

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

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

Version 3.12.0-0-g746cfc56  
2025-10-31

## 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 . . . .	168
1.2	Feedback . . . . .	6	3.2	Plain T <sub>E</sub> X Implementation	407
1.3	Acknowledgements . . . .	7	3.3	L <sup>A</sup> T <sub>E</sub> X Implementation . .	450
<b>2</b>	<b>Interfaces</b>	<b>7</b>	3.4	ConT <sub>E</sub> Xt Implementation	490
2.1	Lua Interface . . . . .	7			
2.2	Plain T <sub>E</sub> X Interface . . . .	53	<b>References</b>		<b>501</b>
2.3	L <sup>A</sup> T <sub>E</sub> X Interface . . . . .	153	<b>Index</b>		<b>503</b>
2.4	ConT <sub>E</sub> Xt Interface . . . .	162			

## 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 . . . . .	286

## 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             "2016-2025 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>.

Only load the package outside the ConT<sub>E</sub>Xt format, where we can use the resolvers API [1, Section 11.5].

```
16 if resolvers == nil then
17   (function()
```

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

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

All the abovelisted modules are statically linked into the current version of the LuaT<sub>E</sub>X engine [2, Section 4.3]. Beside these, we also include the following third-party Lua libraries:

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

```
26 hard lua-tinyyaml
```

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

The plain T<sub>E</sub>X part of the package requires that the plain T<sub>E</sub>X format (or its superset) is loaded, all the Lua prerequisites (see Section 1.1.1), and the following packages:

**expl3** A package that enables the expl3 language [3] from the L<sup>A</sup>T<sub>E</sub>X3 kernel in T<sub>E</sub>X Live ≤ 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 [2, Section 4.2.4].

The Lua File System module is statically linked into the LuaTeX engine [2, 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 [4] 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 [5] 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

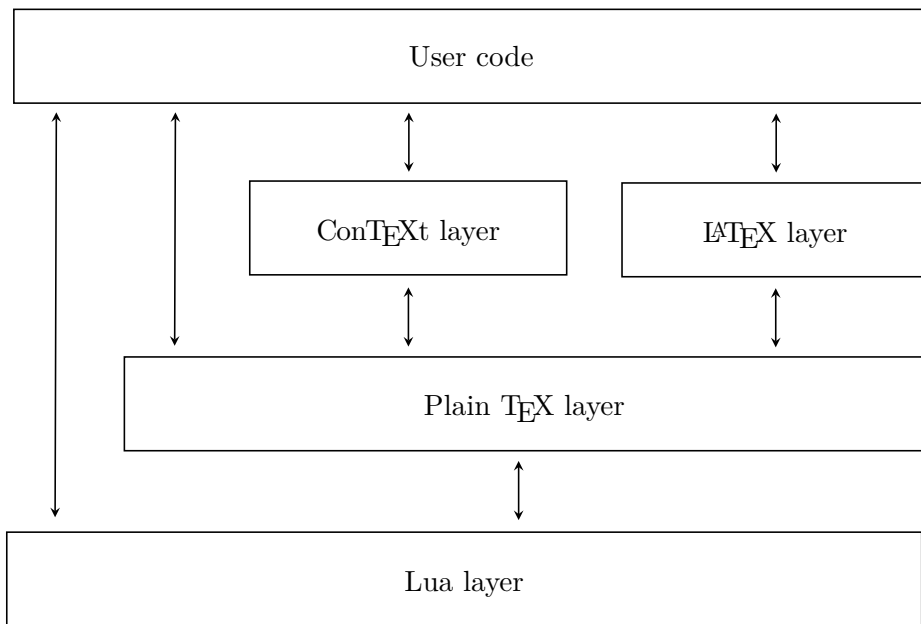
The Lua interface provides the conversion from UTF-8 encoded markdown to plain T<sub>E</sub>X. This interface is used by the plain T<sub>E</sub>X implementation (see Section 3.2) and will be of interest to the developers of other packages and Lua modules.

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: }
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
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
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_gset:NV
424   \g_@@_unquoted_jobname_str
425   \c_sys_jobname_str
426 \bool_new:N
427   \g_@@_jobname_quoted_bool
```

```

428 \regex_replace_all:nnNtF
429 { \A ("|' ) ( .* ) ("|' ) \Z }
430 { \2 }
431 \g_@@_unquoted_jobname_str
432 {
433     \bool_gset_true:N
434     \g_@@_jobname_quoted_bool
435 }
436 {
437     \bool_gset_false:N
438     \g_@@_jobname_quoted_bool
439 }
440 \@@_add_lua_option:nnn
441 { cacheDir }
442 { path }
443 {
444     \markdownOptionOutputDir
445     / _markdown_
446     \str_use:N
447     \g_@@_unquoted_jobname_str
448 }
449 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.

```

450 \@@_add_lua_option:nnn
451 { contentBlocksLanguageMap }
452 { path }
453 { markdown-languages.json }
454 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.

```

455 \@@_add_lua_option:nnn
456   { debugExtensionsFileName }
457   { path }
458   {
459     \markdownOptionOutputDir
460     /
461     \str_use:N
462     \g_@@_unquoted_jobname_str
463     .debug-extensions.json
464   }
465 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 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.

```

466 \@@_add_lua_option:nnn
467   { frozenCacheFileName }
468   { path }
469   { \markdownOptionCacheDir / frozenCache.tex }
470 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`.

```

471 \@@_add_lua_option:nnn

```

---

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

```

472 { autoIdentifiers }
473 { boolean }
474 { false }
475 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.

```

476 \@@_add_lua_option:nnn
477 { blankBeforeBlockquote }
478 { boolean }
479 { false }
480 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.

```

481 \@@_add_lua_option:nnn
482 { blankBeforeCodeFence }
483 { boolean }
484 { false }
485 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.

```

486 \@@_add_lua_option:nnn
487 { blankBeforeDivFence }
488 { boolean }
489 { false }
490 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.

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

```
496 \@@_add_lua_option:nnn
497 { blankBeforeList }
498 { boolean }
499 { false }
500 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.

```
501 \@@_add_lua_option:nnn
502 { bracketedSpans }
503 { boolean }
504 { false }
505 defaultOptions.bracketedSpans = false
```

---

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

`breakableBlockquotes=true, false`

default: true

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

```
506 \@@_add_lua_option:nnn
507 { breakableBlockquotes }
508 { boolean }
509 { true }

510 defaultOptions.breakableBlockquotes = true
```

`citationNbsps=true, false`

default: false

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

```
511 \@@_add_lua_option:nnn
512 { citationNbsps }
513 { boolean }
514 { true }

515 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,

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



```
chap. 1]. Here is one with the name of the author
suppressed -@doe99.
```

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

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

520 defaultOptions.citations = false
```

**codeSpans=true, false**

default: true

**true**      Enable the code span syntax:

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

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

```
``This is a quote.``
```

```
521 \@@_add_lua_option:nnn
522   { codeSpans }
523   { boolean }
524   { true }

525 defaultOptions.codeSpans = true
```

**contentBlocks=true, false**

default: false

true

: Enable the iA Writer content blocks syntax extension [6]:

```
``` 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.

```
526 \@@_add_lua_option:nnn
527   { contentBlocks }
528   { boolean }
529   { false }

530 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
```

```
531 \@@_add_lua_option:nnn
532   { contentLevel }
533   { string }
534   { block }

535 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.

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

540 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.

```
541 \@@_add_lua_option:nnn
542   { definitionLists }
543   { boolean }
544   { false }

545 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.

```
546 \@@_add_lua_option:nnn
547   { ensureJekyllData }
548   { boolean }
549   { false }

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

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

```

\begin{markdown}
- this
- is
- YAML
\end{markdown}
\end{document}

```

```

551 \@@_add_lua_option:nnn
552   { expectJekyllData }
553   { boolean }
554   { false }

555 defaultOptions.expectJekyllData = false

```

`extensions=<filenames>`

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

```
556 metadata.user_extension_api_version = 2
557 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.

```
558 \cs_generate_variant:Nn
559   \@@_add_lua_option:nnn
560   { nnV }
561 \@@_add_lua_option:nnV
562   { extensions }
563   { clist }
564   \c_empty_clist
565 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.

```
566 \@@_add_lua_option:nnn
567   { fancyLists }
568   { boolean }
569   { false }
570 defaultOptions.fancyLists = false
```

---

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

`fencedCode=true, false`

default: `true`

`true` Enable the commonmark fenced code block extension:

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

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

`false` Disable the commonmark fenced code block extension.

```
571 \@@_add_lua_option:nnn
572 { fencedCode }
573 { boolean }
574 { true }

575 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.

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

```

576 \@@_add_lua_option:nnn
577   { fencedCodeAttributes }
578   { boolean }
579   { false }

580 defaultOptions.fencedCodeAttributes = false

```

**fencedDivs**=true, false

default: false

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

```

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

And another.
:::::

```

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

```

581 \@@_add_lua_option:nnn
582   { fencedDivs }
583   { boolean }
584   { false }

585 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.

```

586 \@@_add_lua_option:nnn
587   { finalizeCache }
588   { boolean }
589   { false }

590 defaultOptions.finalizeCache = false

```

---

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



`frozenCacheCounter`= $\langle number \rangle$  default: 0

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

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

```
591 \@@_add_lua_option:nnn
592   { frozenCacheCounter }
593   { counter }
594   { 0 }

595 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`.

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

600 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.

---

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

```

601 \@@_add_lua_option:nnn
602   { hashEnumerators }
603   { boolean }
604   { false }

605 defaultOptions.hashEnumerators = false

```

`headerAttributes=true, false`

default: false

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

```

# My first heading {#foo}

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

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

```

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

```

606 \@@_add_lua_option:nnn
607   { headerAttributes }
608   { boolean }
609   { false }

610 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.

```

611 \@@_add_lua_option:nnn
612   { html }
613   { boolean }
614   { true }

615 defaultOptions.html = true

```

`hybrid=true, false`

default: `false`

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

The `hybrid` option makes it difficult to untangle TeX 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 TeX 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:
\[distance[i] =
  \begin{dcases}
    a & b \\
    c & d
  \end{dcases}
\]
```

```
616 \@@_add_lua_option:nnn
617   { hybrid }
618   { boolean }
619   { false }
620 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.

```
621 \@@_add_lua_option:nnn
622   { inlineCodeAttributes }
623   { boolean }
624   { false }
625 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.]
```

<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).

`false`      Disable the Pandoc inline note syntax extension.

```
626 \@@_add_lua_option:nnn
627   { inlineNotes }
628   { boolean }
629   { false }
630 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]
abstract: |
  This is the abstract.

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

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

```
631 \@@_add_lua_option:nnn
632   { jeekyllData }
633   { boolean }
634   { false }
635 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}
```

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

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

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

```
636 \@@_add_lua_option:nnn
637 { linkAttributes }
638 { boolean }
639 { false }

640 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

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

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

645 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.

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

650 defaultOptions.mark = false
```

---

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

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

`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.
```

`false` Disable the Pandoc note syntax extension.

```
651 \@@_add_lua_option:nnn
652   { notes }
653   { boolean }
654   { false }

655 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.

---

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

```

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

660 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.

```

661 \@@_add_lua_option:nnn
662   { preserveTabs }
663   { boolean }
664   { true }

665 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.

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



```

666 \@@_add_lua_option:nnn
667   { rawAttribute }
668   { boolean }
669   { false }

670 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!

`false` Disable relative references in autolinks.

```

671 \@@_add_lua_option:nnn
672   { relativeReferences }
673   { boolean }
674   { false }

675 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.

```

676 \@@_add_lua_option:nnn
677   { shiftHeadings }
678   { number }
679   { 0 }

680 defaultOptions.shiftHeadings = 0

```

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

`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.

```
681 \@@_add_lua_option:nnn
682   { slice }
683   { slice }
684   { ^~$ }
685 defaultOptions.slice = "^ $"
```

`smartEllipses`=`true, false` default: `false`

**true** Convert any ellipses in the input to the `\markdownRenderEllipsis`  $\mathrm{T}_{\mathrm{E}}\mathrm{X}$  macro.

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

```
686 \@@_add_lua_option:nnn
687   { smartEllipses }
688   { boolean }
689   { false }
690 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 `\markdownRenderOliItemWithNumber`  $\mathrm{T}_{\mathrm{E}}\mathrm{X}$  macro.

**false** Ignore the numbers in the ordered list items. Each item will only produce a `\markdownRenderer01Item` TeX macro.

```
691 \@@_add_lua_option:nnn
692 { startNumber }
693 { boolean }
694 { true }
695 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.

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

**stripIndent**=true, false default: false

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

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

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

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

---

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

`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.

```
706 \@@_add_lua_option:nnn
707 { subscripts }
708 { boolean }
709 { false }

710 defaultOptions.subscripts = false
```

`superscripts=true, false` default: false

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

2^10^ is 1024.

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

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

715 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}
```
```

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

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

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

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

720 defaultOptions.tableAttributes = false
```

**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.

```
721 \@@_add_lua_option:nnn
722   { tableCaptions }
723   { boolean }
724   { false }

725 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.

<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).

```

726 \@@_add_lua_option:nnn
727   { taskLists }
728   { boolean }
729   { false }

730 defaultOptions.taskLists = false

```

`texComments=true, false`

default: false

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

```

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

```

Always enabled when `hybrid` is enabled.

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

```

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

735 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.

```

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

740 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.

```
741 \@@_add_lua_option:nnn
742   { texMathDoubleBackslash }
743   { boolean }
744   { false }

745 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.

```
746 \@@_add_lua_option:nnn
747   { texMathSingleBackslash }
748   { boolean }
749   { false }

750 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.

```

751 \@@_add_lua_option:nnn
752   { tightLists }
753   { boolean }
754   { true }

755 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.

