafter\expandafter\expandafter}%
15776 }{%
15777 \expandafter\expandafter\expandafter
15778 \markdownLaTeXNatbibCitations
15779 \expandafter\expandafter\expandafter{%
15780 \expandafter\expandafter\expandafter}%
15781 }%
15782 }{%
15783 \expandafter\expandafter\expandafter
15784 \markdownLaTeXBibLaTeXCitations
15785 \expandafter{\expandafter}%
15786 }},

```

```

15787 textCite = {%
15788   \markdownLaTeXCitationsCounter=1%
15789   \def\markdownLaTeXCitationsTotal{#1}%
15790   \@ifundefined{autocites}{%
15791     \@ifundefined{citep}{%
15792       \expandafter\expandafter\expandafter
15793       \markdownLaTeXBasicTextCitations
15794       \expandafter\expandafter\expandafter{%
15795         \expandafter\expandafter\expandafter}%
15796       \expandafter\expandafter\expandafter{%
15797         \expandafter\expandafter\expandafter}%
15798     }{%
15799       \expandafter\expandafter\expandafter
15800       \markdownLaTeXNatbibTextCitations
15801       \expandafter\expandafter\expandafter{%
15802         \expandafter\expandafter\expandafter}%
15803     }%
15804   }{%
15805     \expandafter\expandafter\expandafter
15806     \markdownLaTeXBibLaTeXTextCitations
15807     \expandafter{\expandafter}%
15808   }}}

```

### 3.3.4.7 Links

Here is an implementation for hypertext links and relative references.

```

15809 \RequirePackage{url}
15810 \RequirePackage{expl3}
15811 \ExplSyntaxOn
15812 \cs_gset_protected:Npn
15813   \markdownRendererLinkPrototype
15814   #1#2#3#4
15815   {
15816     \tl_set:Nn \l_tmpa_tl { #1 }
15817     \tl_set:Nn \l_tmpb_tl { #2 }
15818     \bool_set:Nn
15819       \l_tmpa_bool
15820       {
15821         \tl_if_eq_p:NN
15822           \l_tmpa_tl
15823           \l_tmpb_tl
15824       }
15825     \tl_set:Nn \l_tmpa_tl { #4 }
15826     \bool_set:Nn
15827       \l_tmpb_bool
15828       {
15829         \tl_if_empty_p:N

```

```

15830         \l_tmpa_tl
15831     }

```

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.

```

15832     \bool_if:nTF
15833     {
15834         \l_tmpa_bool && \l_tmpb_bool
15835     }
15836     {
15837         \markdownLaTeXRendererAutolink { #2 } { #3 }
15838     }
15839     {
15840         \markdownLaTeXRendererDirectOrIndirectLink
15841         { #1 } { #2 } { #3 } { #4 }
15842     }
15843 }
15844 \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.

```

15845     \tl_set:Nn
15846         \l_tmpa_tl
15847         { #2 }
15848     \tl_trim_spaces:N
15849         \l_tmpa_tl
15850     \tl_set:Nx
15851         \l_tmpb_tl
15852         {
15853             \tl_range:Nnn
15854                 \l_tmpa_tl
15855                 { 1 }
15856                 { 1 }
15857         }
15858     \str_if_eq:NNTF
15859         \l_tmpb_tl
15860         \c_hash_str
15861         {
15862             \tl_set:Nx
15863                 \l_tmpb_tl
15864                 {
15865                     \tl_range:Nnn
15866                         \l_tmpa_tl
15867                         { 2 }
15868                         { -1 }
15869                 }
15870             \exp_args:NV

```

```

15871         \ref
15872         \l_tmpb_tl
15873     }
15874     {
15875         \url { #2 }
15876     }
15877 }
15878 \ExplSyntaxOff
15879 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
15880     #1\footnote{\ifx\empty#4\empty\else#4: \fi\url{#3}}}%

```

### 3.3.4.8 Tables

Here is a basic implementation of tables. If the booktabs package is loaded, then it is used to produce horizontal lines.

```

15881 \newcount\markdownLaTeXRowCount
15882 \newcount\markdownLaTeXRowTotal
15883 \newcount\markdownLaTeXColumnCounter
15884 \newcount\markdownLaTeXColumnTotal
15885 \newtoks\markdownLaTeXTable
15886 \newtoks\markdownLaTeXTableAlignment
15887 \newtoks\markdownLaTeXTableEnd
15888 \AtBeginDocument{%
15889     \@ifpackageloaded{booktabs}{%
15890         \def\markdownLaTeXTopRule{\toprule}%
15891         \def\markdownLaTeXMidRule{\midrule}%
15892         \def\markdownLaTeXBottomRule{\bottomrule}%
15893     }{%
15894         \def\markdownLaTeXTopRule{\hline}%
15895         \def\markdownLaTeXMidRule{\hline}%
15896         \def\markdownLaTeXBottomRule{\hline}%
15897     }%
15898 }
15899 \markdownSetup{rendererPrototypes={
15900     table = {%
15901         \markdownLaTeXTable={}%
15902         \markdownLaTeXTableAlignment={}%
15903         \markdownLaTeXTableEnd={%
15904             \markdownLaTeXBottomRule
15905             \end{tabular}}}%
15906     \ifx\empty#1\empty\else
15907         \addto@hook\markdownLaTeXTable{%
15908             \begin{table}
15909             \centering}%
15910         \addto@hook\markdownLaTeXTableEnd{%
15911             \caption{#1}}}%
15912     \fi

```

```

15913   }
15914 }}

```

If the `tableAttributes` option is enabled, we will register any identifiers, so that they can be used as L<sup>A</sup>T<sub>E</sub>X labels for referencing tables.

```

15915 \ExplSyntaxOn
15916 \seq_new:N
15917   \l_@@_table_identifiers_seq
15918 \markdownSetup {
15919   rendererPrototypes = {
15920     table += {
15921       \seq_map_inline:Nn
15922         \l_@@_table_identifiers_seq
15923       {
15924         \addto@hook
15925           \markdownLaTeXTableEnd
15926           { \label { ##1 } }
15927       }
15928     },
15929   }
15930 }
15931 \markdownSetup {
15932   rendererPrototypes = {
15933     tableAttributeContextBegin = {
15934       \group_begin:
15935       \markdownSetup {
15936         rendererPrototypes = {
15937           attributeIdentifier = {
15938             \seq_put_right:Nn
15939               \l_@@_table_identifiers_seq
15940               { ##1 }
15941           },
15942         },
15943       }
15944     },
15945     tableAttributeContextEnd = {
15946       \group_end:
15947     },
15948   },
15949 }
15950 \ExplSyntaxOff
15951 \markdownSetup{rendererPrototypes={
15952   table += {%
15953     \ifx\empty#1\empty\else
15954       \addto@hook\markdownLaTeXTableEnd{%
15955         \end{table}}}%
15956   \fi