```

756 \@@_add_lua_option:nnn
757   { underscores }
758   { boolean }
759   { true }
760 \ExplSyntaxOff

761 defaultOptions.underscores = true

```

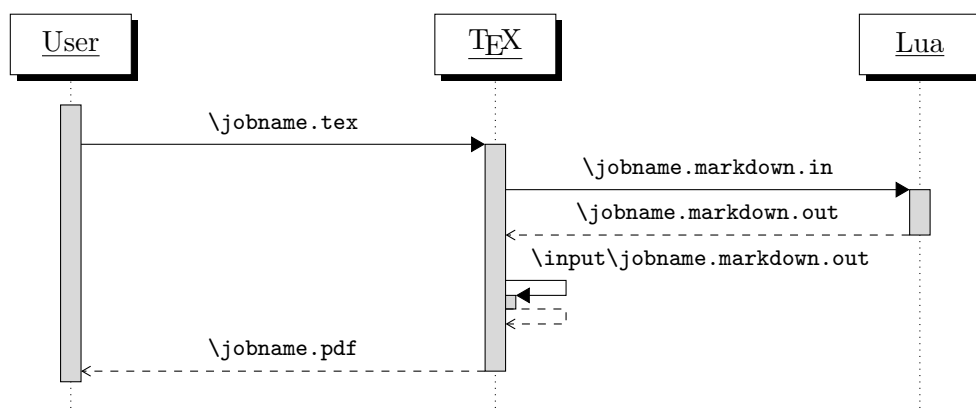


### 2.1.7 Command-Line Interface

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.

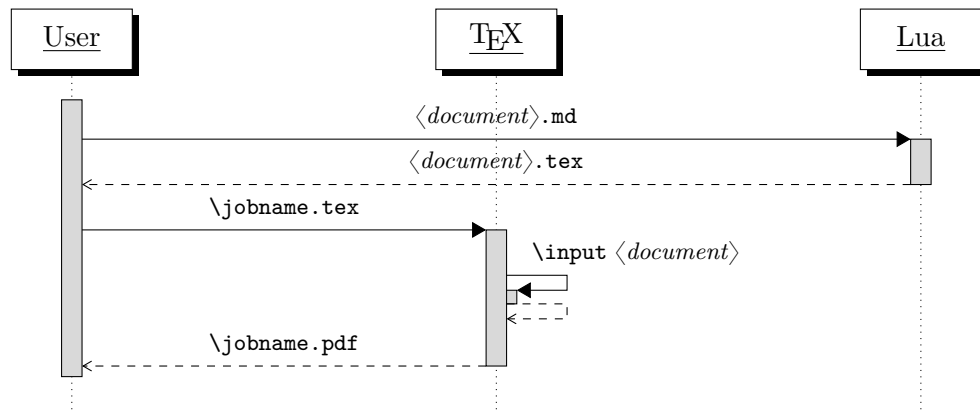
A solution to the above problems is to decouple the conversion from the typesetting. For this reason, a command-line Lua interface for converting a markdown document to  $\text{\TeX}$  is also provided, see Figure 3.



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

```

762 .TH MARKDOWN2TEX 1 "(((LASTMODIFIED)))"
763 .SH NAME
764 markdown2tex \- convert .md files to .tex
765 .SH SYNOPSIS
</lua-cli-manpage> <*lua-cli>
766 local HELP_STRING = "Usage: " .. [[
</lua-cli> <*lua-cli,lua-cli-manpage>
767 markdown2tex [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
768
</lua-cli,lua-cli-manpage> <*lua-cli-manpage>
769 .SH DESCRIPTION
  
```



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

```

770 % \end{macrocode}
771 </lua-cli-manpage>
772 <*lua-cli, lua-cli-manpage>
773 % \begin{macrocode}
774 OPTIONS are documented in Section 2.2.1 of the Markdown Package User
775 Manual (https://ctan.org/pkg/markdown).
776
777 When OUTPUT_FILE is unspecified, the result of the conversion will be
778 written to the standard output. When INPUT_FILE is also unspecified, the
779 result of the conversion will be read from the standard input.
780 % \end{macrocode}
781 </lua-cli, lua-cli-manpage>
782 <*lua-cli>
783 % \begin{macrocode}
784
785 Report bugs to: witiko@mail.muni.cz
786 Markdown package home page: <https://github.com/witiko/markdown>]]
787
788 local VERSION_STRING = [[
789 markdown2tex (Markdown) ]] .. metadata.version .. [[
790
791 Copyright (C) ]] .. table.concat(metadata.copyright,
792                                     "\nCopyright (C) ") .. [[
793
794 License: ]] .. metadata.license
795
796 local function warn(s)
797   io.stderr:write("Warning: " .. s .. "\n")
798 end

```

```

799
800 local function error(s)
801   io.stderr:write("Error: " .. s .. "\n")
802   os.exit(1)
803 end

```

To make it easier to copy-and-paste options from Pandoc [7] 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 [8] also show that snake\_case is faster to read than camelCase.

```

804 local function camel_case(option_name)
805   local cased_option_name = option_name:gsub("_(%l)", function(match)
806     return match:sub(2, 2):upper()
807   end)
808   return cased_option_name
809 end
810
811 local function snake_case(option_name)
812   local cased_option_name = option_name:gsub("%l%u", function(match)
813     return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()
814   end)
815   return cased_option_name
816 end
817
818 local cases = {camel_case, snake_case}
819 local various_case_options = {}
820 for option_name, _ in pairs(defaultOptions) do
821   for _, case in ipairs(cases) do
822     various_case_options[case(option_name)] = option_name
823   end
824 end
825
826 local process_options = true
827 local options = {}
828 local input_filename
829 local output_filename
830 for i = 1, #arg do
831   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.

```

832     if arg[i] == "--" then
833       process_options = false
834       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.

```

835     elseif arg[i]:match("=") then
836         local key, value = arg[i]:match("(.-)=(.*)")
837         if defaultOptions[key] == nil and
838             various_case_options[key] ~= nil then
839             key = various_case_options[key]
840         end

```

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

```

841         local default_type = type(defaultOptions[key])
842         if default_type == "boolean" then
843             options[key] = (value == "true")
844         elseif default_type == "number" then
845             options[key] = tonumber(value)
846         elseif default_type == "table" then
847             options[key] = {}
848             for item in value:gmatch("[^,]+") do
849                 table.insert(options[key], item)
850             end
851         else
852             if default_type ~= "string" then
853                 if default_type == "nil" then
854                     warn('Option "' .. key .. '" not recognized.')
855                 else
856                     warn('Option "' .. key .. '" type not recognized, ' ..
857                         'please file a report to the package maintainer.')
858                 end
859                 warn('Parsing the ' .. 'value "' .. value .. '" of option "' ..
860                     key .. '" as a string.')
861             end
862             options[key] = value
863         end
864         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.

```

865     elseif arg[i] == "--help" or arg[i] == "-h" then
866         print(HELP_STRING)
867         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.

```

868     elseif arg[i] == "--version" or arg[i] == "-v" then
869         print(VERSION_STRING)
870         os.exit()
871     end
872 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.

```

873 if input_filename == nil then
874     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.

```

875 elseif output_filename == nil then
876     output_filename = arg[i]
877 else
878     error('Unexpected argument: "' .. arg[i] .. '".')
879 end
880 ::continue::
881 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.

```

882 \def\markdownLastModified{((LASTMODIFIED))}%
883 \def\markdownVersion{((VERSION))}%

```

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

```
\input markdown
```

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

### 2.2.1 Typesetting Markdown and YAML

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

#### 2.2.1.1 Typesetting Markdown and YAML directly

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

```
884 \let\markdownBegin\relax
885 \let\markdownEnd\relax
```

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

There are several limitations to the macros you need to be aware of:

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

Another limitation concerns spaces at the right end of an input line. In markdown, these are used to produce a forced line break. However, any such spaces are removed before the lines enter the input buffer of T<sub>E</sub>X [9, p. 46]. As a corollary, the `\markdownBegin` macro also ignores them.

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

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

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

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

```

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

```

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

```

886 \let\uyamlBegin\uyamlBegin
887 \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}
\markdownBegin
title: _Hello_ **world** ...
author: John Doe
\markdownEnd
\bye

```

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

```

888 \let\markinline\relax

```

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

```

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

```

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

```
\input markdown
\markdownSetup{contentLevel=inline}
\markdownBegin
_Hello_ **world** ...
\markdownEnd
\bye
```

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

### 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.

```
889 \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.

```
890 \def\yamlInput#1{%
891   \begingroup
892   \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
893   \markdownInput{#1}%
894   \endgroup
895 }%
```

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.

```

896 \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.

```

897 \ExplSyntaxOn
898 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
899 \cs_generate_variant:Nn
900   \tl_const:Nn
901   { NV }
902 \tl_if_exist:NF
903   \c_@@_top_layer_tl
904   {
905     \tl_const:NV
906       \c_@@_top_layer_tl
907       \c_@@_option_layer_plain_tex_tl
908   }

```

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

```

909 \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.

```

910 \prop_new:N \g_@@_plain_tex_option_types_prop
911 \prop_new:N \g_@@_default_plain_tex_options_prop
912 \seq_gput_right:NV
913   \g_@@_option_layers_seq
914   \c_@@_option_layer_plain_tex_tl
915 \cs_new:Nn
916   \@@_add_plain_tex_option:nnn
917   {
918     \@@_add_option:Vnnn
919     \c_@@_option_layer_plain_tex_tl
920     { #1 }
921     { #2 }
922     { #3 }
923   }

```

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.

```

924 \cs_new:Nn
925   \@@_setup:n
926   {
927     \keys_set:nn
928       { markdown/options }
929       { #1 }
930   }
931 \cs_gset_eq:NN
932   \markdownSetup
933   \@@_setup:n

```

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

```

934 \cs_gset_eq:NN
935   \yamlSetup
936   \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.

```

937 \prg_new_conditional:Nnn
938   \@@_if_option:n
939   { TF, T, F }
940   {

```

```

941 \@@_get_option_type:nN
942 { #1 }
943 \l_tmpa_tl
944 \str_if_eq:NNF
945 \l_tmpa_tl
946 \c_@@_option_type_boolean_tl
947 {
948   \msg_error:nnxx
949   { markdown }
950   { expected-boolean-option }
951   { #1 }
952   { \l_tmpa_tl }
953 }
954 \@@_get_option_value:nN
955 { #1 }
956 \l_tmpa_tl
957 \str_if_eq:NNTF
958 \l_tmpa_tl
959 \c_@@_option_value_true_tl
960 { \prg_return_true: }
961 { \prg_return_false: }
962 }
963 \msg_new:nnn
964 { markdown }
965 { expected-boolean-option }
966 {
967   Option~#1~has~type~#2,~
968   but~a~boolean~was~expected.
969 }
970 \let
971 \markdownIfOption
972 \@@_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`.

```

973 \@@_add_plain_tex_option:nnn
974 { frozenCache }

```

```

975 { boolean }
976 { 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 T<sub>E</sub>X document to populate and finalize the cache.
5. Enable the `frozenCache` option.
6. Publish the source code of the plain T<sub>E</sub>X document and the `cacheDir` directory.

**2.2.2.2 File and Directory Names** The `\markdownOptionInputTempFileName` macro sets the filename of the temporary input file that is created during the buffering of markdown text from a T<sub>E</sub>X source. It defaults to `\jobname.markdown.in`.

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

```

977 \tl_set:Nn
978   \l_tmpa_tl
979   {
980     \str_use:N
981       \g_@@_unquoted_jobname_str
982       .markdown.in
983   }
984 \bool_if:NT
985   \g_@@_jobname_quoted_bool
986   {
987     \tl_put_left:Nn
988       \l_tmpa_tl
989       { " }
990     \tl_put_right:Nn
991       \l_tmpa_tl
992       { " }
993   }
994 \cs_generate_variant:Nn
995   \@@_add_plain_tex_option:nnn
996   { nnV }
997 \@@_add_plain_tex_option:nnV
998   { inputTempFileName }
999   { path }
1000 \l_tmpa_tl

```

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 T<sub>E</sub>X implementation. The option defaults to . or, since T<sub>E</sub>X Live 2024, to the value of the `-output-directory` option of your T<sub>E</sub>X engine.

In MikTeX, this automatic detection is currently only supported with LuaTeX<sup>33</sup>. If you need to use MikTeX and cannot use 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 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.

```
1001 \@@_add_plain_tex_option:nnn
1002   { outputDir }
1003   { path }
1004   { . }
```

### 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 2.2.3). 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 ConTeXt. 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 ConTeXt 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.

```
1005 \@@_add_plain_tex_option:nnn
```

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

```

1006 { plain }
1007 { boolean }
1008 { 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  $\text{\LaTeX}$  document:

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

Here is how you would enable the macro in a  $\text{ConTeXt}$  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.

```

1009 \@@_add_plain_tex_option:nnn
1010 { noDefaults }
1011 { boolean }
1012 { 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  $\text{\TeX}$  package documentation using the  $\text{Doc \LaTeX}$  package [10] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```

1013 \seq_gput_right:Nn
1014 \g_@@_plain_tex_options_seq
1015 { stripPercentSigns }
1016 \prop_gput:Nnn
1017 \g_@@_plain_tex_option_types_prop
1018 { stripPercentSigns }
1019 { boolean }
1020 \prop_gput:Nnx
1021 \g_@@_default_plain_tex_options_prop
1022 { stripPercentSigns }
1023 { 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.

```
1024 \cs_new:Nn
1025   \@@_define_option_commands_and_keyvals:
1026   {
1027     \seq_map_inline:Nn
1028       \g_@@_option_layers_seq
1029       {
1030         \seq_map_inline:cn
1031           { g_@@_ ##1 _options_seq }
1032           {
1033             \@@_define_option_command:n
1034               { ####1 }
```

To make it easier to copy-and-paste options from Pandoc [7] 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 [8] also show that `snake_case` is faster to read than `camelCase`.

```
1035         \@@_with_various_cases:nn
1036         { ####1 }
1037         {
1038           \@@_define_option_keyval:nnn
1039             { ##1 }
1040             { ####1 }
1041             { #####1 }
1042         }
1043     }
1044 }
1045 }
1046 \cs_new:Nn
1047   \@@_define_option_command:n
1048   {
```

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

```

1049 \str_if_eq:nnTF
1050 { #1 }
1051 { outputDir }
1052 { \@@_define_option_command_output_dir: }
1053 {

```

Do not override options defined before loading the package.

```

1054 \@@_option_tl_to_csname:nN
1055 { #1 }
1056 \l_tmpa_tl
1057 \cs_if_exist:cF
1058 { \l_tmpa_tl }
1059 {
1060 \@@_get_default_option_value:nN
1061 { #1 }
1062 \l_tmpa_tl
1063 \@@_set_option_value:nV
1064 { #1 }
1065 \l_tmpa_tl
1066 }
1067 }
1068 }
1069 \ExplSyntaxOff
1070 \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 [2, 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.

```

1071 \ExplSyntaxOn
1072 \cs_new:Nn
1073 \@@_define_option_command_output_dir:
1074 {
1075 \cs_if_free:NT
1076 \markdownOptionOutputDir
1077 {
1078 \bool_if:nTF
1079 {
1080 \cs_if_exist_p:N
1081 \luabridge_tl_set:Nn &&

```



```

1082      (
1083      \int_compare_p:nNn
1084      { \g_luabridge_method_int }
1085      =
1086      { \c_luabridge_method_directlua_int } ||
1087      \sys_if_shell_unrestricted_p:
1088      )
1089    }
1090    {

```

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.