```



```

15957 \addto@hook\markdownLaTeXTable{\begin{tabular}}%
15958 \markdownLaTeXRowCount=0%
15959 \markdownLaTeXRowTotal=#2%
15960 \markdownLaTeXColumnTotal=#3%
15961 \markdownLaTeXRenderTableRow
15962 }
15963 }}
15964 \def\markdownLaTeXRenderTableRow#1{%
15965 \markdownLaTeXColumnCounter=0%
15966 \ifnum\markdownLaTeXRowCount=0\relax
15967 \markdownLaTeXReadAlignments#1%
15968 \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
15969 \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
15970 \the\markdownLaTeXTableAlignment}}%
15971 \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
15972 \else
15973 \markdownLaTeXRenderTableCell#1%
15974 \fi
15975 \ifnum\markdownLaTeXRowCount=1\relax
15976 \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
15977 \fi
15978 \advance\markdownLaTeXRowCount by 1\relax
15979 \ifnum\markdownLaTeXRowCount>\markdownLaTeXRowTotal\relax
15980 \the\markdownLaTeXTable
15981 \the\markdownLaTeXTableEnd
15982 \expandafter\@gobble
15983 \fi\markdownLaTeXRenderTableRow}
15984 \def\markdownLaTeXReadAlignments#1{%
15985 \advance\markdownLaTeXColumnCounter by 1\relax
15986 \if#1d%
15987 \addto@hook\markdownLaTeXTableAlignment{1}%
15988 \else
15989 \addto@hook\markdownLaTeXTableAlignment{#1}%
15990 \fi
15991 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
15992 \expandafter\@gobble
15993 \fi\markdownLaTeXReadAlignments}
15994 \def\markdownLaTeXRenderTableCell#1{%
15995 \advance\markdownLaTeXColumnCounter by 1\relax
15996 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
15997 \addto@hook\markdownLaTeXTable{#1&}%
15998 \else
15999 \addto@hook\markdownLaTeXTable{#1\\}%
16000 \expandafter\@gobble
16001 \fi\markdownLaTeXRenderTableCell}

```

### 3.3.4.9 Line Blocks

Here is a basic implementation of line blocks. If the `verse` package is loaded, then it is used to produce the verses.

```

16002
16003 \markdownIfOption{lineBlocks}{%
16004   \RequirePackage{verse}
16005   \markdownSetup{rendererPrototypes={
16006     lineBlockBegin = {%
16007       \begingroup
16008       \def\markdownRendererHardLineBreak{\\}%
16009       \begin{verse}%
16010     },
16011     lineBlockEnd = {%
16012       \end{verse}%
16013     \endgroup
16014   },
16015   }}
16016 }{}
16017

```

#### 3.3.4.10 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.

```

16018 \ExplSyntaxOn
16019 \keys_define:nn
16020 { markdown / jekyllData }
16021 {
16022   author .code:n = {
16023     \author
16024     { #1 }
16025   },
16026   date .code:n = {
16027     \date
16028     { #1 }
16029   },
16030   title .code:n = {
16031     \title
16032     { #1 }

```

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.

```

16033   \AddToHook
16034   { begindocument / end }
16035   { \maketitle }

```

```

16036     },
16037   }

```

#### 3.3.4.11 Marked Text

If the `mark` option is enabled, we will load either the `soul` package or the `lua-ul` package and use it to implement marked text.

```

16038 \@@_if_option:nT
16039   { mark }
16040   {
16041     \sys_if_engine luatex:TF
16042     {
16043       \RequirePackage
16044         { luacolor }
16045       \RequirePackage
16046         { lua-ul }
16047       \markdownSetup
16048         {
16049           rendererPrototypes = {
16050             mark = {
16051               \highLight
16052                 { #1 }
16053             },
16054           }
16055         }
16056     }
16057     {
16058       \RequirePackage
16059         { xcolor }
16060       \RequirePackage
16061         { soul }
16062       \markdownSetup
16063         {
16064           rendererPrototypes = {
16065             mark = {
16066               \hl
16067                 { #1 }
16068             },
16069           }
16070         }
16071     }
16072   }

```

#### 3.3.4.12 Strike-Through

If the `strikeThrough` option is enabled, we will load either the `soul` package or the `lua-ul` package and use it to implement strike-throughs.

```

16073 \@@_if_option:nT
16074 { strikeThrough }
16075 {
16076   \sys_if_engine luatex:TF
16077   {
16078     \RequirePackage
16079     { lua-ul }
16080     \markdownSetup
16081     {
16082       rendererPrototypes = {
16083         strikeThrough = {
16084           \strikeThrough
16085           { #1 }
16086         },
16087       }
16088     }
16089   }
16090   {
16091     \RequirePackage
16092     { soul }
16093     \markdownSetup
16094     {
16095       rendererPrototypes = {
16096         strikeThrough = {
16097           \st
16098           { #1 }
16099         },
16100       }
16101     }
16102   }
16103 }

```

### 3.3.4.13 Images and their attributes

We define images to be rendered as floating figures using the command `\includegraphics`, where the image label is the alt text and the image title is the caption of the figure.

If the `linkAttributes` option is enabled, 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 and we will register any identifiers, so that they can be used as  $\text{\LaTeX}$  labels for referencing figures.

```

16104 \seq_new:N
16105 \l_@@_image_identifiers_seq
16106 \markdownSetup {
16107   rendererPrototypes = {
16108     image = {
16109       \tl_if_empty:nTF

```

```

16110     { #4 }
16111     {
16112         \begin { center }
16113         \includegraphics
16114             [ alt = { #1 } ]
16115             { #3 }
16116         \end { center }
16117     }
16118     {
16119         \begin { figure }
16120         \begin { center }
16121             \includegraphics
16122                 [ alt = { #1 } ]
16123                 { #3 }
16124             \caption { #4 }
16125             \seq_map_inline:Nn
16126                 \l_@@_image_identifiers_seq
16127                 { \label { ##1 } }
16128             \end { center }
16129         \end { figure }
16130     }
16131 },
16132 }
16133 }
16134 \@@_if_option:nT
16135 { linkAttributes }
16136 {
16137     \RequirePackage { graphicx }
16138 }
16139 \markdownSetup {
16140     rendererPrototypes = {
16141         imageAttributeContextBegin = {
16142             \group_begin:
16143             \markdownSetup {
16144                 rendererPrototypes = {
16145                     attributeIdentifier = {
16146                         \seq_put_right:Nn
16147                             \l_@@_image_identifiers_seq
16148                             { ##1 }
16149                     },
16150                     attributeKeyValue = {
16151                         \setkeys
16152                             { Gin }
16153                             { { ##1 } = { ##2 } }
16154                     },
16155                 },
16156             }

```

```

16157     },
16158     imageAttributeContextEnd = {
16159         \group_end:
16160     },
16161 },
16162 }
16163 \ExplSyntaxOff

```

### 3.3.4.14 Raw Attributes

In the raw block and inline raw span renderer prototypes, display the content using the default templates of the package `luaxml` when the raw attribute is `html` and default to the plain TeX renderer prototypes otherwise, translating raw attribute `latex` to `tex`.

```

16164 \ExplSyntaxOn
16165 \cs_new:Nn
16166   \@@_luaxml_print_html:n
16167   {
16168     \luabridge_now:n
16169     {
16170       local~input_file = assert(io.open(" #1 ", "r"))
16171       local~input = assert(input_file:read("*a"))
16172       assert(input_file:close())
16173       input = "<body>" .. input .. "</body>"
16174       local~dom = require("luaxml-domobject").html_parse(input)
16175       local~output = require("luaxml-htmltemplates"):process_dom(dom)
16176       print(output)
16177     }
16178   }
16179 \cs_gset_protected:Npn
16180   \markdownRendererInputRawInlinePrototype#1#2
16181   {
16182     \str_case:nnF
16183       { #2 }
16184       {
16185         { latex }
16186         {
16187           \@@_plain_tex_default_input_raw_inline:nn
16188           { #1 }
16189           { tex }
16190         }
16191         { html }
16192         {

```

If we are using `TeX4ht`<sup>39</sup>, we will pass HTML elements to the output HTML document unchanged.

---

<sup>39</sup>See <https://tug.org/tex4ht/>.

```

16193         \cs_if_exist:NTF
16194         \HCode
16195         {
16196             \if_mode_vertical:
16197             \IgnorePar
16198             \EndP
16199             \fi:
16200             \special
16201             { t4ht* < #1 }
16202         }
16203         {
16204             \@@_luaxml_print_html:n
16205             { #1 }
16206         }
16207     }
16208 }
16209 {
16210     \@@_plain_tex_default_input_raw_inline:nn
16211     { #1 }
16212     { #2 }
16213 }
16214 }
16215 \cs_gset_protected:Npn
16216 \markdownRendererInputRawBlockPrototype#1#2
16217 {
16218     \str_case:nnF
16219     { #2 }
16220     {
16221         { latex }
16222         {
16223             \@@_plain_tex_default_input_raw_block:nn
16224             { #1 }
16225             { tex }
16226         }
16227         { html }
16228         {

```

If we are using  $\text{\TeX}4\text{ht}$ <sup>40</sup>, we will pass HTML elements to the output HTML document unchanged.

```

16229         \cs_if_exist:NTF
16230         \HCode
16231         {
16232             \if_mode_vertical:
16233             \IgnorePar
16234             \fi:
16235             \EndP

```

---

<sup>40</sup>See <https://tug.org/tex4ht/>.

```

16236         \special
16237         { t4ht* < #1 }
16238     \par
16239     \ShowPar
16240 }
16241 {
16242     \@@_luaxml_print_html:n
16243     { #1 }
16244 }
16245 }
16246 }
16247 {
16248     \@@_plain_tex_default_input_raw_block:nn
16249     { #1 }
16250     { #2 }
16251 }
16252 }

```

### 3.3.4.15 Bracketed spans

If the `bracketedSpans` option is enabled, we will register any identifiers, so that they can be used as L<sup>A</sup>T<sub>E</sub>X labels for referencing the last L<sup>A</sup>T<sub>E</sub>X counter that has been incremented in e.g. ordered lists.

```

16253 \seq_new:N
16254 \l_@@_bracketed_span_identifiers_seq
16255 \markdownSetup {
16256     rendererPrototypes = {
16257         bracketedSpanAttributeContextBegin = {
16258             \group_begin:
16259             \markdownSetup {
16260                 rendererPrototypes = {
16261                     attributeIdentifier = {
16262                         \seq_put_right:Nn
16263                         \l_@@_bracketed_span_identifiers_seq
16264                         { ##1 }
16265                     },
16266                 },
16267             }
16268         },
16269         bracketedSpanAttributeContextEnd = {
16270             \seq_map_inline:Nn
16271             \l_@@_bracketed_span_identifiers_seq
16272             { \label { ##1 } }
16273             \group_end:
16274         },
16275     },
16276 }

```



```

16277 \ExplSyntaxOff
16278 \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.

```

16279 \newcommand\markdownMakeOther{%
16280   \count0=128\relax
16281   \loop
16282     \catcode\count0=11\relax
16283     \advance\count0 by 1\relax
16284   \ifnum\count0<256\repeat}%

```

## 3.4 ConT<sub>E</sub>Xt Implementation

The ConT<sub>E</sub>Xt implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV ConT<sub>E</sub>Xt formats *seem* to implement (the documentation is scarce) the majority of the plain T<sub>E</sub>X format required by the plain T<sub>E</sub>X implementation. As a consequence, we can directly reuse the existing plain T<sub>E</sub>X implementation after supplying the missing plain T<sub>E</sub>X macros.

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `\enableregime` macro. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the filecontents L<sup>A</sup>T<sub>E</sub>X package.

```

16285 \def\markdownMakeOther{%
16286   \count0=128\relax
16287   \loop
16288     \catcode\count0=11\relax
16289     \advance\count0 by 1\relax
16290   \ifnum\count0<256\repeat

```

On top of that, make the pipe character (`|`) inactive during the scanning. This is necessary, since the character is active in ConT<sub>E</sub>Xt.

```

16291   \catcode`|=12}%

```

### 3.4.1 Typesetting Markdown

The `\inputmarkdown` and `\inputyaml` macros are defined to accept an optional argument with options recognized by the ConT<sub>E</sub>Xt interface (see Section 2.4.2).

```

16292 \long\def\inputmarkdown{%
16293   \dosingleempty
16294   \doinputmarkdown}%

```

```

16295 \long\def\doinputmarkdown[#1]#2{%
16296   \begingroup
16297     \iffirstargument
16298       \setupmarkdown[#1]%
16299     \fi
16300     \markdownInput{#2}%
16301   \endgroup}%
16302 \long\def\inputyaml{%
16303   \dosingleempty
16304   \doinputyaml}%
16305 \long\def\doinputyaml[#1]#2{%
16306   \doinputmarkdown
16307     [jekyllData, expectJekyllData, ensureJekyllData, #1]{#2}}%

```

The `\startmarkdown`, `\stopmarkdown`, `\startyaml`, and `\stopyaml` macros are implemented using the `\markdownReadAndConvert` macro.

In Knuth’s  $\text{\TeX}$ , trailing spaces are removed very early on when a line is being put to the input buffer. [18, sec. 31]. According to Eijkhout [19, sec. 2.2], this is because “these spaces are hard to see in an editor”. At the moment, there is no option to suppress this behavior in (Lua) $\text{\TeX}$ , but Con $\text{\TeX}$ t MkIV funnels all input through its own input handler. This makes it possible to suppress the removal of trailing spaces in Con $\text{\TeX}$ t MkIV and therefore to insert hard line breaks into markdown text.

```

16308 \startluacode
16309   document.markdown_buffering = false
16310   local function preserve_trailing_spaces(line)
16311     if document.markdown_buffering then
16312       line = line:gsub("[ \t][ \t]$", "\t\t")
16313     end
16314     return line
16315   end
16316   resolvers.installinputlinehandler(preserve_trailing_spaces)
16317 \stopluacode
16318 \begingroup
16319   \catcode`\|=0%
16320   \catcode`\|=12%
16321   |gdef|startmarkdown{%
16322     |ctxlua{document.markdown_buffering = true}%
16323     |markdownReadAndConvert{\stopmarkdown}%
16324                               {|\stopmarkdown}}%
16325   |gdef|stopmarkdown{%
16326     |ctxlua{document.markdown_buffering = false}%
16327     |markdownEnd}%
16328   |gdef|startyaml{%
16329     |begingroup
16330     |ctxlua{document.markdown_buffering = true}%
16331     |setupyaml[jekyllData, expectJekyllData, ensureJekyllData]%
16332     |markdownReadAndConvert{\stopyaml}%

```

```

16333                                     {\stopyaml}}}%
16334   |gdef|stopyaml{%
16335     |ctxlua{document.markdown_buffering = false}%
16336     |yamlEnd}%
16337 |endgroup

```

### 3.4.2 Themes

This section overrides the plain T<sub>E</sub>X implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in ConT<sub>E</sub>Xt themes provided with the Markdown package.