```

1091      \group_begin:
1092      \cctab_select:N
1093      \c_str_cctab
1094      \luabridge_tl_set:Nn
1095      \l_tmpa_tl
1096      {
1097        print(
1098        (status.output_directory)
1099        or~os.getenv("TEXMF_OUTPUT_DIRECTORY")
1100        or~"."
1101        )
1102      }
1103      \tl_gset:NV
1104      \markdownOptionOutputDir
1105      \l_tmpa_tl
1106      \group_end:
1107    }
1108    {
1109      \tl_gset:Nn
1110      \markdownOptionOutputDir
1111      { . }
1112    }
1113  }
1114 }
1115 \cs_new:Nn
1116 \@@_set_option_value:nn
1117 {
1118   \@@_define_option:n
1119   { #1 }
1120   \@@_get_option_type:nN
1121   { #1 }
1122   \l_tmpa_tl
1123   \str_if_eq:NNTF
1124   \c_@@_option_type_counter_tl
1125   \l_tmpa_tl

```

```

1126     {
1127         \@@_option_tl_to_csname:nN
1128         { #1 }
1129         \l_tmpa_tl
1130         \int_gset:cn
1131         { \l_tmpa_tl }
1132         { #2 }
1133     }
1134     {
1135         \@@_option_tl_to_csname:nN
1136         { #1 }
1137         \l_tmpa_tl
1138         \cs_set:cpn
1139         { \l_tmpa_tl }
1140         { #2 }
1141     }
1142 }
1143 \cs_generate_variant:Nn
1144 \@@_set_option_value:nn
1145 { nV }
1146 \cs_new:Nn
1147 \@@_define_option:n
1148 {
1149     \@@_option_tl_to_csname:nN
1150     { #1 }
1151     \l_tmpa_tl
1152     \cs_if_free:cT
1153     { \l_tmpa_tl }
1154     {
1155         \@@_get_option_type:nN
1156         { #1 }
1157         \l_tmpb_tl
1158         \str_if_eq:NNT
1159         \c_@@_option_type_counter_tl
1160         \l_tmpb_tl
1161         {
1162             \@@_option_tl_to_csname:nN
1163             { #1 }
1164             \l_tmpa_tl
1165             \int_new:c
1166             { \l_tmpa_tl }
1167         }
1168     }
1169 }
1170 \cs_new:Nn
1171 \@@_define_option_keyval:nnn
1172 {

```

```

1173 \prop_get:cnN
1174 { g_@@_ #1 _option_types_prop }
1175 { #2 }
1176 \l_tmpa_tl
1177 \str_if_eq:VTF
1178 \l_tmpa_tl
1179 \c_@@_option_type_boolean_tl
1180 {
1181 \keys_define:nn
1182 { markdown/options }
1183 {

```

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

```

1184 #3 .code:n = {
1185 \tl_set:Nx
1186 \l_tmpa_tl
1187 {
1188 \str_case:nnF
1189 { ##1 }
1190 {
1191 { yes } { true }
1192 { no } { false }
1193 }
1194 { ##1 }
1195 }
1196 \@@_set_option_value:nV
1197 { #2 }
1198 \l_tmpa_tl
1199 },
1200 #3 .default:n = { true },
1201 }
1202 }
1203 {
1204 \keys_define:nn
1205 { markdown/options }
1206 {
1207 #3 .code:n = {
1208 \@@_set_option_value:nn
1209 { #2 }
1210 { ##1 }
1211 },
1212 }
1213 }

```

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.

```

1214 \str_if_eq:VVT
1215 \l_tmpa_tl
1216 \c_@@_option_type_clist_tl
1217 {
1218 \tl_set:Nn
1219 \l_tmpa_tl
1220 { #3 }
1221 \tl_reverse:N
1222 \l_tmpa_tl
1223 \str_if_eq:enF
1224 {
1225 \tl_head:V
1226 \l_tmpa_tl
1227 }
1228 { s }
1229 {
1230 \msg_error:nnn
1231 { markdown }
1232 { malformed-name-for-clist-option }
1233 { #3 }
1234 }
1235 \tl_set:Nx
1236 \l_tmpa_tl
1237 {
1238 \tl_tail:V
1239 \l_tmpa_tl
1240 }
1241 \tl_reverse:N
1242 \l_tmpa_tl
1243 \tl_put_right:Nn
1244 \l_tmpa_tl
1245 {
1246 .code:n = {
1247 \@@_get_option_value:nN
1248 { #2 }
1249 \l_tmpa_tl
1250 \clist_set:NV
1251 \l_tmpa_clist
1252 { \l_tmpa_tl , { ##1 } }
1253 \@@_set_option_value:nV
1254 { #2 }
1255 \l_tmpa_clist
1256 }
1257 }
1258 \keys_define:nV

```

```

1259         { markdown/options }
1260         \l_tmpa_tl
1261     }
1262 }
1263 \cs_generate_variant:Nn
1264   \clist_set:Nn
1265   { NV }
1266 \cs_generate_variant:Nn
1267   \keys_define:nn
1268   { nV }
1269 \prg_generate_conditional_variant:Nnn
1270   \str_if_eq:nn
1271   { en }
1272   { p, F }
1273 \msg_new:nnn
1274   { markdown }
1275   { malformed-name-for-clist-option }
1276   {
1277       Clist-option-name~#1~does~not~end~with~-s.
1278   }

```

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

```

1279 \str_if_eq:VVT
1280   \c_@@_top_layer_tl
1281   \c_@@_option_layer_plain_tex_tl
1282   {
1283       \@@_define_option_commands_and_keyvals:
1284   }
1285 \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 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 L<sup>A</sup>T<sub>E</sub>X package, which provides similar functionality with its `\usetheme` macro [11, 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  $\langle theme author \rangle / \langle theme purpose \rangle / \langle private naming scheme \rangle$ , where the *private naming scheme* may contain additional forward slashes. For example, a theme by a user `witiko` for the MU theme of the Beamer document class may have the name `witiko/beamer/MU`.

Theme names are munged to allow structure inside theme names without dictating where the themes should be located inside the T<sub>E</sub>X directory structure. For example, loading a theme named `witiko/beamer/MU` would load a T<sub>E</sub>X document package named `markdownthemewitiko_beamer_MU.tex`.

If  $\@ \langle theme version \rangle$  is specified after  $\langle theme name \rangle$ , then the text *theme version* will be available in the macro `\markdownThemeVersion` when the theme is loaded. If  $\@ \langle theme version \rangle$  is not specified, the macro `\markdownThemeVersion` will contain the text `latest` [12].

```

1286 \ExplSyntaxOn
1287 \keys_define:nn
1288   { markdown/options }
1289   {
1290     theme .code:n = {
1291       \@@_set_theme:n
1292       { #1 }
1293     },
1294     import .code:n = {
1295       \tl_set:Nn
1296       \l_tmpa_tl
1297       { #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.

```

1298       \tl_replace_all:NnV
1299       \l_tmpa_tl
1300       { / }
1301       \c_backslash_str
1302       \keys_set:nV
1303       { markdown/options/import }
1304       \l_tmpa_tl
1305     },
1306   }

```

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.

```

1307 \seq_new:N
1308   \g_@@_theme_names_seq
1309 \seq_new:N
1310   \g_@@_theme_versions_seq
1311 \tl_new:N
1312   \g_@@_current_theme_tl
1313 \tl_gset:Nn
1314   \g_@@_current_theme_tl
1315   { }
1316 \seq_gput_right:NV
1317   \g_@@_theme_names_seq
1318   \g_@@_current_theme_tl
1319 \cs_new:Npn
1320   \markdownThemeVersion
1321   { }
1322 \seq_gput_right:NV
1323   \g_@@_theme_versions_seq
1324   \g_@@_current_theme_tl
1325 \cs_new:Nn
1326   \@@_set_theme:n
1327   {

```

First, we validate the theme name.

```

1328   \str_if_in:nnF
1329     { #1 }
1330     { / }
1331     {
1332       \msg_error:nnn
1333         { markdown }
1334         { unqualified-theme-name }
1335         { #1 }
1336     }
1337   \str_if_in:nnT
1338     { #1 }
1339     { _ }
1340     {
1341       \msg_error:nnn
1342         { markdown }
1343         { underscores-in-theme-name }
1344         { #1 }
1345     }

```

Next, we extract the theme version.

```

1346   \str_if_in:nnTF
1347     { #1 }
1348     { @ }
1349     {
1350       \regex_extract_once:nnN

```

```

1351         { (.* ) @ (.* ) }
1352         { #1 }
1353         \l_tmpa_seq
1354     \seq_gpop_left:NN
1355         \l_tmpa_seq
1356         \l_tmpa_tl
1357     \seq_gpop_left:NN
1358         \l_tmpa_seq
1359         \l_tmpa_tl
1360     \tl_gset:NV
1361         \g_@@_current_theme_tl
1362         \l_tmpa_tl
1363     \seq_gpop_left:NN
1364         \l_tmpa_seq
1365         \l_tmpa_tl
1366     \cs_gset:Npe
1367         \markdownThemeVersion
1368         {
1369             \tl_use:N
1370                 \l_tmpa_tl
1371         }
1372     }
1373     {
1374         \tl_gset:Nn
1375             \g_@@_current_theme_tl
1376             { #1 }
1377         \cs_gset:Npn
1378             \markdownThemeVersion
1379             { latest }
1380     }

```

Next, we munge the theme name.

```

1381     \str_set:NV
1382         \l_tmpa_str
1383         \g_@@_current_theme_tl
1384     \str_replace_all:Nnn
1385         \l_tmpa_str
1386         { / }
1387         { _ }

```

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

```

1388     \tl_set:NV
1389         \l_tmpa_tl
1390         \g_@@_current_theme_tl
1391     \tl_put_right:Nn
1392         \g_@@_current_theme_tl
1393         { / }

```



```

1394 \seq_gput_right:NV
1395 \g_@@_theme_names_seq
1396 \g_@@_current_theme_tl
1397 \seq_gput_right:NV
1398 \g_@@_theme_versions_seq
1399 \markdownThemeVersion
1400 \@@_load_theme:VeV
1401 \l_tmpa_tl
1402 { \markdownThemeVersion }
1403 \l_tmpa_str

```

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

```

1404 \seq_gpop_right:NN
1405 \g_@@_theme_names_seq
1406 \l_tmpa_tl
1407 \seq_get_right:NN
1408 \g_@@_theme_names_seq
1409 \l_tmpa_tl
1410 \tl_gset:NV
1411 \g_@@_current_theme_tl
1412 \l_tmpa_tl
1413 \seq_gpop_right:NN
1414 \g_@@_theme_versions_seq
1415 \l_tmpa_tl
1416 \seq_get_right:NN
1417 \g_@@_theme_versions_seq
1418 \l_tmpa_tl
1419 \cs_gset:Npe
1420 \markdownThemeVersion
1421 {
1422   \tl_use:N
1423   \l_tmpa_tl
1424 }
1425 }
1426 \msg_new:nnnn
1427 { markdown }
1428 { unqualified-theme-name }
1429 { Won't~load~theme~with~unqualified~name~#1 }
1430 { Theme~names~must~contain~at~least~one~forward~slash }
1431 \msg_new:nnnn
1432 { markdown }
1433 { underscores-in-theme-name }
1434 { Won't~load~theme~with~an~underscore~in~its~name~#1 }
1435 { Theme~names~must~not~contain~underscores~in~their~names }
1436 \cs_generate_variant:Nn
1437 \tl_replace_all:Nnn

```

```

1438 { NnV }
1439 \cs_generate_variant:Nn
1440 \cs_gset:Npn
1441 { 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.

```

1442 \prop_new:N
1443 \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
end
@enduml
...

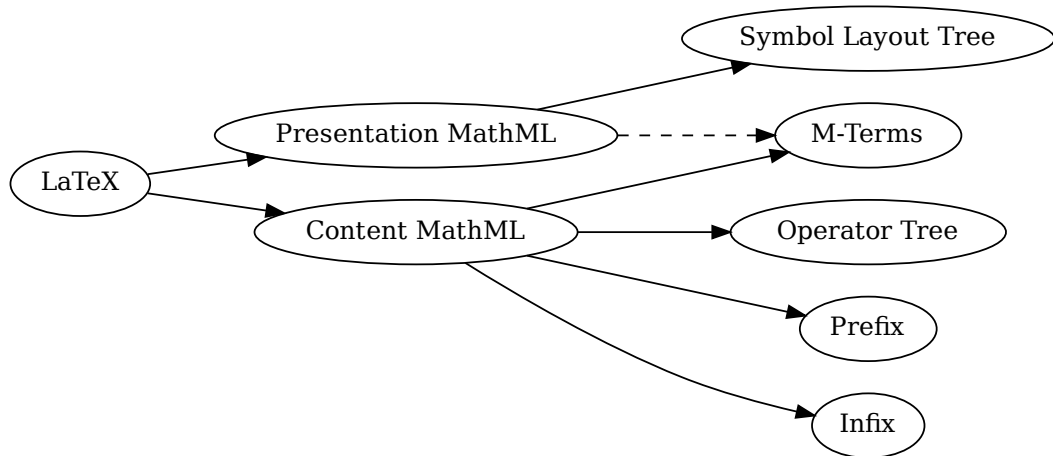
```

```

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.



**Figure 4: An example directed graph**

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.

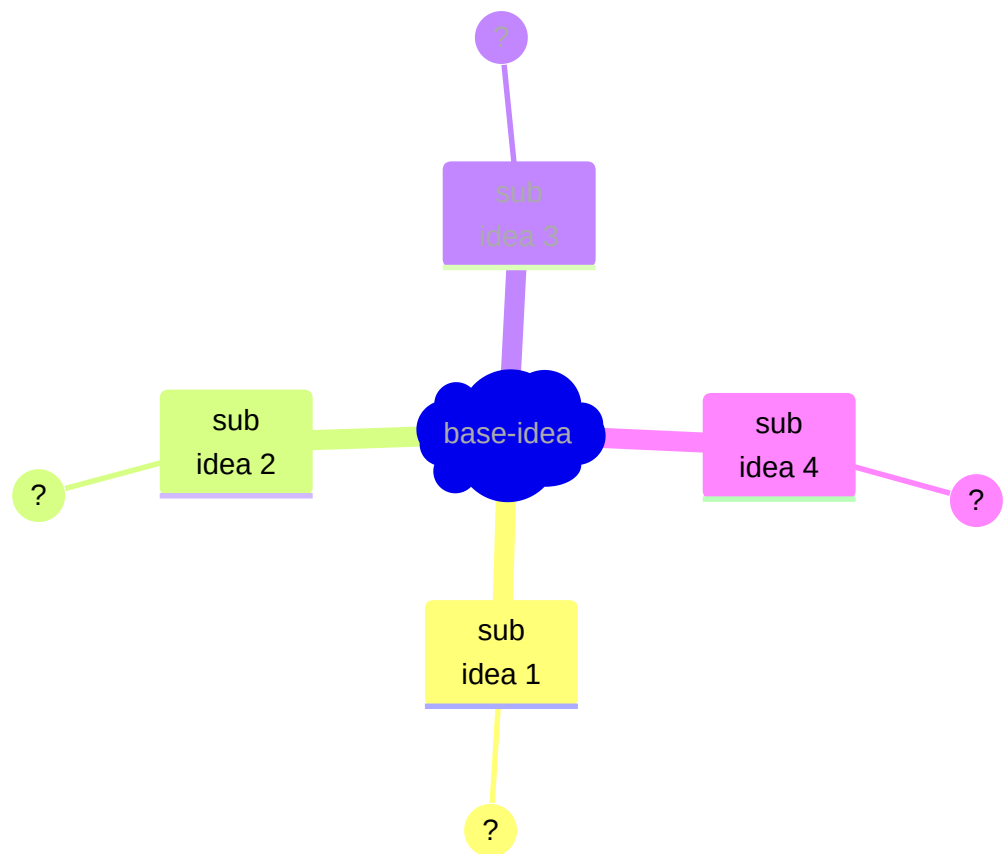
```

\documentclass{article}
\usepackage[import=witiko/graphicx/http]{markdown}
\begin{document}
\begin{markdown}