```

16338 \ExplSyntaxOn
16339 \prop_new:N \g_@@_context_loaded_themes_linenos_prop
16340 \prop_new:N \g_@@_context_loaded_themes_versions_prop
16341 \cs_gset:Nn % noqa: w401
16342   \@@_load_theme:nnn
16343   {

```

Determine whether either this is a built-in theme according to the prop `\g_@@_context_built_in_themes_prop` or a file named `t-markdowntheme<munged theme name>.tex` exists. If it does, load it. Otherwise, try loading a plain T<sub>E</sub>X theme instead.

```

16344   \bool_if:nTF
16345     {
16346       \bool_lazy_or_p:nn
16347         {
16348           \prop_if_in_p:Nn
16349             \g_@@_context_built_in_themes_prop
16350             { #1 }
16351         }
16352         {
16353           \file_if_exist_p:n
16354             { t - markdown theme #3.tex }
16355         }
16356     }
16357   {
16358     \prop_get:NnNTF
16359       \g_@@_context_loaded_themes_linenos_prop
16360       { #1 }
16361     \l_tmpa_tl
16362     {
16363       \prop_get:NnN
16364         \g_@@_context_loaded_themes_versions_prop
16365         { #1 }
16366       \l_tmpb_tl
16367       \str_if_eq:nVTF

```

```

16368         { #2 }
16369         \l_tmpb_tl
16370         {
16371             \msg_warning:nnnVn
16372             { markdown }
16373             { repeatedly-loaded-context-theme }
16374             { #1 }
16375             \l_tmpa_tl
16376             { #2 }
16377         }
16378         {
16379             \msg_error:nnnnVV
16380             { markdown }
16381             { different-versions-of-context-theme }
16382             { #1 }
16383             { #2 }
16384             \l_tmpb_tl
16385             \l_tmpa_tl
16386         }
16387     }
16388     {
16389         \prop_gput:Nnx
16390         \g_@@_context_loaded_themes_linenos_prop
16391         { #1 }
16392         { \tex_the:D \tex_inputlineno:D } % noqa: W200
16393         \prop_gput:Nnn
16394         \g_@@_context_loaded_themes_versions_prop
16395         { #1 }
16396         { #2 }

```

Load built-in plain TeX themes from the prop `\g_@@_context_built_in_themes_prop` and from the filesystem otherwise.

```

16397         \prop_if_in:NnTF
16398         \g_@@_context_built_in_themes_prop
16399         { #1 }
16400         {
16401             \msg_info:nnnn
16402             { markdown }
16403             { loading-built-in-context-theme }
16404             { #1 }
16405             { #2 }
16406             \prop_item:Nn
16407             \g_@@_context_built_in_themes_prop
16408             { #1 }
16409         }
16410         {
16411             \msg_info:nnnn

```

```

16412         { markdown }
16413         { loading-context-theme }
16414         { #1 }
16415         { #2 }
16416         \usemodule
16417         [ t ]
16418         [ markdown theme #3 ]
16419     }
16420 }
16421 }
16422 {
16423     \@@_plain_tex_load_theme:nnn
16424     { #1 }
16425     { #2 }
16426     { #3 }
16427 }
16428 }
16429 \msg_new:nnn
16430 { markdown }
16431 { loading-built-in-context-theme }
16432 { Loading~version~#2~of~built-in~ConTeXt~Markdown~theme~#1 }
16433 \msg_new:nnn
16434 { markdown }
16435 { loading-context-theme }
16436 { Loading~version~#2~of~ConTeXt~Markdown~theme~#1 }
16437 \msg_new:nnn
16438 { markdown }
16439 { repeatedly-loaded-context-theme }
16440 {
16441     Version~#3~of~ConTeXt~Markdown~theme~#1~was~previously~
16442     loaded~on~line~#2,~not~loading~it~again
16443 }
16444 \msg_new:nnn
16445 { markdown }
16446 { different-versions-of-context-theme }
16447 {
16448     Tried~to~load~version~#2~of~ConTeXt~Markdown~theme~#1~
16449     but~version~#3~has~already~been~loaded~on~line~#4
16450 }
16451 \ExplSyntaxOff

```

The [witiko/markdown/defaults](#) ConTeXt theme provides default definitions for token renderer prototypes. First, the ConTeXt theme loads the plain TeX theme with the default definitions for plain TeX:

```
16452 \markdownLoadPlainTeXTheme
```

Next, the ConTeXt theme overrides some of the plain TeX 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.

```
16453 \markdownIfOption{plain}{\iffalse}{\iftrue}
16454 \def\markdownRendererHardLineBreakPrototype{\blank}%
16455 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
16456 \def\markdownRendererRightBracePrototype{\textbraceright}%
16457 \def\markdownRendererDollarSignPrototype{\textdollar}%
16458 \def\markdownRendererPercentSignPrototype{\percent}%
16459 \def\markdownRendererUnderscorePrototype{\textunderscore}%
16460 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
16461 \def\markdownRendererBackslashPrototype{\textbackslash}%
16462 \def\markdownRendererTildePrototype{\textasciitilde}%
16463 \def\markdownRendererPipePrototype{\char`|}%
16464 \def\markdownRendererLinkPrototype#1#2#3#4{%
16465   \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
16466     \fi\texttt<\hyphenatedurl{#3}>}}%
16467 \usemodule[database]
16468 \defineseparatedlist
16469   [MarkdownConTeXtCSV]
16470   [separator={,},
16471     before=\bTABLE,after=\eTABLE,
16472     first=\bTR,last=\eTR,
16473     left=\bTD,right=\eTD]
16474 \def\markdownConTeXtCSV{csv}
16475 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
16476   \def\markdownConTeXtCSV@arg{#1}%
16477   \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
16478     \placetable[] [tab:#1]{#4}{%
16479       \processseparatedfile[MarkdownConTeXtCSV][#3]}%
16480   \else
16481     \markdownInput{#3}%
16482   \fi}%
16483 \def\markdownRendererImagePrototype#1#2#3#4{%
16484   \placefigure[] []{#4}{\externalfigure[#3]}}%
16485 \def\markdownRendererUlBeginPrototype{\startitemize}%
16486 \def\markdownRendererUlBeginTightPrototype{\startitemize[packed]}%
16487 \def\markdownRendererUlItemPrototype{\item}%
16488 \def\markdownRendererUlEndPrototype{\stopitemize}%
16489 \def\markdownRendererUlEndTightPrototype{\stopitemize}%
16490 \def\markdownRendererOlBeginPrototype{\startitemize[n]}%
16491 \def\markdownRendererOlBeginTightPrototype{\startitemize[packed,n]}%
16492 \def\markdownRendererOlItemPrototype{\item}%
16493 \def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
16494 \def\markdownRendererOlEndPrototype{\stopitemize}%
16495 \def\markdownRendererOlEndTightPrototype{\stopitemize}%
```

```

16496 \definedescription
16497   [MarkdownConTeXtDlItemPrototype]
16498   [location=hanging,
16499    margin=standard,
16500    headstyle=bold]%
16501 \definestartstop
16502   [MarkdownConTeXtDlPrototype]
16503   [before=\blank,
16504    after=\blank]%
16505 \definestartstop
16506   [MarkdownConTeXtDlTightPrototype]
16507   [before=\blank\startpacked,
16508    after=\stoppacked\blank]%
16509 \def\markdownRendererDlBeginPrototype{%
16510   \startMarkdownConTeXtDlPrototype}%
16511 \def\markdownRendererDlBeginTightPrototype{%
16512   \startMarkdownConTeXtDlTightPrototype}%
16513 \def\markdownRendererDlItemPrototype#1{%
16514   \startMarkdownConTeXtDlItemPrototype{#1}}%
16515 \def\markdownRendererDlItemEndPrototype{%
16516   \stopMarkdownConTeXtDlItemPrototype}%
16517 \def\markdownRendererDlEndPrototype{%
16518   \stopMarkdownConTeXtDlPrototype}%
16519 \def\markdownRendererDlEndTightPrototype{%
16520   \stopMarkdownConTeXtDlTightPrototype}%
16521 \def\markdownRendererEmphasisPrototype#1{\em#1}%
16522 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
16523 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
16524 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
16525 \def\markdownRendererLineBlockBeginPrototype{%
16526   \begingroup
16527     \def\markdownRendererHardLineBreak{
16528       }%
16529     \startlines
16530   }%
16531 \def\markdownRendererLineBlockEndPrototype{%
16532   \stoplines
16533   \endgroup
16534 }%
16535 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%

```

#### 3.4.3.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

16536 \ExplSyntaxOn
16537 \cs_gset:Npn
16538   \markdownRendererInputFencedCodePrototype#1#2#3

```

```

16539 {
16540   \tl_if_empty:nTF
16541     { #2 }
16542     { \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]
\setuptyping [latex] [option=TEX]

\starttext
  \startmarkdown
~~~ latex
\documentclass{article}
\begin{document}
  Hello world!
\end{document}
~~~
  \stopmarkdown
\stoptext

```

```

16543 {
16544   \regex_extract_once:nnN
16545     { \w* }
16546     { #2 }
16547     \l_tmpa_seq
16548     \seq_pop_left:NN
16549     \l_tmpa_seq
16550     \l_tmpa_tl
16551     \typefile[ \l_tmpa_tl ][] {#1}
16552   }
16553 }
16554 \ExplSyntaxOff
16555 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
16556 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
16557 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
16558 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
16559 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
16560 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
16561 \def\markdownRendererThematicBreakPrototype{%
16562   \blackrule[height=1pt, width=\hsize]}%
16563 \def\markdownRendererNotePrototype#1{\footnote{#1}}%
16564 \def\markdownRendererTickedBoxPrototype{$\boxtimes$}

```



```

16565 \def\markdownRendererHalfTickedBoxPrototype{\boxdot$}
16566 \def\markdownRendererUntickedBoxPrototype{\square$}
16567 \def\markdownRendererStrikeThroughPrototype#1{\overstrikes{#1}}
16568 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
16569 \def\markdownRendererSubscriptPrototype#1{\low{#1}}
16570 \def\markdownRendererDisplayMathPrototype#1{%
16571   \startformula#1\stopformula}%

```

### 3.4.3.2 Tables

There is a basic implementation of tables.

```

16572 \newcount\markdownConTeXtRowCounter
16573 \newcount\markdownConTeXtRowTotal
16574 \newcount\markdownConTeXtColumnCounter
16575 \newcount\markdownConTeXtColumnTotal
16576 \newtoks\markdownConTeXtTable
16577 \newtoks\markdownConTeXtTableFloat
16578 \def\markdownRendererTablePrototype#1#2#3{%
16579   \markdownConTeXtTable={}%
16580   \ifx\empty#1\empty
16581     \markdownConTeXtTableFloat={%
16582       \the\markdownConTeXtTable}%
16583   \else
16584     \markdownConTeXtTableFloat={%
16585       \placetable{#1}{\the\markdownConTeXtTable}}%
16586   \fi
16587   \begingroup
16588   \setupTABLE[r][each][topframe=off, bottomframe=off,
16589     leftframe=off, rightframe=off]
16590   \setupTABLE[c][each][topframe=off, bottomframe=off,
16591     leftframe=off, rightframe=off]
16592   \setupTABLE[r][1][topframe=on, bottomframe=on]
16593   \setupTABLE[r][#1][bottomframe=on]
16594   \markdownConTeXtRowCounter=0%
16595   \markdownConTeXtRowTotal=#2%
16596   \markdownConTeXtColumnTotal=#3%
16597   \markdownConTeXtRenderTableRow}
16598 \def\markdownConTeXtRenderTableRow#1{%
16599   \markdownConTeXtColumnCounter=0%
16600   \ifnum\markdownConTeXtRowCounter=0\relax
16601     \markdownConTeXtReadAlignments#1%
16602     \markdownConTeXtTable={\bTABLE}%
16603   \else
16604     \markdownConTeXtTable=\expandafter{%
16605       \the\markdownConTeXtTable\bTR}%
16606     \markdownConTeXtRenderTableCell#1%
16607     \markdownConTeXtTable=\expandafter{%

```

```

16608     \the\markdownConTeXtTable\eTR}%
16609 \fi
16610 \advance\markdownConTeXtRowCounter by 1\relax
16611 \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
16612   \markdownConTeXtTable=\expandafter{%
16613     \the\markdownConTeXtTable\eTABLE}%
16614   \the\markdownConTeXtTableFloat
16615   \endgroup
16616   \expandafter\gobbleoneargument
16617 \fi\markdownConTeXtRenderTableRow}
16618 \def\markdownConTeXtReadAlignments#1{%
16619   \advance\markdownConTeXtColumnCounter by 1\relax
16620   \if#1d%
16621     \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
16622   \fi\if#1l%
16623     \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
16624   \fi\if#1c%
16625     \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
16626   \fi\if#1r%
16627     \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
16628   \fi
16629   \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
16630   \else
16631     \expandafter\gobbleoneargument
16632   \fi\markdownConTeXtReadAlignments}
16633 \def\markdownConTeXtRenderTableCell#1{%
16634   \advance\markdownConTeXtColumnCounter by 1\relax
16635   \markdownConTeXtTable=\expandafter{%
16636     \the\markdownConTeXtTable\bTD#1\eTD}%
16637   \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
16638   \else
16639     \expandafter\gobbleoneargument
16640   \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`.