\end{markdown}
\end{document}

```

Typesetting the above document produces the output shown in Figure 7. The



**Figure 5: An example mindmap**

## Simple Request-Response Flow

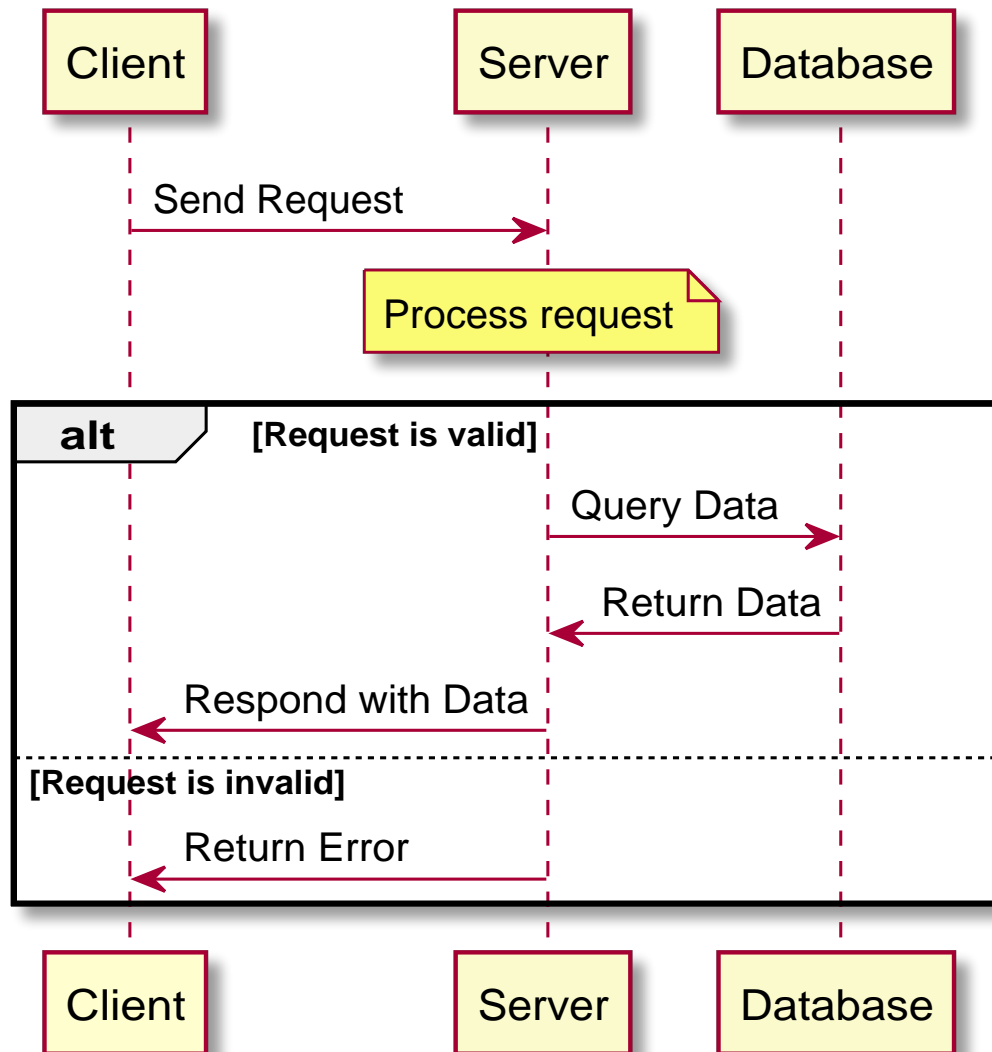


Figure 6: An example UML sequence diagram

```

\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**

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”.

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

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.

```

1444 \prop_new:N
1445   \g_@@_snippets_prop
1446 \cs_new:Nn
1447   \@@_setup_snippet:nn
1448   {
1449     \tl_if_empty:nT
1450       { #1 }
1451       {
1452         \msg_error:nnn
1453           { markdown }
1454           { empty-snippet-name }
1455           { #1 }
1456       }
1457     \tl_set:NV
1458       \l_tmpa_tl
1459       \g_@@_current_theme_tl
1460     \tl_put_right:Nn
1461       \l_tmpa_tl
1462       { #1 }
1463     \@@_if_snippet_exists:nT
1464       { #1 }
1465       {
1466         \msg_warning:nnV
1467           { markdown }
1468           { redefined-snippet }
1469           \l_tmpa_tl
1470       }
1471     \keys_precompile:nnN
1472       { markdown/options }
1473       { #2 }
1474     \l_tmpb_tl
1475     \prop_gput:NVV
1476       \g_@@_snippets_prop
1477       \l_tmpa_tl
1478       \l_tmpb_tl
1479   }
1480 \cs_gset_eq:NN
1481   \markdownSetupSnippet
1482   \@@_setup_snippet:nn

```



```

1483 \msg_new:nnnn
1484   { markdown }
1485   { empty-snippet-name }
1486   { Empty-snippet-name~#1 }
1487   { Pick~a~non-empty~name~for~your~snippet }
1488 \msg_new:nnn
1489   { markdown }
1490   { redefined-snippet }
1491   { Redefined~snippet~#1 }

```

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

```

1492 \tl_new:N
1493   \l_@@_current_snippet_tl
1494 \prg_new_conditional:Nnn
1495   \@@_if_snippet_exists:n
1496   { TF, T }
1497   {
1498     \tl_set:NV
1499       \l_@@_current_snippet_tl
1500       \g_@@_current_theme_tl
1501     \tl_put_right:Nn
1502       \l_@@_current_snippet_tl
1503       { #1 }
1504     \prop_if_in:NVTF
1505       \g_@@_snippets_prop
1506       \l_@@_current_snippet_tl
1507       { \prg_return_true: }
1508       { \prg_return_false: }
1509   }
1510 \cs_gset_eq:NN
1511   \markdownIfSnippetExists
1512   \@@_if_snippet_exists:nTF

```

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

```

1513 \keys_define:nn
1514   { markdown/options }
1515   {
1516     snippet .code:n = {
1517       \tl_set:NV
1518         \l_tmpa_tl
1519         \g_@@_current_theme_tl
1520       \tl_put_right:Nn
1521         \l_tmpa_tl
1522         { #1 }
1523       \@@_if_snippet_exists:nTF
1524         { #1 }
1525     }

```

```

1526         \prop_get:NVN
1527         \g_@@_snippets_prop
1528         \l_tmpa_tl
1529         \l_tmpb_tl
1530         \tl_use:N
1531         \l_tmpb_tl
1532     }
1533     {
1534         \msg_error:nnV
1535         { markdown }
1536         { undefined-snippet }
1537         \l_tmpa_tl
1538     }
1539 }
1540 }
1541 \msg_new:nnn
1542 { markdown }
1543 { undefined-snippet }
1544 { Can't~invoke~undefined~snippet~#1 }
1545 \ExplSyntaxOff

```

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

```

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

```

The following ordered list will be preceded by arabic numerals:

1. wahid
2. aithnayn

```

\end{markdown}
\begin{markdown}[snippet=romanNumerals]

```

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```

\end{markdown}

```

If the `romanNumerals` snippet were defined in the `jdoe/lists` theme, we could im-

port 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,
  },
}
```

```
1546 \ExplSyntaxOn
1547 \tl_new:N
1548   \l_@@_import_current_theme_tl
1549 \keys_define:nn
1550   { markdown/options/import }
1551   {
```

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

```
1552     unknown .default:n = {},
1553     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.

```
1554     \tl_set_eq:NN
```

```

1555     \l_@@_import_current_theme_tl
1556     \l_keys_key_str
1557     \tl_replace_all:NVn
1558     \l_@@_import_current_theme_tl
1559     \c_backslash_str
1560     { / }

```

Here, we import the snippets.

```

1561     \clist_map_inline:nn
1562     { #1 }
1563     {
1564         \regex_extract_once:nnNTF
1565         { ^(.*)\s+as\s+(.*)$ }
1566         { ##1 }
1567         \l_tmpa_seq
1568         {
1569             \seq_pop:NN
1570             \l_tmpa_seq
1571             \l_tmpa_tl
1572             \seq_pop:NN
1573             \l_tmpa_seq
1574             \l_tmpa_tl
1575             \seq_pop:NN
1576             \l_tmpa_seq
1577             \l_tmpb_tl
1578         }
1579         {
1580             \tl_set:Nn
1581             \l_tmpa_tl
1582             { ##1 }
1583             \tl_set:Nn
1584             \l_tmpb_tl
1585             { ##1 }
1586         }
1587         \tl_put_left:Nn
1588         \l_tmpa_tl
1589         { / }
1590         \tl_put_left:NV
1591         \l_tmpa_tl
1592         \l_@@_import_current_theme_tl
1593         \@@_setup_snippet:Vx
1594         \l_tmpb_tl
1595         { snippet = { \l_tmpa_tl } }
1596     }

```

Here, we load the theme.

```

1597     \@@_set_theme:V
1598     \l_@@_import_current_theme_tl

```

```

1599     },
1600   }
1601   \cs_generate_variant:Nn
1602     \tl_replace_all:Nnn
1603     { NVn }
1604   \cs_generate_variant:Nn
1605     \@@_set_theme:n
1606     { V }
1607   \cs_generate_variant:Nn
1608     \@@_setup_snippet:nn
1609     { 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.

```
1610 \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.

```

1611 \prop_new:N \g_@@_renderer_arities_prop
1612 \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  $\langle identifier \rangle$ .

`\markdownRendererAttributeName` 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.

```

1613 \ExplSyntaxOn
1614 \cs_gset_protected:Npn
1615   \markdownRendererAttributeIdentifier
1616   {
1617     \markdownRendererAttributeIdentifierPrototype
1618   }
1619 \seq_gput_right:Nn
1620   \g_@@_renderers_seq
1621   { attributeIdentifier }
1622 \prop_gput:Nnn
1623   \g_@@_renderer_arities_prop
1624   { attributeIdentifier }
1625   { 1 }
1626 \cs_gset_protected:Npn
1627   \markdownRendererAttributeClassName
1628   {
1629     \markdownRendererAttributeClassNamePrototype
1630   }
1631 \seq_gput_right:Nn
1632   \g_@@_renderers_seq
1633   { attributeClassName }
1634 \prop_gput:Nnn
1635   \g_@@_renderer_arities_prop
1636   { attributeClassName }
1637   { 1 }
1638 \cs_gset_protected:Npn
1639   \markdownRendererAttributeKeyValue
1640   {
1641     \markdownRendererAttributeKeyValuePrototype
1642   }
1643 \seq_gput_right:Nn
1644   \g_@@_renderers_seq
1645   { attributeKeyValue }
1646 \prop_gput:Nnn
1647   \g_@@_renderer_arities_prop
1648   { attributeKeyValue }
1649   { 2 }
1650 \ExplSyntaxOff

```

### 2.2.5.2 Block Quote Renderers

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

```

1651 \ExplSyntaxOn
1652 \cs_gset_protected:Npn
1653   \markdownRendererBlockQuoteBegin
1654   {
1655     \markdownRendererBlockQuoteBeginPrototype
1656   }
1657 \seq_gput_right:Nn
1658   \g_@@_renderers_seq
1659   { blockQuoteBegin }
1660 \prop_gput:Nnn
1661   \g_@@_renderer_arities_prop
1662   { blockQuoteBegin }
1663   { 0 }
1664 \ExplSyntaxOff

```

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

```

1665 \ExplSyntaxOn
1666 \cs_gset_protected:Npn
1667   \markdownRendererBlockQuoteEnd
1668   {
1669     \markdownRendererBlockQuoteEndPrototype
1670   }
1671 \seq_gput_right:Nn
1672   \g_@@_renderers_seq
1673   { blockQuoteEnd }
1674 \prop_gput:Nnn
1675   \g_@@_renderer_arities_prop
1676   { blockQuoteEnd }
1677   { 0 }
1678 \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.

```

1679 \ExplSyntaxOn
1680 \cs_gset_protected:Npn
1681   \markdownRendererBracketedSpanAttributeContextBegin
1682   {
1683     \markdownRendererBracketedSpanAttributeContextBeginPrototype
1684   }
1685 \seq_gput_right:Nn

```



```

1686 \g_@@_renderers_seq
1687 { bracketedSpanAttributeContextBegin }
1688 \prop_gput:Nnn
1689 \g_@@_renderer_arities_prop
1690 { bracketedSpanAttributeContextBegin }
1691 { 0 }
1692 \cs_gset_protected:Npn
1693 \markdownRendererBracketedSpanAttributeContextEnd
1694 {
1695   \markdownRendererBracketedSpanAttributeContextEndPrototype
1696 }
1697 \seq_gput_right:Nn
1698 \g_@@_renderers_seq
1699 { bracketedSpanAttributeContextEnd }
1700 \prop_gput:Nnn
1701 \g_@@_renderer_arities_prop
1702 { bracketedSpanAttributeContextEnd }
1703 { 0 }
1704 \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.

```

1705 \ExplSyntaxOn
1706 \cs_gset_protected:Npn
1707 \markdownRendererUlBegin
1708 {
1709   \markdownRendererUlBeginPrototype
1710 }
1711 \seq_gput_right:Nn
1712 \g_@@_renderers_seq
1713 { ulBegin }
1714 \prop_gput:Nnn
1715 \g_@@_renderer_arities_prop
1716 { ulBegin }
1717 { 0 }
1718 \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.

```

1719 \ExplSyntaxOn
1720 \cs_gset_protected:Npn

```

```

1721 \markdownRendererUlBeginTight
1722 {
1723   \markdownRendererUlBeginTightPrototype
1724 }
1725 \seq_gput_right:Nn
1726 \g_@@_renderers_seq
1727 { ulBeginTight }
1728 \prop_gput:Nnn
1729 \g_@@_renderer_arities_prop
1730 { ulBeginTight }
1731 { 0 }
1732 \ExplSyntaxOff

```

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

```

1733 \ExplSyntaxOn
1734 \cs_gset_protected:Npn
1735   \markdownRendererUlItem
1736   {
1737     \markdownRendererUlItemPrototype
1738   }
1739 \seq_gput_right:Nn
1740 \g_@@_renderers_seq
1741 { ulItem }
1742 \prop_gput:Nnn
1743 \g_@@_renderer_arities_prop
1744 { ulItem }
1745 { 0 }
1746 \ExplSyntaxOff

```

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

```

1747 \ExplSyntaxOn
1748 \cs_gset_protected:Npn
1749   \markdownRendererUlItemEnd
1750   {
1751     \markdownRendererUlItemEndPrototype
1752   }
1753 \seq_gput_right:Nn
1754 \g_@@_renderers_seq
1755 { ulItemEnd }
1756 \prop_gput:Nnn
1757 \g_@@_renderer_arities_prop
1758 { ulItemEnd }
1759 { 0 }
1760 \ExplSyntaxOff

```

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

```

1761 \ExplSyntaxOn
1762 \cs_gset_protected:Npn
1763   \markdownRendererUEnd
1764   {
1765     \markdownRendererUEndPrototype
1766   }
1767 \seq_gput_right:Nn
1768   \g_@@_renderers_seq
1769   { ulEnd }
1770 \prop_gput:Nnn
1771   \g_@@_renderer_arities_prop
1772   { ulEnd }
1773   { 0 }
1774 \ExplSyntaxOff

```

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

```

1775 \ExplSyntaxOn
1776 \cs_gset_protected:Npn
1777   \markdownRendererUEndTight
1778   {
1779     \markdownRendererUEndTightPrototype
1780   }
1781 \seq_gput_right:Nn
1782   \g_@@_renderers_seq
1783   { ulEndTight }
1784 \prop_gput:Nnn
1785   \g_@@_renderer_arities_prop
1786   { ulEndTight }
1787   { 0 }
1788 \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.

```

1789 \ExplSyntaxOn
1790 \cs_gset_protected:Npn
1791   \markdownRendererCite
1792   {
1793     \markdownRendererCitePrototype
1794   }
1795 \seq_gput_right:Nn
1796   \g_@@_renderers_seq
1797   { cite }
1798 \prop_gput:Nnn
1799   \g_@@_renderer_arities_prop
1800   { cite }
1801   { 1 }
1802 \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.