```

16641 \ExplSyntaxOn
16642 \cs_gset:Npn
16643   \markdownRendererInputRawInlinePrototype#1#2
16644   {
16645     \str_case:nnF
16646       { #2 }
16647       {
16648         { latex }
16649         {

```

```

16650         \@@_plain_tex_default_input_raw_inline:nn
16651         { #1 }
16652         { context }
16653     }
16654 }
16655 {
16656     \@@_plain_tex_default_input_raw_inline:nn
16657     { #1 }
16658     { #2 }
16659 }
16660 }
16661 \cs_gset:Npn
16662   \markdownRendererInputRawBlockPrototype#1#2
16663   {
16664     \str_case:nnF
16665     { #2 }
16666     {
16667       { context }
16668       {
16669         \@@_plain_tex_default_input_raw_block:nn
16670         { #1 }
16671         { tex }
16672       }
16673     }
16674     {
16675       \@@_plain_tex_default_input_raw_block:nn
16676       { #1 }
16677       { #2 }
16678     }
16679   }
16680 \cs_gset_eq:NN
16681   \markdownRendererInputRawBlockPrototype
16682   \markdownRendererInputRawInlinePrototype
16683   \fi % Closes ` \markdownIfOption{plain}{\iffalse}{\iftrue}`
16684 \ExplSyntaxOff
16685 \stopmodule
16686 \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).

```

16687 \ExplSyntaxOn
16688 \str_if_eq:VVT
16689   \c_@@_top_layer_tl
16690   \c_@@_option_layer_context_tl
16691   {
16692     \use:c

```

```

16693     { ExplSyntaxOff }
16694 \@@_if_option:nF
16695     { noDefaults }
16696     {
16697       \@@_if_option:nTF
16698         { experimental }
16699         {
16700           \@@_setup:n
16701             { theme = witiko/markdown/defaults@experimental }
16702         }
16703         {
16704           \@@_setup:n
16705             { theme = witiko/markdown/defaults }
16706         }
16707     }
16708 \use:c
16709     { ExplSyntaxOn }
16710 }
16711 \ExplSyntaxOff
16712 \stopmodule
16713 \protect

```

## References

- [1] LuaTeX development team. *LuaTeX reference manual*. Version 1.21. Feb. 1, 2025. URL: <http://mirrors.ctan.org/systems/doc/luatex/luatex.pdf> (visited on 05/12/2025).
- [2] L<sup>A</sup>T<sub>E</sub>X Project. *l3kernel. L<sup>A</sup>T<sub>E</sub>X3 programming conventions*. Dec. 25, 2024. URL: <https://ctan.org/pkg/l3kernel> (visited on 01/06/2025).
- [3] Frank Mittelbach, Ulrike Fischer, and L<sup>A</sup>T<sub>E</sub>X Project. *The documentmetadata-support code*. June 1, 2024. URL: <https://mirrors.ctan.org/macros/latex/required/latex-lab/documentmetadata-support-code.pdf> (visited on 10/21/2024).
- [4] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: <https://www.muni.cz/en/research/projects/32984> (visited on 02/19/2018).
- [5] Anton Sotkov. *File transclusion syntax for Markdown*. Jan. 19, 2017. URL: <https://github.com/iainc/Markdown-Content-Blocks> (visited on 01/08/2018).
- [6] John MacFarlane. *Pandoc. a universal document converter*. 2022. URL: <https://pandoc.org/> (visited on 10/05/2022).

- [7] 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).
- [8] 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.
- [9] 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).
- [10] 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).
- [11] Vít Starý Novotný. *Versioned Themes*. Markdown Enhancement Proposal. Oct. 13, 2024. URL: <https://github.com/Witiko/markdown/discussions/514> (visited on 10/21/2024).
- [12] 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).
- [13] Vít Starý Novotný. *Routing YAML metadata to expl3 key-values*. Markdown Enhancement Proposal. Oct. 14, 2024. URL: <https://github.com/witiko/markdown/discussions/517> (visited on 01/06/2025).
- [14] Frank Mittelbach. *L<sup>A</sup>T<sub>E</sub>X’s hook management*. June 26, 2024. URL: <https://mirrors.ctan.org/macros/latex/base/lthooks-code.pdf> (visited on 10/02/2024).
- [15] 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).
- [16] Roberto Ierusalimsky. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.
- [17] Johannes Braams et al. *The L<sup>A</sup>T<sub>E</sub>X<sub>2<sub>ε</sub></sub> Sources*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/source2e.pdf> (visited on 01/08/2018).
- [18] 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.
- [19] 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

autoIdentifiers 21, 33, 86, 101

blankBeforeBlockquote	21
blankBeforeCodeFence	22
blankBeforeDivFence	22
blankBeforeHeading	22
blankBeforeList	23
bracketedSpans	23, 88, 464
breakableBlockquotes	23
cacheDir	4, 16, 19, 59, 60, 160, 172, 375, 396, 415
citationNbsps	24
citations	24, 91
codeSpans	25
contentBlocks	20, 25, 35
contentBlocksLanguageMap	20
contentLevel	26
debugExtensions	9, 20, 26, 318
debugExtensionsFileName	20, 26
defaultOptions	10, 52, 373, 374
definitionLists	26, 95
eagerCache	16, 372
ensureJekyllData	27
entities.char_entity	219
entities.dec_entity	218
entities.hex_entity	218
entities.hex_entity_with_x_char	218
escape_minimal	223
escape_programmatic_text	223
escape_typographic_text	223
expandtabs	280
expectJekyllData	27, 27
experimental	5, 17, 434
extensions	29, 167, 323
extensions.bracketed_spans	324
extensions.citations	325
extensions.content_blocks	329
extensions.definition_lists	332
extensions.fancy_lists	334
extensions.fenced_code	340
extensions.fenced_divs	346
extensions.header_attributes	350
extensions.inline_code_attributes	352

<code>extensions.jekyll_data</code>	368
<code>extensions.line_blocks</code>	352
<code>extensions.link_attributes</code>	354
<code>extensions.mark</code>	353
<code>extensions.notes</code>	355
<code>extensions.pipe_table</code>	358
<code>extensions.raw_inline</code>	362
<code>extensions.strike_through</code>	363
<code>extensions.subscripts</code>	363
<code>extensions.superscripts</code>	364
<code>extensions.tex_math</code>	365
<code>fancyLists</code>	30, 112–118, 434
<code>fencedCode</code>	30, 40, 92, 100, 118, 386, 389
<code>fencedCodeAttributes</code>	31, 86, 100, 389
<code>fencedDiv</code>	101
<code>fencedDivs</code>	31, 41
<code>finalizeCache</code>	17, 21, 32, 32, 59, 60, 159, 372, 374
<code>frozenCache</code>	21, 32, 60, 75, 76, 159, 387, 395
<code>frozenCacheCounter</code>	32, 374, 423, 424
<code>frozenCacheFileName</code>	21, 32, 59, 374
<code>\g_markdown_diagrams_infostrings_prop</code>	391
<code>gfmAutoIdentifiers</code>	21, 33, 86, 101
<code>hashEnumerators</code>	33
<code>headerAttributes</code>	33, 41, 86, 101
<code>html</code>	34, 104, 105, 447
<code>hybrid</code>	34, 34, 40, 46, 48, 62, 76, 120, 160, 223, 281, 423
<code>inlineCodeAttributes</code>	36, 86, 93
<code>inlineNotes</code>	36
<code>\input</code>	56, 374
<code>\inputmarkdown</code>	162, 164, 165, 465
<code>inputTempFileName</code>	60, 62, 417, 418, 420, 421
<code>\inputyaml</code>	162, 164, 465
<code>iterlines</code>	280
<code>jekyllData</code>	3, 27, 28, 36, 129–132, 134
<code>\l_file_search_path_seq</code>	422
<code>languages_json</code>	329, 329
<code>lineBlocks</code>	37, 108