```

1803 \ExplSyntaxOn
1804 \cs_gset_protected:Npn
1805   \markdownRendererTextCite
1806   {
1807     \markdownRendererTextCitePrototype
1808   }
1809 \seq_gput_right:Nn
1810   \g_@@_renderers_seq
1811   { textCite }
1812 \prop_gput:Nnn
1813   \g_@@_renderer_arities_prop
1814   { textCite }
1815   { 1 }
1816 \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.

```

1817 \ExplSyntaxOn
1818 \cs_gset_protected:Npn
1819   \markdownRendererInputVerbatim
1820   {
1821     \markdownRendererInputVerbatimPrototype
1822   }
1823 \seq_gput_right:Nn

```

```

1824 \g_@@_renderers_seq
1825 { inputVerbatim }
1826 \prop_gput:Nnn
1827 \g_@@_renderer_arities_prop
1828 { inputVerbatim }
1829 { 1 }
1830 \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.

```

1831 \ExplSyntaxOn
1832 \cs_gset_protected:Npn
1833 \markdownRendererInputFencedCode
1834 {
1835   \markdownRendererInputFencedCodePrototype
1836 }
1837 \seq_gput_right:Nn
1838 \g_@@_renderers_seq
1839 { inputFencedCode }
1840 \prop_gput:Nnn
1841 \g_@@_renderer_arities_prop
1842 { inputFencedCode }
1843 { 3 }
1844 \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.

```

1845 \ExplSyntaxOn
1846 \cs_gset_protected:Npn
1847 \markdownRendererCodeSpan
1848 {
1849   \markdownRendererCodeSpanPrototype
1850 }
1851 \seq_gput_right:Nn
1852 \g_@@_renderers_seq
1853 { codeSpan }
1854 \prop_gput:Nnn
1855 \g_@@_renderer_arities_prop
1856 { codeSpan }
1857 { 1 }
1858 \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.

```
1859 \ExplSyntaxOn
1860 \cs_gset_protected:Npn
1861   \markdownRendererCodeSpanAttributeContextBegin
1862   {
1863     \markdownRendererCodeSpanAttributeContextBeginPrototype
1864   }
1865 \seq_gput_right:Nn
1866   \g_@@_renderers_seq
1867   { codeSpanAttributeContextBegin }
1868 \prop_gput:Nnn
1869   \g_@@_renderer_arities_prop
1870   { codeSpanAttributeContextBegin }
1871   { 0 }
1872 \cs_gset_protected:Npn
1873   \markdownRendererCodeSpanAttributeContextEnd
1874   {
1875     \markdownRendererCodeSpanAttributeContextEndPrototype
1876   }
1877 \seq_gput_right:Nn
1878   \g_@@_renderers_seq
1879   { codeSpanAttributeContextEnd }
1880 \prop_gput:Nnn
1881   \g_@@_renderer_arities_prop
1882   { codeSpanAttributeContextEnd }
1883   { 0 }
1884 \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.

```
1885 \ExplSyntaxOn
1886 \cs_gset_protected:Npn
1887   \markdownRendererContentBlock
1888   {
1889     \markdownRendererContentBlockPrototype
1890   }
1891 \seq_gput_right:Nn
```

```

1892 \g_@@_renderers_seq
1893 { contentBlock }
1894 \prop_gput:Nnn
1895 \g_@@_renderer_arities_prop
1896 { contentBlock }
1897 { 4 }
1898 \ExplSyntaxOff

```

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

```

1899 \ExplSyntaxOn
1900 \cs_gset_protected:Npn
1901 \markdownRendererContentBlockOnlineImage
1902 {
1903 \markdownRendererContentBlockOnlineImagePrototype
1904 }
1905 \seq_gput_right:Nn
1906 \g_@@_renderers_seq
1907 { contentBlockOnlineImage }
1908 \prop_gput:Nnn
1909 \g_@@_renderer_arities_prop
1910 { contentBlockOnlineImage }
1911 { 4 }
1912 \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 T<sub>E</sub>X directory structure. In this file, you will define a mapping between filename extensions and the language names recognized by your favorite syntax highlighter; there may exist other creative uses beside syntax highlighting. The `Languages.json` file provided by Sotkov [6] is a good starting point.

```

1913 \ExplSyntaxOn
1914 \cs_gset_protected:Npn

```

---

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

```

1915 \markdownRendererContentBlockCode
1916 {
1917   \markdownRendererContentBlockCodePrototype
1918 }
1919 \seq_gput_right:Nn
1920 \g_@@_renderers_seq
1921 { contentBlockCode }
1922 \prop_gput:Nnn
1923 \g_@@_renderer_arities_prop
1924 { contentBlockCode }
1925 { 5 }
1926 \ExplSyntaxOff

```

### 2.2.5.10 Definition List Renderers

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

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

```

1927 \ExplSyntaxOn
1928 \cs_gset_protected:Npn
1929 \markdownRendererDlBegin
1930 {
1931   \markdownRendererDlBeginPrototype
1932 }
1933 \seq_gput_right:Nn
1934 \g_@@_renderers_seq
1935 { dlBegin }
1936 \prop_gput:Nnn
1937 \g_@@_renderer_arities_prop
1938 { dlBegin }
1939 { 0 }
1940 \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.

```

1941 \ExplSyntaxOn
1942 \cs_gset_protected:Npn
1943 \markdownRendererDlBeginTight
1944 {
1945   \markdownRendererDlBeginTightPrototype
1946 }
1947 \seq_gput_right:Nn

```



```

1948 \g_@@_renderers_seq
1949 { dlBeginTight }
1950 \prop_gput:Nnn
1951 \g_@@_renderer_arities_prop
1952 { dlBeginTight }
1953 { 0 }
1954 \ExplSyntaxOff

```

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

```

1955 \ExplSyntaxOn
1956 \cs_gset_protected:Npn
1957 \markdownRendererDlItem
1958 {
1959   \markdownRendererDlItemPrototype
1960 }
1961 \seq_gput_right:Nn
1962 \g_@@_renderers_seq
1963 { dlItem }
1964 \prop_gput:Nnn
1965 \g_@@_renderer_arities_prop
1966 { dlItem }
1967 { 1 }
1968 \ExplSyntaxOff

```

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

```

1969 \ExplSyntaxOn
1970 \cs_gset_protected:Npn
1971 \markdownRendererDlItemEnd
1972 {
1973   \markdownRendererDlItemEndPrototype
1974 }
1975 \seq_gput_right:Nn
1976 \g_@@_renderers_seq
1977 { dlItemEnd }
1978 \prop_gput:Nnn
1979 \g_@@_renderer_arities_prop
1980 { dlItemEnd }
1981 { 0 }
1982 \ExplSyntaxOff

```

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

```

1983 \ExplSyntaxOn
1984 \cs_gset_protected:Npn
1985 \markdownRendererDlDefinitionBegin

```

```

1986 {
1987   \markdownRendererDlDefinitionBeginPrototype
1988 }
1989 \seq_gput_right:Nn
1990 \g_@@_renderers_seq
1991 { dlDefinitionBegin }
1992 \prop_gput:Nnn
1993 \g_@@_renderer_arities_prop
1994 { dlDefinitionBegin }
1995 { 0 }
1996 \ExplSyntaxOff

```

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

```

1997 \ExplSyntaxOn
1998 \cs_gset_protected:Npn
1999   \markdownRendererDlDefinitionEnd
2000 {
2001   \markdownRendererDlDefinitionEndPrototype
2002 }
2003 \seq_gput_right:Nn
2004 \g_@@_renderers_seq
2005 { dlDefinitionEnd }
2006 \prop_gput:Nnn
2007 \g_@@_renderer_arities_prop
2008 { dlDefinitionEnd }
2009 { 0 }
2010 \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.

```

2011 \ExplSyntaxOn
2012 \cs_gset_protected:Npn
2013   \markdownRendererDlEnd
2014 {
2015   \markdownRendererDlEndPrototype
2016 }
2017 \seq_gput_right:Nn
2018 \g_@@_renderers_seq
2019 { dlEnd }
2020 \prop_gput:Nnn
2021 \g_@@_renderer_arities_prop
2022 { dlEnd }
2023 { 0 }
2024 \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.

```

2025 \ExplSyntaxOn
2026 \cs_gset_protected:Npn
2027   \markdownRendererDlEndTight
2028   {
2029     \markdownRendererDlEndTightPrototype
2030   }
2031 \seq_gput_right:Nn
2032   \g_@@_renderers_seq
2033   { dlEndTight }
2034 \prop_gput:Nnn
2035   \g_@@_renderer_arities_prop
2036   { dlEndTight }
2037   { 0 }
2038 \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.

```

2039 \ExplSyntaxOn
2040 \cs_gset_protected:Npn
2041   \markdownRendererEllipsis
2042   {
2043     \markdownRendererEllipsisPrototype
2044   }
2045 \seq_gput_right:Nn
2046   \g_@@_renderers_seq
2047   { ellipsis }
2048 \prop_gput:Nnn
2049   \g_@@_renderer_arities_prop
2050   { ellipsis }
2051   { 0 }
2052 \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.

```

2053 \ExplSyntaxOn
2054 \cs_gset_protected:Npn

```

```

2055 \markdownRendererEmphasis
2056 {
2057   \markdownRendererEmphasisPrototype
2058 }
2059 \seq_gput_right:Nn
2060 \g_@@_renderers_seq
2061 { emphasis }
2062 \prop_gput:Nnn
2063 \g_@@_renderer_arities_prop
2064 { emphasis }
2065 { 1 }
2066 \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.

```

2067 \ExplSyntaxOn
2068 \cs_gset_protected:Npn
2069   \markdownRendererStrongEmphasis
2070   {
2071     \markdownRendererStrongEmphasisPrototype
2072   }
2073 \seq_gput_right:Nn
2074 \g_@@_renderers_seq
2075 { strongEmphasis }
2076 \prop_gput:Nnn
2077 \g_@@_renderer_arities_prop
2078 { strongEmphasis }
2079 { 1 }
2080 \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.

```

2081 \ExplSyntaxOn
2082 \cs_gset_protected:Npn
2083   \markdownRendererFencedCodeAttributeContextBegin
2084   {
2085     \markdownRendererFencedCodeAttributeContextBeginPrototype
2086   }
2087 \seq_gput_right:Nn
2088 \g_@@_renderers_seq

```

```

2089 { fencedCodeAttributeContextBegin }
2090 \prop_gput:Nnn
2091 \g_@@_renderer_arities_prop
2092 { fencedCodeAttributeContextBegin }
2093 { 0 }
2094 \cs_gset_protected:Npn
2095 \markdownRendererFencedCodeAttributeContextEnd
2096 {
2097   \markdownRendererFencedCodeAttributeContextEndPrototype
2098 }
2099 \seq_gput_right:Nn
2100 \g_@@_renderers_seq
2101 { fencedCodeAttributeContextEnd }
2102 \prop_gput:Nnn
2103 \g_@@_renderer_arities_prop
2104 { fencedCodeAttributeContextEnd }
2105 { 0 }
2106 \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.

```

2107 \ExplSyntaxOn
2108 \cs_gset_protected:Npn
2109 \markdownRendererFencedDivAttributeContextBegin
2110 {
2111   \markdownRendererFencedDivAttributeContextBeginPrototype
2112 }
2113 \seq_gput_right:Nn
2114 \g_@@_renderers_seq
2115 { fencedDivAttributeContextBegin }
2116 \prop_gput:Nnn
2117 \g_@@_renderer_arities_prop
2118 { fencedDivAttributeContextBegin }
2119 { 0 }
2120 \cs_gset_protected:Npn
2121 \markdownRendererFencedDivAttributeContextEnd
2122 {
2123   \markdownRendererFencedDivAttributeContextEndPrototype
2124 }
2125 \seq_gput_right:Nn
2126 \g_@@_renderers_seq
2127 { fencedDivAttributeContextEnd }
2128 \prop_gput:Nnn

```

```

2129 \g_@@_renderer_arities_prop
2130 { fencedDivAttributeContextEnd }
2131 { 0 }
2132 \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.

```

2133 \ExplSyntaxOn
2134 \cs_gset_protected:Npn
2135 \markdownRendererHeaderAttributeContextBegin
2136 {
2137   \markdownRendererHeaderAttributeContextBeginPrototype
2138 }
2139 \seq_gput_right:Nn
2140 \g_@@_renderers_seq
2141 { headerAttributeContextBegin }
2142 \prop_gput:Nnn
2143 \g_@@_renderer_arities_prop
2144 { headerAttributeContextBegin }
2145 { 0 }
2146 \cs_gset_protected:Npn
2147 \markdownRendererHeaderAttributeContextEnd
2148 {
2149   \markdownRendererHeaderAttributeContextEndPrototype
2150 }
2151 \seq_gput_right:Nn
2152 \g_@@_renderers_seq
2153 { headerAttributeContextEnd }
2154 \prop_gput:Nnn
2155 \g_@@_renderer_arities_prop
2156 { headerAttributeContextEnd }
2157 { 0 }
2158 \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.

```

2159 \ExplSyntaxOn
2160 \cs_gset_protected:Npn
2161 \markdownRendererHeadingOne

```

```

2162 {
2163   \markdownRendererHeadingOnePrototype
2164 }
2165 \seq_gput_right:Nn
2166 \g_@@_renderers_seq
2167 { headingOne }
2168 \prop_gput:Nnn
2169 \g_@@_renderer_arities_prop
2170 { headingOne }
2171 { 1 }
2172 \ExplSyntaxOff

```

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

```

2173 \ExplSyntaxOn
2174 \cs_gset_protected:Npn
2175   \markdownRendererHeadingTwo
2176   {
2177     \markdownRendererHeadingTwoPrototype
2178   }
2179 \seq_gput_right:Nn
2180 \g_@@_renderers_seq
2181 { headingTwo }
2182 \prop_gput:Nnn
2183 \g_@@_renderer_arities_prop
2184 { headingTwo }
2185 { 1 }
2186 \ExplSyntaxOff

```

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

```

2187 \ExplSyntaxOn
2188 \cs_gset_protected:Npn
2189   \markdownRendererHeadingThree
2190   {
2191     \markdownRendererHeadingThreePrototype
2192   }
2193 \seq_gput_right:Nn
2194 \g_@@_renderers_seq
2195 { headingThree }
2196 \prop_gput:Nnn
2197 \g_@@_renderer_arities_prop
2198 { headingThree }
2199 { 1 }
2200 \ExplSyntaxOff

```

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

```

2201 \ExplSyntaxOn
2202 \cs_gset_protected:Npn
2203   \markdownRendererHeadingFour
2204   {
2205     \markdownRendererHeadingFourPrototype
2206   }
2207 \seq_gput_right:Nn
2208   \g_@@_renderers_seq
2209   { headingFour }
2210 \prop_gput:Nnn
2211   \g_@@_renderer_arities_prop
2212   { headingFour }
2213   { 1 }
2214 \ExplSyntaxOff

```

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

```

2215 \ExplSyntaxOn
2216 \cs_gset_protected:Npn
2217   \markdownRendererHeadingFive
2218   {
2219     \markdownRendererHeadingFivePrototype
2220   }
2221 \seq_gput_right:Nn
2222   \g_@@_renderers_seq
2223   { headingFive }
2224 \prop_gput:Nnn
2225   \g_@@_renderer_arities_prop
2226   { headingFive }
2227   { 1 }
2228 \ExplSyntaxOff

```

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

```

2229 \ExplSyntaxOn
2230 \cs_gset_protected:Npn
2231   \markdownRendererHeadingSix
2232   {
2233     \markdownRendererHeadingSixPrototype
2234   }
2235 \seq_gput_right:Nn
2236   \g_@@_renderers_seq
2237   { headingSix }
2238 \prop_gput:Nnn

```



```

2239 \g_@@_renderer_arities_prop
2240 { headingSix }
2241 { 1 }
2242 \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.

```

2243 \ExplSyntaxOn
2244 \cs_gset_protected:Npn
2245 \markdownRendererInlineHtmlComment
2246 {
2247 \markdownRendererInlineHtmlCommentPrototype
2248 }
2249 \seq_gput_right:Nn
2250 \g_@@_renderers_seq
2251 { inlineHtmlComment }
2252 \prop_gput:Nnn
2253 \g_@@_renderer_arities_prop
2254 { inlineHtmlComment }
2255 { 1 }
2256 \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.

```

2257 \ExplSyntaxOn
2258 \cs_gset_protected:Npn
2259 \markdownRendererInlineHtmlTag
2260 {
2261 \markdownRendererInlineHtmlTagPrototype
2262 }
2263 \seq_gput_right:Nn
2264 \g_@@_renderers_seq
2265 { inlineHtmlTag }
2266 \prop_gput:Nnn

```

```

2267 \g_@@_renderer_arities_prop
2268 { inlineHtmlTag }
2269 { 1 }
2270 \cs_gset_protected:Npn
2271 \markdownRendererInputBlockHtmlElement
2272 {
2273   \markdownRendererInputBlockHtmlElementPrototype
2274 }
2275 \seq_gput_right:Nn
2276 \g_@@_renderers_seq
2277 { inputBlockHtmlElement }
2278 \prop_gput:Nnn
2279 \g_@@_renderer_arities_prop
2280 { inputBlockHtmlElement }
2281 { 1 }
2282 \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.

```

2283 \ExplSyntaxOn
2284 \cs_gset_protected:Npn
2285 \markdownRendererImage
2286 {
2287   \markdownRendererImagePrototype
2288 }
2289 \seq_gput_right:Nn
2290 \g_@@_renderers_seq
2291 { image }
2292 \prop_gput:Nnn
2293 \g_@@_renderer_arities_prop
2294 { image }
2295 { 4 }
2296 \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.

```

2297 \ExplSyntaxOn
2298 \cs_gset_protected:Npn

```

```

2299 \markdownRendererImageAttributeContextBegin
2300 {
2301     \markdownRendererImageAttributeContextBeginPrototype
2302 }
2303 \seq_gput_right:Nn
2304 \g_@@_renderers_seq
2305 { imageAttributeContextBegin }
2306 \prop_gput:Nnn
2307 \g_@@_renderer_arities_prop
2308 { imageAttributeContextBegin }
2309 { 0 }
2310 \cs_gset_protected:Npn
2311 \markdownRendererImageAttributeContextEnd
2312 {
2313     \markdownRendererImageAttributeContextEndPrototype
2314 }
2315 \seq_gput_right:Nn
2316 \g_@@_renderers_seq
2317 { imageAttributeContextEnd }
2318 \prop_gput:Nnn
2319 \g_@@_renderer_arities_prop
2320 { imageAttributeContextEnd }
2321 { 0 }
2322 \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.

```

2323 \ExplSyntaxOn
2324 \cs_gset_protected:Npn
2325 \markdownRendererInterblockSeparator
2326 {
2327     \markdownRendererInterblockSeparatorPrototype
2328 }
2329 \seq_gput_right:Nn
2330 \g_@@_renderers_seq
2331 { interblockSeparator }
2332 \prop_gput:Nnn
2333 \g_@@_renderer_arities_prop
2334 { interblockSeparator }
2335 { 0 }
2336 \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.

```

2337 \ExplSyntaxOn
2338 \cs_gset_protected:Npn
2339   \markdownRendererParagraphSeparator
2340   {
2341     \markdownRendererParagraphSeparatorPrototype
2342   }
2343 \seq_gput_right:Nn
2344   \g_@@_renderers_seq
2345   { paragraphSeparator }
2346 \prop_gput:Nnn
2347   \g_@@_renderer_arities_prop
2348   { paragraphSeparator }
2349   { 0 }
2350 \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.

```

2351 \ExplSyntaxOn
2352 \cs_gset_protected:Npn
2353   \markdownRendererLineBlockBegin
2354   {
2355     \markdownRendererLineBlockBeginPrototype
2356   }
2357 \seq_gput_right:Nn
2358   \g_@@_renderers_seq
2359   { lineBlockBegin }
2360 \prop_gput:Nnn
2361   \g_@@_renderer_arities_prop
2362   { lineBlockBegin }
2363   { 0 }
2364 \cs_gset_protected:Npn
2365   \markdownRendererLineBlockEnd
2366   {
2367     \markdownRendererLineBlockEndPrototype
2368   }
2369 \seq_gput_right:Nn
2370   \g_@@_renderers_seq
2371   { lineBlockEnd }

```

```

2372 \prop_gput:Nnn
2373   \g_@@_renderer_arities_prop
2374   { lineBlockEnd }
2375   { 0 }
2376 \ExplSyntaxOff

```

### 2.2.5.23 Line Break Renderers

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

```

2377 \ExplSyntaxOn
2378 \cs_gset_protected:Npn
2379   \markdownRendererSoftLineBreak
2380   {
2381     \markdownRendererSoftLineBreakPrototype
2382   }
2383 \seq_gput_right:Nn
2384   \g_@@_renderers_seq
2385   { softLineBreak }
2386 \prop_gput:Nnn
2387   \g_@@_renderer_arities_prop
2388   { softLineBreak }
2389   { 0 }
2390 \ExplSyntaxOff

```

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

```

2391 \ExplSyntaxOn
2392 \cs_gset_protected:Npn
2393   \markdownRendererHardLineBreak
2394   {
2395     \markdownRendererHardLineBreakPrototype
2396   }
2397 \seq_gput_right:Nn
2398   \g_@@_renderers_seq
2399   { hardLineBreak }
2400 \prop_gput:Nnn
2401   \g_@@_renderer_arities_prop
2402   { hardLineBreak }
2403   { 0 }
2404 \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.

```

2405 \ExplSyntaxOn
2406 \cs_gset_protected:Npn
2407   \markdownRendererLink
2408   {
2409     \markdownRendererLinkPrototype
2410   }
2411 \seq_gput_right:Nn
2412   \g_@@_renderers_seq
2413   { link }
2414 \prop_gput:Nnn
2415   \g_@@_renderer_arities_prop
2416   { link }
2417   { 4 }
2418 \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.

```

2419 \ExplSyntaxOn
2420 \cs_gset_protected:Npn
2421   \markdownRendererLinkAttributeContextBegin
2422   {
2423     \markdownRendererLinkAttributeContextBeginPrototype
2424   }
2425 \seq_gput_right:Nn
2426   \g_@@_renderers_seq
2427   { linkAttributeContextBegin }
2428 \prop_gput:Nnn
2429   \g_@@_renderer_arities_prop
2430   { linkAttributeContextBegin }
2431   { 0 }
2432 \cs_gset_protected:Npn
2433   \markdownRendererLinkAttributeContextEnd
2434   {
2435     \markdownRendererLinkAttributeContextEndPrototype
2436   }
2437 \seq_gput_right:Nn
2438   \g_@@_renderers_seq
2439   { linkAttributeContextEnd }
2440 \prop_gput:Nnn
2441   \g_@@_renderer_arities_prop
2442   { linkAttributeContextEnd }
2443   { 0 }

```

```
2444 \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.

```
2445 \ExplSyntaxOn
2446 \cs_gset_protected:Npn
2447   \markdownRendererMark
2448   {
2449     \markdownRendererMarkPrototype
2450   }
2451 \seq_gput_right:Nn
2452   \g_@@_renderers_seq
2453   { mark }
2454 \prop_gput:Nnn
2455   \g_@@_renderer_arities_prop
2456   { mark }
2457   { 1 }
2458 \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.

```
2459 \ExplSyntaxOn
2460 \cs_gset_protected:Npn
2461   \markdownRendererDocumentBegin
2462   {
2463     \markdownRendererDocumentBeginPrototype
2464   }
2465 \seq_gput_right:Nn
2466   \g_@@_renderers_seq
2467   { documentBegin }
2468 \prop_gput:Nnn
2469   \g_@@_renderer_arities_prop
2470   { documentBegin }
2471   { 0 }
2472 \cs_gset_protected:Npn
2473   \markdownRendererDocumentEnd
```

```

2474 {
2475   \markdownRendererDocumentEndPrototype
2476 }
2477 \seq_gput_right:Nn
2478 \g_@@_renderers_seq
2479 { documentEnd }
2480 \prop_gput:Nnn
2481 \g_@@_renderer_arities_prop
2482 { documentEnd }
2483 { 0 }
2484 \ExplSyntaxOff

```

### 2.2.5.28 Non-Breaking Space Renderer

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

```

2485 \ExplSyntaxOn
2486 \cs_gset_protected:Npn
2487   \markdownRendererNbsp
2488   {
2489     \markdownRendererNbspPrototype
2490   }
2491 \seq_gput_right:Nn
2492 \g_@@_renderers_seq
2493 { nbsp }
2494 \prop_gput:Nnn
2495 \g_@@_renderer_arities_prop
2496 { nbsp }
2497 { 0 }
2498 \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.

```

2499 \def\markdownRendererNote{%
2500   \markdownRendererNotePrototype}%
2501 \ExplSyntaxOn
2502 \seq_gput_right:Nn
2503 \g_@@_renderers_seq
2504 { note }
2505 \prop_gput:Nnn
2506 \g_@@_renderer_arities_prop
2507 { note }
2508 { 1 }
2509 \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.

```
2510 \ExplSyntaxOn
2511 \cs_gset_protected:Npn
2512   \markdownRendererOlBegin
2513   {
2514     \markdownRendererOlBeginPrototype
2515   }
2516 \seq_gput_right:Nn
2517   \g_@@_renderers_seq
2518   { olBegin }
2519 \prop_gput:Nnn
2520   \g_@@_renderer_arities_prop
2521   { olBegin }
2522   { 0 }
2523 \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.

```
2524 \ExplSyntaxOn
2525 \cs_gset_protected:Npn
2526   \markdownRendererOlBeginTight
2527   {
2528     \markdownRendererOlBeginTightPrototype
2529   }
2530 \seq_gput_right:Nn
2531   \g_@@_renderers_seq
2532   { olBeginTight }
2533 \prop_gput:Nnn
2534   \g_@@_renderer_arities_prop
2535   { olBeginTight }
2536   { 0 }
2537 \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`).

```

2538 \ExplSyntaxOn
2539 \cs_gset_protected:Npn
2540   \markdownRendererFancyOlBegin
2541   {
2542     \markdownRendererFancyOlBeginPrototype
2543   }
2544 \seq_gput_right:Nn
2545   \g_@@_renderers_seq
2546   { fancyOlBegin }
2547 \prop_gput:Nnn
2548   \g_@@_renderer_arities_prop
2549   { fancyOlBegin }
2550   { 2 }
2551 \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.

```

2552 \ExplSyntaxOn
2553 \cs_gset_protected:Npn
2554   \markdownRendererFancyOlBeginTight
2555   {
2556     \markdownRendererFancyOlBeginTightPrototype
2557   }
2558 \seq_gput_right:Nn
2559   \g_@@_renderers_seq
2560   { fancyOlBeginTight }
2561 \prop_gput:Nnn
2562   \g_@@_renderer_arities_prop
2563   { fancyOlBeginTight }
2564   { 2 }
2565 \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.

```

2566 \ExplSyntaxOn
2567 \cs_gset_protected:Npn
2568   \markdownRendererOlItem
2569   {
2570     \markdownRendererOlItemPrototype
2571   }
2572 \seq_gput_right:Nn

```

```

2573 \g_@@_renderers_seq
2574 { olItem }
2575 \prop_gput:Nnn
2576 \g_@@_renderer_arities_prop
2577 { olItem }
2578 { 0 }
2579 \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.

```

2580 \ExplSyntaxOn
2581 \cs_gset_protected:Npn
2582 \markdownRendererOlItemEnd
2583 {
2584   \markdownRendererOlItemEndPrototype
2585 }
2586 \seq_gput_right:Nn
2587 \g_@@_renderers_seq
2588 { olItemEnd }
2589 \prop_gput:Nnn
2590 \g_@@_renderer_arities_prop
2591 { olItemEnd }
2592 { 0 }
2593 \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.

```

2594 \ExplSyntaxOn
2595 \cs_gset_protected:Npn
2596 \markdownRendererOlItemWithNumber
2597 {
2598   \markdownRendererOlItemWithNumberPrototype
2599 }
2600 \seq_gput_right:Nn
2601 \g_@@_renderers_seq
2602 { olItemWithNumber }
2603 \prop_gput:Nnn
2604 \g_@@_renderer_arities_prop
2605 { olItemWithNumber }
2606 { 1 }
2607 \ExplSyntaxOff

```

The `\markdownRendererFancyO1Item` 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.

```

2608 \ExplSyntaxOn
2609 \cs_gset_protected:Npn
2610   \markdownRendererFancyO1Item
2611   {
2612     \markdownRendererFancyO1ItemPrototype
2613   }
2614 \seq_gput_right:Nn
2615   \g_@@_renderers_seq
2616   { fancyO1Item }
2617 \prop_gput:Nnn
2618   \g_@@_renderer_arities_prop
2619   { fancyO1Item }
2620   { 0 }
2621 \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.

```

2622 \ExplSyntaxOn
2623 \cs_gset_protected:Npn
2624   \markdownRendererFancyO1ItemEnd
2625   {
2626     \markdownRendererFancyO1ItemEndPrototype
2627   }
2628 \seq_gput_right:Nn
2629   \g_@@_renderers_seq
2630   { fancyO1ItemEnd }
2631 \prop_gput:Nnn
2632   \g_@@_renderer_arities_prop
2633   { fancyO1ItemEnd }
2634   { 0 }
2635 \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.

```

2636 \ExplSyntaxOn
2637 \cs_gset_protected:Npn
2638   \markdownRendererFancyO1ItemWithNumber
2639   {
2640     \markdownRendererFancyO1ItemWithNumberPrototype
2641   }

```

```

2642 \seq_gput_right:Nn
2643   \g_@@_renderers_seq
2644   { fancyO1ItemWithNumber }
2645 \prop_gput:Nnn
2646   \g_@@_renderer_arities_prop
2647   { fancyO1ItemWithNumber }
2648   { 1 }
2649 \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.

```

2650 \ExplSyntaxOn
2651 \cs_gset_protected:Npn
2652   \markdownRendererO1End
2653   {
2654     \markdownRendererO1EndPrototype
2655   }
2656 \seq_gput_right:Nn
2657   \g_@@_renderers_seq
2658   { olEnd }
2659 \prop_gput:Nnn
2660   \g_@@_renderer_arities_prop
2661   { olEnd }
2662   { 0 }
2663 \ExplSyntaxOff

```

The `\markdownRendererO1EndTight` macro represents the end of an ordered list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `tightLists` option is enabled and the `fancyLists` option is disabled. The macro receives no arguments.

```

2664 \ExplSyntaxOn
2665 \cs_gset_protected:Npn
2666   \markdownRendererO1EndTight
2667   {
2668     \markdownRendererO1EndTightPrototype
2669   }
2670 \seq_gput_right:Nn
2671   \g_@@_renderers_seq
2672   { olEndTight }
2673 \prop_gput:Nnn
2674   \g_@@_renderer_arities_prop
2675   { olEndTight }
2676   { 0 }
2677 \ExplSyntaxOff

```

The `\markdownRendererFancy01End` 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.