linkAttributes	37, 86, 106, 109, 299, 460
mark	38, 110, 459
\markdown	155, 156, 427
markdown	154, 154, 155, 425, 426
markdown*	154, 154, 159, 425
\markdownBegin	54, 54, 55, 56, 152, 154, 155, 162, 163
\markdownCleanup	416
\markdownConvert	416
\markdownEnd	54, 54, 55, 56, 152, 154–156, 162, 163
\markdownError	152, 152
\markdownEscape	54, 57, 424
\markdownIfOption	58
\markdownIfSnippetExists	80
\markdownInfo	152, 152
\markdownInput	54, 56, 154, 156, 159, 164, 422, 425
\markdownInputFilename	415
\markdownInputFileStream	416
\markdownInputPlainTeX	425
\markdownLoadPlainTeXTheme	160, 166, 385
\markdownLuaExecute	419, 422
\markdownLuaOptions	412, 416
\markdownMakeOther	152, 465
\markdownOptionFinalizeCache	59
\markdownOptionFrozenCache	59
\markdownOptionHybrid	62
\markdownOptionInputTempFileName	60
\markdownOptionNoDefaults	61
\markdownOptionOutputDir	60, 60, 63, 64
\markdownOptionPlain	61
\markdownOptionStripPercentSigns	62
\markdownOutputFileStream	416
\markdownPrepare	415
\markdownPrepareInputFilename	415
\markdownPrepareLuaOptions	412
\markdownReadAndConvert	152, 416, 425–427, 466
\markdownReadAndConvertProcessLine	418, 418
\markdownReadAndConvertStripPercentSigns	417
\markdownReadAndConvertTab	416
\markdownRendererAttributeClassName	86
\markdownRendererAttributeIdentifier	86
\markdownRendererAttributeKeyValue	86



<code>\markdownRendererBlockQuoteBegin</code>	87
<code>\markdownRendererBlockQuoteEnd</code>	87
<code>\markdownRendererBracketedSpanAttributeContextBegin</code>	88
<code>\markdownRendererBracketedSpanAttributeContextEnd</code>	88
<code>\markdownRendererCite</code>	91, 91
<code>\markdownRendererCodeSpan</code>	93
<code>\markdownRendererCodeSpanAttributeContextBegin</code>	93
<code>\markdownRendererCodeSpanAttributeContextEnd</code>	93
<code>\markdownRendererContentBlock</code>	94, 94
<code>\markdownRendererContentBlockCode</code>	95
<code>\markdownRendererContentBlockOnlineImage</code>	94
<code>\markdownRendererDisplayMath</code>	126
<code>\markdownRendererDlBegin</code>	96
<code>\markdownRendererDlBeginTight</code>	96
<code>\markdownRendererDlDefinitionBegin</code>	97
<code>\markdownRendererDlDefinitionEnd</code>	97
<code>\markdownRendererDlEnd</code>	98
<code>\markdownRendererDlEndTight</code>	98
<code>\markdownRendererDlItem</code>	96
<code>\markdownRendererDlItemEnd</code>	97
<code>\markdownRendererDocumentBegin</code>	111
<code>\markdownRendererDocumentEnd</code>	111
<code>\markdownRendererEllipsis</code>	42, 99
<code>\markdownRendererEmphasis</code>	99, 138
<code>\markdownRendererError</code>	128
<code>\markdownRendererFancyOlBegin</code>	113, 114
<code>\markdownRendererFancyOlBeginTight</code>	113
<code>\markdownRendererFancyOlEnd</code>	117
<code>\markdownRendererFancyOlEndTight</code>	118
<code>\markdownRendererFancyOlItem</code>	115
<code>\markdownRendererFancyOlItemEnd</code>	116
<code>\markdownRendererFancyOlItemWithNumber</code>	116
<code>\markdownRendererFencedCodeAttributeContextBegin</code>	100
<code>\markdownRendererFencedCodeAttributeContextEnd</code>	100
<code>\markdownRendererFencedDivAttributeContextBegin</code>	101
<code>\markdownRendererFencedDivAttributeContextEnd</code>	101
<code>\markdownRendererHalfTickedBox</code>	127
<code>\markdownRendererHardLineBreak</code>	109
<code>\markdownRendererHeaderAttributeContextBegin</code>	101
<code>\markdownRendererHeaderAttributeContextEnd</code>	101
<code>\markdownRendererHeadingFive</code>	104
<code>\markdownRendererHeadingFour</code>	103

<code>\markdownRendererHeadingOne</code>	102
<code>\markdownRendererHeadingSix</code>	104
<code>\markdownRendererHeadingThree</code>	103
<code>\markdownRendererHeadingTwo</code>	102
<code>\markdownRendererImage</code>	106
<code>\markdownRendererImageAttributeContextBegin</code>	106
<code>\markdownRendererImageAttributeContextEnd</code>	106
<code>\markdownRendererInlineHtmlComment</code>	104
<code>\markdownRendererInlineHtmlTag</code>	105
<code>\markdownRendererInlineMath</code>	126
<code>\markdownRendererInputBlockHtmlElement</code>	105
<code>\markdownRendererInputFencedCode</code>	92
<code>\markdownRendererInputRawBlock</code>	118
<code>\markdownRendererInputRawInline</code>	118
<code>\markdownRendererInputVerbatim</code>	92
<code>\markdownRendererInterblockSeparator</code>	107
<code>\markdownRendererJekyllDataBegin</code>	129
<code>\markdownRendererJekyllDataBoolean</code>	131
<code>\markdownRendererJekyllDataEmpty</code>	134
<code>\markdownRendererJekyllDataEnd</code>	129
<code>\markdownRendererJekyllDataMappingBegin</code>	129
<code>\markdownRendererJekyllDataMappingEnd</code>	130
<code>\markdownRendererJekyllDataNumber</code>	131
<code>\markdownRendererJekyllDataProgrammaticString</code>	132, 132, 133
<code>\markdownRendererJekyllDataSequenceBegin</code>	130
<code>\markdownRendererJekyllDataSequenceEnd</code>	131
<code>\markdownRendererJekyllDataString</code>	133, 137
<code>\markdownRendererJekyllDataStringPrototype</code>	147
<code>\markdownRendererJekyllDataTypographicString</code>	132, 132, 133, 369
<code>\markdownRendererLineBlockBegin</code>	108
<code>\markdownRendererLineBlockEnd</code>	108
<code>\markdownRendererLink</code>	109, 138
<code>\markdownRendererLinkAttributeContextBegin</code>	110
<code>\markdownRendererLinkAttributeContextEnd</code>	110
<code>\markdownRendererMark</code>	110
<code>\markdownRendererNbsp</code>	111
<code>\markdownRendererNote</code>	112
<code>\markdownRendererOlBegin</code>	112
<code>\markdownRendererOlBeginTight</code>	113
<code>\markdownRendererOlEnd</code>	116
<code>\markdownRendererOlEndTight</code>	117
<code>\markdownRendererOlItem</code>	42, 114