```

2678 \ExplSyntaxOn
2679 \cs_gset_protected:Npn
2680   \markdownRendererFancy01End
2681   {
2682     \markdownRendererFancy01EndPrototype
2683   }
2684 \seq_gput_right:Nn
2685   \g_@@_renderers_seq
2686   { fancy01End }
2687 \prop_gput:Nnn
2688   \g_@@_renderer_arities_prop
2689   { fancy01End }
2690   { 0 }
2691 \ExplSyntaxOff

```

The `\markdownRendererFancy01EndTight` 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.

```

2692 \ExplSyntaxOn
2693 \cs_gset_protected:Npn
2694   \markdownRendererFancy01EndTight
2695   {
2696     \markdownRendererFancy01EndTightPrototype
2697   }
2698 \seq_gput_right:Nn
2699   \g_@@_renderers_seq
2700   { fancy01EndTight }
2701 \prop_gput:Nnn
2702   \g_@@_renderer_arities_prop
2703   { fancy01EndTight }
2704   { 0 }
2705 \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.

```

2706 \ExplSyntaxOn

```

```

2707 \cs_gset_protected:Npn
2708   \markdownRendererInputRawInline
2709   {
2710     \markdownRendererInputRawInlinePrototype
2711   }
2712 \seq_gput_right:Nn
2713   \g_@@_renderers_seq
2714   { inputRawInline }
2715 \prop_gput:Nnn
2716   \g_@@_renderer_arities_prop
2717   { inputRawInline }
2718   { 2 }
2719 \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.

```

2720 \ExplSyntaxOn
2721 \cs_gset_protected:Npn
2722   \markdownRendererInputRawBlock
2723   {
2724     \markdownRendererInputRawBlockPrototype
2725   }
2726 \seq_gput_right:Nn
2727   \g_@@_renderers_seq
2728   { inputRawBlock }
2729 \prop_gput:Nnn
2730   \g_@@_renderer_arities_prop
2731   { inputRawBlock }
2732   { 2 }
2733 \ExplSyntaxOff

```

### 2.2.5.32 Section Renderers

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

```

2734 \ExplSyntaxOn
2735 \cs_gset_protected:Npn
2736   \markdownRendererSectionBegin
2737   {
2738     \markdownRendererSectionBeginPrototype
2739   }
2740 \seq_gput_right:Nn
2741   \g_@@_renderers_seq
2742   { sectionBegin }
2743 \prop_gput:Nnn

```

```

2744 \g_@@_renderer_arities_prop
2745 { sectionBegin }
2746 { 0 }
2747 \cs_gset_protected:Npn
2748 \markdownRendererSectionEnd
2749 {
2750   \markdownRendererSectionEndPrototype
2751 }
2752 \seq_gput_right:Nn
2753 \g_@@_renderers_seq
2754 { sectionEnd }
2755 \prop_gput:Nnn
2756 \g_@@_renderer_arities_prop
2757 { sectionEnd }
2758 { 0 }
2759 \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.

```

2760 \ExplSyntaxOn
2761 \cs_gset_protected:Npn
2762 \markdownRendererReplacementCharacter
2763 {
2764   \markdownRendererReplacementCharacterPrototype
2765 }
2766 \seq_gput_right:Nn
2767 \g_@@_renderers_seq
2768 { replacementCharacter }
2769 \prop_gput:Nnn
2770 \g_@@_renderer_arities_prop
2771 { replacementCharacter }
2772 { 0 }
2773 \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`.

```

2774 \ExplSyntaxOn
2775 \cs_gset_protected:Npn
2776 \markdownRendererLeftBrace
2777 {
2778   \markdownRendererLeftBracePrototype
2779 }

```



```

2780 \seq_gput_right:Nn
2781   \g_@@_renderers_seq
2782   { leftBrace }
2783 \prop_gput:Nnn
2784   \g_@@_renderer_arities_prop
2785   { leftBrace }
2786   { 0 }
2787 \cs_gset_protected:Npn
2788   \markdownRendererRightBrace
2789   {
2790     \markdownRendererRightBracePrototype
2791   }
2792 \seq_gput_right:Nn
2793   \g_@@_renderers_seq
2794   { rightBrace }
2795 \prop_gput:Nnn
2796   \g_@@_renderer_arities_prop
2797   { rightBrace }
2798   { 0 }
2799 \cs_gset_protected:Npn
2800   \markdownRendererDollarSign
2801   {
2802     \markdownRendererDollarSignPrototype
2803   }
2804 \seq_gput_right:Nn
2805   \g_@@_renderers_seq
2806   { dollarSign }
2807 \prop_gput:Nnn
2808   \g_@@_renderer_arities_prop
2809   { dollarSign }
2810   { 0 }
2811 \cs_gset_protected:Npn
2812   \markdownRendererPercentSign
2813   {
2814     \markdownRendererPercentSignPrototype
2815   }
2816 \seq_gput_right:Nn
2817   \g_@@_renderers_seq
2818   { percentSign }
2819 \prop_gput:Nnn
2820   \g_@@_renderer_arities_prop
2821   { percentSign }
2822   { 0 }
2823 \cs_gset_protected:Npn
2824   \markdownRendererAmpersand
2825   {
2826     \markdownRendererAmpersandPrototype

```

```

2827     }
2828 \seq_gput_right:Nn
2829   \g_@@_renderers_seq
2830   { ampersand }
2831 \prop_gput:Nnn
2832   \g_@@_renderer_arities_prop
2833   { ampersand }
2834   { 0 }
2835 \cs_gset_protected:Npn
2836   \markdownRendererUnderscore
2837   {
2838     \markdownRendererUnderscorePrototype
2839   }
2840 \seq_gput_right:Nn
2841   \g_@@_renderers_seq
2842   { underscore }
2843 \prop_gput:Nnn
2844   \g_@@_renderer_arities_prop
2845   { underscore }
2846   { 0 }
2847 \cs_gset_protected:Npn
2848   \markdownRendererHash
2849   {
2850     \markdownRendererHashPrototype
2851   }
2852 \seq_gput_right:Nn
2853   \g_@@_renderers_seq
2854   { hash }
2855 \prop_gput:Nnn
2856   \g_@@_renderer_arities_prop
2857   { hash }
2858   { 0 }
2859 \cs_gset_protected:Npn
2860   \markdownRendererCircumflex
2861   {
2862     \markdownRendererCircumflexPrototype
2863   }
2864 \seq_gput_right:Nn
2865   \g_@@_renderers_seq
2866   { circumflex }
2867 \prop_gput:Nnn
2868   \g_@@_renderer_arities_prop
2869   { circumflex }
2870   { 0 }
2871 \cs_gset_protected:Npn
2872   \markdownRendererBackslash
2873   {

```

```

2874     \markdownRendererBackslashPrototype
2875   }
2876 \seq_gput_right:Nn
2877   \g_@@_renderers_seq
2878   { backslash }
2879 \prop_gput:Nnn
2880   \g_@@_renderer_arities_prop
2881   { backslash }
2882   { 0 }
2883 \cs_gset_protected:Npn
2884   \markdownRendererTilde
2885   {
2886     \markdownRendererTildePrototype
2887   }
2888 \seq_gput_right:Nn
2889   \g_@@_renderers_seq
2890   { tilde }
2891 \prop_gput:Nnn
2892   \g_@@_renderer_arities_prop
2893   { tilde }
2894   { 0 }
2895 \cs_gset_protected:Npn
2896   \markdownRendererPipe
2897   {
2898     \markdownRendererPipePrototype
2899   }
2900 \seq_gput_right:Nn
2901   \g_@@_renderers_seq
2902   { pipe }
2903 \prop_gput:Nnn
2904   \g_@@_renderer_arities_prop
2905   { pipe }
2906   { 0 }
2907 \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.

```

2908 \ExplSyntaxOn
2909 \cs_gset_protected:Npn
2910   \markdownRendererStrikeThrough
2911   {
2912     \markdownRendererStrikeThroughPrototype
2913   }

```

```

2914 \seq_gput_right:Nn
2915   \g_@@_renderers_seq
2916   { strikeThrough }
2917 \prop_gput:Nnn
2918   \g_@@_renderer_arities_prop
2919   { strikeThrough }
2920   { 1 }
2921 \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.

```

2922 \ExplSyntaxOn
2923 \cs_gset_protected:Npn
2924   \markdownRendererSubscript
2925   {
2926     \markdownRendererSubscriptPrototype
2927   }
2928 \seq_gput_right:Nn
2929   \g_@@_renderers_seq
2930   { subscript }
2931 \prop_gput:Nnn
2932   \g_@@_renderer_arities_prop
2933   { subscript }
2934   { 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.

```

2935 \cs_gset_protected:Npn
2936   \markdownRendererSuperscript
2937   {
2938     \markdownRendererSuperscriptPrototype
2939   }
2940 \seq_gput_right:Nn
2941   \g_@@_renderers_seq
2942   { superscript }
2943 \prop_gput:Nnn
2944   \g_@@_renderer_arities_prop
2945   { superscript }
2946   { 1 }
2947 \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.

```
2948 \ExplSyntaxOn
2949 \cs_gset_protected:Npn
2950   \markdownRendererTableAttributeContextBegin
2951   {
2952     \markdownRendererTableAttributeContextBeginPrototype
2953   }
2954 \seq_gput_right:Nn
2955   \g_@@_renderers_seq
2956   { tableAttributeContextBegin }
2957 \prop_gput:Nnn
2958   \g_@@_renderer_arities_prop
2959   { tableAttributeContextBegin }
2960   { 0 }
2961 \cs_gset_protected:Npn
2962   \markdownRendererTableAttributeContextEnd
2963   {
2964     \markdownRendererTableAttributeContextEndPrototype
2965   }
2966 \seq_gput_right:Nn
2967   \g_@@_renderers_seq
2968   { tableAttributeContextEnd }
2969 \prop_gput:Nnn
2970   \g_@@_renderer_arities_prop
2971   { tableAttributeContextEnd }
2972   { 0 }
2973 \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.

```

2974 \ExplSyntaxOn
2975 \cs_gset_protected:Npn
2976   \markdownRendererTable
2977   {
2978     \markdownRendererTablePrototype
2979   }
2980 \seq_gput_right:Nn
2981   \g_@@_renderers_seq
2982   { table }
2983 \prop_gput:Nnn
2984   \g_@@_renderer_arities_prop
2985   { table }
2986   { 3 }
2987 \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.

```

2988 \ExplSyntaxOn
2989 \cs_gset_protected:Npn
2990   \markdownRendererInlineMath
2991   {
2992     \markdownRendererInlineMathPrototype
2993   }
2994 \seq_gput_right:Nn
2995   \g_@@_renderers_seq
2996   { inlineMath }
2997 \prop_gput:Nnn
2998   \g_@@_renderer_arities_prop
2999   { inlineMath }
3000   { 1 }
3001 \cs_gset_protected:Npn
3002   \markdownRendererDisplayMath
3003   {
3004     \markdownRendererDisplayMathPrototype
3005   }
3006 \seq_gput_right:Nn
3007   \g_@@_renderers_seq
3008   { displayMath }
3009 \prop_gput:Nnn
3010   \g_@@_renderer_arities_prop

```

```

3011 { displayMath }
3012 { 1 }
3013 \ExplSyntaxOff

```

#### 2.2.5.41 Thematic Break Renderer

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

```

3014 \ExplSyntaxOn
3015 \cs_gset_protected:Npn
3016   \markdownRendererThematicBreak
3017   {
3018     \markdownRendererThematicBreakPrototype
3019   }
3020 \seq_gput_right:Nn
3021   \g_@@_renderers_seq
3022   { thematicBreak }
3023 \prop_gput:Nnn
3024   \g_@@_renderer_arities_prop
3025   { thematicBreak }
3026   { 0 }
3027 \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.

```

3028 \ExplSyntaxOn
3029 \cs_gset_protected:Npn
3030   \markdownRendererTickedBox
3031   {
3032     \markdownRendererTickedBoxPrototype
3033   }
3034 \seq_gput_right:Nn
3035   \g_@@_renderers_seq
3036   { tickedBox }
3037 \prop_gput:Nnn
3038   \g_@@_renderer_arities_prop
3039   { tickedBox }
3040   { 0 }
3041 \cs_gset_protected:Npn
3042   \markdownRendererHalfTickedBox
3043   {

```

```

3044     \markdownRendererHalfTickedTextBoxPrototype
3045   }
3046 \seq_gput_right:Nn
3047   \g_@@_renderers_seq
3048   { halfTickedTextBox }
3049 \prop_gput:Nnn
3050   \g_@@_renderer_arities_prop
3051   { halfTickedTextBox }
3052   { 0 }
3053 \cs_gset_protected:Npn
3054   \markdownRendererUntickedTextBox
3055   {
3056     \markdownRendererUntickedTextBoxPrototype
3057   }
3058 \seq_gput_right:Nn
3059   \g_@@_renderers_seq
3060   { untickedTextBox }
3061 \prop_gput:Nnn
3062   \g_@@_renderer_arities_prop
3063   { untickedTextBox }
3064   { 0 }
3065 \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.

```

3066 \ExplSyntaxOn
3067 \cs_gset_protected:Npn
3068   \markdownRendererWarning
3069   {
3070     \markdownRendererWarningPrototype
3071   }
3072 \cs_gset_protected:Npn
3073   \markdownRendererError
3074   {

```



```

3075     \markdownRendererErrorPrototype
3076   }
3077 \seq_gput_right:Nn
3078   \g_@@_renderers_seq
3079   { warning }
3080 \prop_gput:Nnn
3081   \g_@@_renderer_arities_prop
3082   { warning }
3083   { 4 }
3084 \seq_gput_right:Nn
3085   \g_@@_renderers_seq
3086   { error }
3087 \prop_gput:Nnn
3088   \g_@@_renderer_arities_prop
3089   { error }
3090   { 4 }
3091 \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.

```

3092 \ExplSyntaxOn
3093 \cs_gset_protected:Npn
3094   \markdownRendererJekyllDataBegin
3095   {
3096     \markdownRendererJekyllDataBeginPrototype
3097   }
3098 \seq_gput_right:Nn
3099   \g_@@_renderers_seq
3100   { jekyllDataBegin }
3101 \prop_gput:Nnn
3102   \g_@@_renderer_arities_prop
3103   { jekyllDataBegin }
3104   { 0 }
3105 \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.

```

3106 \ExplSyntaxOn
3107 \cs_gset_protected:Npn
3108   \markdownRendererJekyllDataEnd
3109   {
3110     \markdownRendererJekyllDataEndPrototype
3111   }

```

```

3112 \seq_gput_right:Nn
3113   \g_@@_renderers_seq
3114   { jekyllDataEnd }
3115 \prop_gput:Nnn
3116   \g_@@_renderer_arities_prop
3117   { jekyllDataEnd }
3118   { 0 }
3119 \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.

```

3120 \ExplSyntaxOn
3121 \cs_gset_protected:Npn
3122   \markdownRendererJekyllDataMappingBegin
3123   {
3124     \markdownRendererJekyllDataMappingBeginPrototype
3125   }
3126 \seq_gput_right:Nn
3127   \g_@@_renderers_seq
3128   { jekyllDataMappingBegin }
3129 \prop_gput:Nnn
3130   \g_@@_renderer_arities_prop
3131   { jekyllDataMappingBegin }
3132   { 2 }
3133 \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.