<code>\markdownRendererOlItemEnd</code>	114
<code>\markdownRendererOlItemWithNumber</code>	42, 115
<code>\markdownRendererParagraphSeparator</code>	107
<code>\markdownRendererReplacementCharacter</code>	120
<code>\markdownRendererSectionBegin</code>	119
<code>\markdownRendererSectionEnd</code>	119
<code>\markdownRendererSoftLineBreak</code>	108
<code>\markdownRendererStrikeThrough</code>	123
<code>\markdownRendererStrongEmphasis</code>	99
<code>\markdownRendererSubscript</code>	123
<code>\markdownRendererSuperscript</code>	124
<code>\markdownRendererTable</code>	125
<code>\markdownRendererTableAttributeContextBegin</code>	124
<code>\markdownRendererTableAttributeContextEnd</code>	124
<code>\markdownRendererTextCite</code>	91
<code>\markdownRendererThematicBreak</code>	126
<code>\markdownRendererTickedBox</code>	127
<code>\markdownRendererUlBegin</code>	89
<code>\markdownRendererUlBeginTight</code>	89
<code>\markdownRendererUlEnd</code>	90
<code>\markdownRendererUlEndTight</code>	91
<code>\markdownRendererUlItem</code>	89
<code>\markdownRendererUlItemEnd</code>	90
<code>\markdownRendererUntickedBox</code>	127
<code>\markdownRendererWarning</code>	128
<code>\markdownSetup</code>	58, 58, 62, 158, 159, 165, 426, 433
<code>\markdownSetupSnippet</code>	79, 79
<code>\markdownThemeVersion</code>	69, 69, 70
<code>\markdownWarning</code>	152, 152
<code>\markinline</code>	54, 55, 56, 154, 156, 420, 424
<code>\markinlinePlainTeX</code>	424
<code>new</code>	7, 18, 372, 374
<code>notes</code>	38, 112
<code>parsers</code>	239, 279
<code>parsers.commented_line</code>	260
<code>parsers.unicode_data</code>	240
<code>pipeTables</code>	7, 39, 45, 125
<code>preserveTabs</code>	39, 43, 280
<code>rawAttribute</code>	35, 40, 40, 118
<code>reader</code>	8, 30, 167, 239, 279, 324

reader->add_special_character	8, 9, 30, 318
reader->auto_link_email	307
reader->auto_link_url	307
reader->create_parser	280
reader->finalize_grammar	313, 379
reader->initialize_named_group	318
reader->insert_pattern	8, 9, 30, 314, 320
reader->lookup_note_reference	292
reader->lookup_reference	292
reader->normalize_tag	279
reader->options	279
reader->parser_functions	280
reader->parser_functions.name	280
reader->parsers	279, 279
reader->register_link	292
reader->update_rule	314, 317, 320
reader->writer	279
reader.new	279, 279, 379
relativeReferences	40
\setupmarkdown	165, 165
\setupyaml	165
shiftHeadings	7, 41
singletonCache	18
slice	7, 41, 220, 232, 233
smartEllipses	42, 99, 160
\startmarkdown	162, 162, 163, 466
startNumber	42, 114–116
\startyaml	162, 163, 466
\stopmarkdown	162, 162, 163, 466
\stopyaml	162, 163, 466
strikeThrough	42, 123, 459
stripIndent	43, 280
stripPercentSigns	417
subscripts	43, 123
superscripts	44, 124
syntax	315, 319
tableAttributes	44, 124, 456
tableCaptions	7, 44, 45, 124
taskLists	45, 127, 447
texComments	46, 281

<code>texMathDollars</code>	35, 46, 126
<code>texMathDoubleBackslash</code>	35, 47, 126
<code>texMathSingleBackslash</code>	35, 47, 126
<code>tightLists</code>	47, 89, 91, 96, 98, 113, 114, 117, 118, 434
<code>underscores</code>	48
<code>unicodeNormalization</code>	18, 19
<code>unicodeNormalizationForm</code>	18, 19
<code>util.cache</code>	168, 168
<code>util.cache_verbatim</code>	168
<code>util.encode_json_string</code>	168
<code>util.err</code>	168
<code>util.escaper</code>	171
<code>util.expand_tabs_in_line</code>	169
<code>util.flatten</code>	169
<code>util.intersperse</code>	170
<code>util.map</code>	171
<code>util.pathname</code>	172
<code>util.rope_last</code>	170
<code>util.rope_to_string</code>	170
<code>util.salt</code>	172
<code>util.table_copy</code>	168
<code>util.walk</code>	169, 170
<code>util.warning</code>	172
<code>walkable_syntax</code>	8, 20, 26, 313, 314, 317–319
<code>writer</code>	167, 167, 219, 324
<code>writer-&gt;active_attributes</code>	231, 231, 232
<code>writer-&gt;attribute_type_levels</code>	231
<code>writer-&gt;attributes</code>	229
<code>writer-&gt;block_html_element</code>	227
<code>writer-&gt;blockquote</code>	228
<code>writer-&gt;bulletitem</code>	226
<code>writer-&gt;bulletlist</code>	225
<code>writer-&gt;citations</code>	325
<code>writer-&gt;code</code>	224
<code>writer-&gt;contentblock</code>	330
<code>writer-&gt;defer_call</code>	239, 239
<code>writer-&gt;definitionlist</code>	332
<code>writer-&gt;display_math</code>	365
<code>writer-&gt;div_begin</code>	346
<code>writer-&gt;div_end</code>	346

writer->document	228
writer->ellipsis	221
writer->emphasis	227
writer->error	224
writer->escape	223
writer->escaped_chars	222, 223
writer->escaped_minimal_strings	222, 223
writer->escaped_strings	222
writer->escaped_uri_chars	222, 223
writer->fancyitem	336
writer->fancylist	335
writer->fencedCode	341
writer->flatten_inlines	220, 220
writer->get_state	238
writer->hard_line_break	221
writer->heading	236
writer->identifier	223
writer->image	225
writer->infostring	223
writer->inline_html_comment	227
writer->inline_html_tag	227
writer->inline_math	365
writer->interblocksep	221
writer->is_writing	220, 220
writer->jekyllData	369
writer->lineblock	352
writer->link	224
writer->mark	353
writer->math	223
writer->nbsp	220
writer->note	356
writer->options	219
writer->ordereditem	226
writer->orderedlist	226
writer->paragraph	221
writer->paragraphsep	221
writer->plain	220
writer->pop_attributes	231, 232
writer->push_attributes	231, 231, 232
writer->rawBlock	341
writer->rawInline	362
writer->set_state	238

writer->slice_begin	220
writer->slice_end	220
writer->soft_line_break	221
writer->space	220
writer->span	324
writer->strike_through	363
writer->string	223
writer->strong	227
writer->subscript	363
writer->superscript	364
writer->table	359
writer->thematic_break	222
writer->textbox	227
writer->undosep	221, 322
writer->uri	223
writer->verbatim	228
writer->warning	172, 223
writer.new	219, 219, 379
\yaml	156
yaml	154, 155, 156, 425
\yamlBegin	54, 55, 152, 155, 163
\yamlEnd	54, 55, 152, 155, 156, 163
\yamlInput	54, 56, 154, 157, 164, 425
\yamlSetup	58