```

3134 \ExplSyntaxOn
3135 \cs_gset_protected:Npn
3136   \markdownRendererJekyllDataMappingEnd
3137   {
3138     \markdownRendererJekyllDataMappingEndPrototype
3139   }
3140 \seq_gput_right:Nn
3141   \g_@@_renderers_seq
3142   { jekyllDataMappingEnd }
3143 \prop_gput:Nnn
3144   \g_@@_renderer_arities_prop
3145   { jekyllDataMappingEnd }
3146   { 0 }
3147 \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.

```

3148 \ExplSyntaxOn
3149 \cs_gset_protected:Npn
3150   \markdownRendererJekyllDataSequenceBegin
3151   {
3152     \markdownRendererJekyllDataSequenceBeginPrototype
3153   }
3154 \seq_gput_right:Nn
3155   \g_@@_renderers_seq
3156   { jekyllDataSequenceBegin }
3157 \prop_gput:Nnn
3158   \g_@@_renderer_arities_prop
3159   { jekyllDataSequenceBegin }
3160   { 2 }
3161 \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.

```

3162 \ExplSyntaxOn
3163 \cs_gset_protected:Npn
3164   \markdownRendererJekyllDataSequenceEnd
3165   {
3166     \markdownRendererJekyllDataSequenceEndPrototype
3167   }
3168 \seq_gput_right:Nn
3169   \g_@@_renderers_seq
3170   { jekyllDataSequenceEnd }
3171 \prop_gput:Nnn
3172   \g_@@_renderer_arities_prop
3173   { jekyllDataSequenceEnd }
3174   { 0 }
3175 \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.

```

3176 \ExplSyntaxOn
3177 \cs_gset_protected:Npn

```

```

3178 \markdownRendererJekyllDataBoolean
3179 {
3180   \markdownRendererJekyllDataBooleanPrototype
3181 }
3182 \seq_gput_right:Nn
3183 \g_@@_renderers_seq
3184 { jekyllDataBoolean }
3185 \prop_gput:Nnn
3186 \g_@@_renderer_arities_prop
3187 { jekyllDataBoolean }
3188 { 2 }
3189 \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.

```

3190 \ExplSyntaxOn
3191 \cs_gset_protected:Npn
3192 \markdownRendererJekyllDataNumber
3193 {
3194   \markdownRendererJekyllDataNumberPrototype
3195 }
3196 \seq_gput_right:Nn
3197 \g_@@_renderers_seq
3198 { jekyllDataNumber }
3199 \prop_gput:Nnn
3200 \g_@@_renderer_arities_prop
3201 { jekyllDataNumber }
3202 { 2 }
3203 \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  $\text{T}_{\text{E}}\text{X}$  characters in the string have been replaced by  $\text{T}_{\text{E}}\text{X}$  macros, `\markdownRendererJekyllDataProgrammaticString` receives the raw scalar value. Therefore, whereas the `\markdownRendererJekyllDataTypographicString` macro is more appropriate for texts that are supposed to be typeset with  $\text{T}_{\text{E}}\text{X}$ , 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 T<sub>E</sub>X.

```

3204 \ExplSyntaxOn
3205 \cs_gset_protected:Npn
3206   \markdownRendererJekyllDataTypographicString
3207   {
3208     \markdownRendererJekyllDataTypographicStringPrototype
3209   }
3210 \cs_gset_protected:Npn
3211   \markdownRendererJekyllDataProgrammaticString
3212   {
3213     \markdownRendererJekyllDataProgrammaticStringPrototype
3214   }
3215 \seq_gput_right:Nn
3216   \g_@@_renderers_seq
3217   { jekyllDataTypographicString }
3218 \prop_gput:Nnn
3219   \g_@@_renderer_arities_prop
3220   { jekyllDataTypographicString }
3221   { 2 }
3222 \seq_gput_right:Nn
3223   \g_@@_renderers_seq
3224   { jekyllDataProgrammaticString }
3225 \prop_gput:Nnn
3226   \g_@@_renderer_arities_prop
3227   { jekyllDataProgrammaticString }
3228   { 2 }
3229 \ExplSyntaxOff

```

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

```

3230 \ExplSyntaxOn
3231 \cs_gset:Npn
3232   \markdownRendererJekyllDataTypographicString
3233   {
3234     \cs_if_exist:NTF
3235       \markdownRendererJekyllDataString
3236       {
3237         \@@_if_option:nTF
3238           { experimental }
3239           {
3240             \markdownError
3241             {
3242               The~jekyllDataString~renderer~has~been~deprecated,~

```

```

3243         to-be-removed-in-Markdown-4.0.0
3244     }
3245 }
3246 {
3247     \markdownWarning
3248     {
3249         The~jekyllDataString~renderer~has~been~deprecated,~
3250         to-be-removed-in-Markdown-4.0.0
3251     }
3252     \markdownRendererJekyllDataString
3253 }
3254 }
3255 {
3256     \cs_if_exist:NTF
3257     \markdownRendererJekyllDataStringPrototype
3258     {
3259         \@@_if_option:nTF
3260         { experimental }
3261         {
3262             \markdownError
3263             {
3264                 The~jekyllDataString~renderer~prototype~
3265                 has~been~deprecated,~
3266                 to-be-removed-in-Markdown-4.0.0
3267             }
3268         }
3269         {
3270             \markdownWarning
3271             {
3272                 The~jekyllDataString~renderer~prototype~
3273                 has~been~deprecated,~
3274                 to-be-removed-in-Markdown-4.0.0
3275             }
3276             \markdownRendererJekyllDataStringPrototype
3277         }
3278     }
3279     {
3280         \markdownRendererJekyllDataTypographicStringPrototype
3281     }
3282 }
3283 }
3284 \seq_gput_right:Nn
3285 \g_@@_renderers_seq
3286 { jekyllDataString }
3287 \prop_gput:Nnn
3288 \g_@@_renderer_arities_prop
3289 { jekyllDataString }

```

```

3290 { 2 }
3291 \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.

```

3292 \ExplSyntaxOn
3293 \cs_gset_protected:Npn
3294   \markdownRendererJekyllDataEmpty
3295   {
3296     \markdownRendererJekyllDataEmptyPrototype
3297   }
3298 \seq_gput_right:Nn
3299   \g_@@_renderers_seq
3300   { jekyllDataEmpty }
3301 \prop_gput:Nnn
3302   \g_@@_renderer_arities_prop
3303   { jekyllDataEmpty }
3304   { 1 }
3305 \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.

```

3306 \ExplSyntaxOn
3307 \cs_new:Nn \@@_define_renderers:
3308   {
3309     \seq_map_inline:Nn
3310       \g_@@_renderers_seq
3311       {
3312         \@@_define_renderer:n
3313         { ##1 }
3314       }
3315   }
3316 \cs_new:Nn \@@_define_renderer:n
3317   {

```

```

3318 \@@_renderer_tl_to_csname:nN
3319 { #1 }
3320 \l_tmpa_tl
3321 \prop_get:NnN
3322 \g_@@_renderer_arities_prop
3323 { #1 }
3324 \l_tmpb_tl
3325 \@@_define_renderer:ncV
3326 { #1 }
3327 { \l_tmpa_tl }
3328 \l_tmpb_tl
3329 }
3330 \cs_new:Nn \@@_renderer_tl_to_csname:nN
3331 {
3332 \tl_set:Nn
3333 \l_tmpa_tl
3334 { \str_uppercase:n { #1 } }
3335 \tl_set:Nx
3336 #2
3337 {
3338 markdownRenderer
3339 \tl_head:f { \l_tmpa_tl }
3340 \tl_tail:n { #1 }
3341 }
3342 }
3343 \tl_new:N
3344 \l_@@_renderer_definition_tl
3345 \bool_new:N
3346 \g_@@_appending_renderer_bool
3347 \bool_new:N
3348 \g_@@_unprotected_renderer_bool
3349 \cs_new:Nn \@@_define_renderer:nNn
3350 {
3351 \keys_define:nn
3352 { markdown/options/renderers }
3353 {
3354 #1 .code:n = {
3355 \tl_set:Nn
3356 \l_@@_renderer_definition_tl
3357 { ##1 }
3358 \regex_replace_all:nnN
3359 { \cP\#0 }
3360 { #1 }
3361 \l_@@_renderer_definition_tl
3362 \bool_if:NT
3363 \g_@@_appending_renderer_bool
3364 {

```



```

3365         \@@_tl_set_from_cs:NNn
3366         \l_tmpa_tl
3367         #2
3368         { #3 }
3369         \tl_put_left:NV
3370         \l_@@_renderer_definition_tl
3371         \l_tmpa_tl
3372     }
3373     \bool_if:NTF
3374     \g_@@_unprotected_renderer_bool
3375     {
3376         \tl_set:Nn
3377         \l_tmpa_tl
3378         { \cs_set:Npn }
3379     }
3380     {
3381         \tl_set:Nn
3382         \l_tmpa_tl
3383         { \cs_set_protected:Npn }
3384     }
3385     \exp_last_unbraced:NNV
3386     \cs_generate_from_arg_count:NNnV
3387     #2
3388     \l_tmpa_tl
3389     { #3 }
3390     \l_@@_renderer_definition_tl
3391 },
3392 }

```

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.

```

3393     \str_if_eq:nnT
3394     { #1 }
3395     { jekyllDataString }
3396     {
3397         \cs_undefine:N
3398         #2
3399     }
3400 }

```

We define the function `\@@_tl_set_from_cs:NNn` [13]. 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.

```

3401 \cs_new_protected:Nn
3402 \@@_tl_set_from_cs:NNn

```

```

3403 {
3404     \tl_set:Nn
3405         \l_tmpa_tl
3406         { #2 }
3407     \int_step_inline:nn
3408         { #3 }
3409     {
3410         \exp_args:Nnc
3411             \tl_put_right:Nn
3412                 \l_tmpa_tl
3413                 { @@_tl_set_from_cs_parameter_ ##1 }
3414     }
3415     \exp_args:NNV
3416         \tl_set:No
3417             \l_tmpb_tl
3418             \l_tmpa_tl
3419     \regex_replace_all:nnN
3420         { \cP. }
3421         { \0\0 }
3422         \l_tmpb_tl
3423     \int_step_inline:nn
3424         { #3 }
3425     {
3426         \regex_replace_all:nnN
3427             { \c { @@_tl_set_from_cs_parameter_ ##1 } }
3428             { \cP\# ##1 }
3429             \l_tmpb_tl
3430     }
3431     \tl_set:NV
3432         #1
3433         \l_tmpb_tl
3434 }
3435 \cs_generate_variant:Nn
3436     \@@_define_renderer:nNn
3437     { ncV }
3438 \cs_generate_variant:Nn
3439     \cs_generate_from_arg_count:NNnn
3440     { NNnV }
3441 \cs_generate_variant:Nn
3442     \tl_put_left:Nn
3443     { Nv }
3444 \keys_define:nn
3445     { markdown/options }
3446     {
3447         renderers .code:n = {
3448             \bool_gset_false:N
3449                 \g_@@_unprotected_renderer_bool

```

```

3450     \keys_set:nn
3451     { markdown/options/renderers }
3452     { #1 }
3453 },
3454 unprotectedRenderers .code:n = {
3455     \bool_gset_true:N
3456     \g_@@_unprotected_renderer_bool
3457     \keys_set:nn
3458     { markdown/options/renderers }
3459     { #1 }
3460 },
3461 }

```

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.
  }
}

```

```

3462 \tl_new:N
3463 \l_@@_renderer_glob_definition_tl
3464 \seq_new:N
3465 \l_@@_renderer_glob_results_seq
3466 \regex_const:Nn
3467 \c_@@_appending_key_regex
3468 { \s*+ $ }
3469 \keys_define:nn
3470 { markdown/options/renderers }
3471 {
3472     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 += {...},
    },
  },
},
}
}

```

```

3473 % TODO: Use `\\regex_if_match` in TeX Live 2025.
3474 \\regex_match:NVTf
3475 \\c_@@_appending_key_regex
3476 \\l_keys_key_str
3477 {
3478   \\bool_gset_true:N
3479   \\g_@@_appending_renderer_bool
3480   \\tl_set:Nv
3481   \\l_tmpa_tl
3482   \\l_keys_key_str
3483   \\regex_replace_once:NnN
3484   \\c_@@_appending_key_regex
3485   { }
3486   \\l_tmpa_tl
3487   \\tl_set:Nx
3488   \\l_tmpb_tl
3489   { { \\l_tmpa_tl } = }
3490   \\tl_put_right:Nn
3491   \\l_tmpb_tl
3492   { { #1 } }
3493   \\keys_set:nV
3494   { markdown/options/renderers }
3495   \\l_tmpb_tl
3496   \\bool_gset_false:N
3497   \\g_@@_appending_renderer_bool
3498 }

```

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

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

Wildcards and enumerations can be combined:

```
\markdownSetup{
  renderers = {
    *1Item(1End) = {"},      % 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.
  }
}
```

```
3499      {
3500          \@@_glob_seq:VnN
3501          \l_keys_key_str
3502          { g_@@_renderers_seq }
3503          \l_@@_renderer_glob_results_seq
3504          \seq_if_empty:NTF
3505          \l_@@_renderer_glob_results_seq
3506          {
3507              \msg_error:nnV
3508              { markdown }
3509              { undefined-renderer }
3510              \l_keys_key_str
3511          }
3512      {
3513          \tl_set:Nn
```

```

3514         \l_@@_renderer_glob_definition_tl
3515         { \exp_not:n { #1 } }
3516     \seq_map_inline:Nn
3517     \l_@@_renderer_glob_results_seq
3518     {
3519         \tl_set:Nn
3520         \l_tmpa_tl
3521         { { ##1 } = }
3522         \tl_put_right:Nx
3523         \l_tmpa_tl
3524         { { \l_@@_renderer_glob_definition_tl } }
3525         \keys_set:nV
3526         { markdown/options/renderers }
3527         \l_tmpa_tl
3528     }
3529 }
3530 }
3531 },
3532 }
3533 \msg_new:nnn
3534 { markdown }
3535 { undefined-renderer }
3536 {
3537     Renderer~#1~is~undefined.
3538 }
3539 \cs_generate_variant:Nn
3540 \@@_glob_seq:nnN
3541 { VnN }
3542 \cs_generate_variant:Nn
3543 \cs_generate_from_arg_count:NNnn
3544 { cNvV }
3545 \cs_generate_variant:Nn
3546 \msg_error:nnn
3547 { nnV }
3548 \prg_generate_conditional_variant:Nnn
3549 % TODO: Use `regex_if_match` in TeX Live 2025.
3550 \regex_match:Nn % noqa: w202
3551 { NV }
3552 { TF }
3553 \prop_new:N
3554 \g_@@_glob_cache_prop
3555 \tl_new:N
3556 \l_@@_current_glob_tl
3557 \cs_new:Nn
3558 \@@_glob_seq:nnN
3559 {
3560     \tl_set:Nn

```

```

3561     \l_@@_current_glob_tl
3562     { ^ #1 $ }
3563 \prop_get:NeNTF
3564     \g_@@_glob_cache_prop
3565     { #2 / \l_@@_current_glob_tl }
3566     \l_tmpa_clist
3567     {
3568         \seq_set_from_clist:NN
3569         #3
3570         \l_tmpa_clist
3571     }
3572     {
3573         \seq_clear:N
3574         #3
3575         \regex_replace_all:nnN
3576         { \* }
3577         { .* }
3578         \l_@@_current_glob_tl
3579         \regex_set:NV
3580         \l_tmpa_regex
3581         \l_@@_current_glob_tl
3582         \seq_map_inline:cn
3583         { #2 }
3584         {
3585             % TODO: Use ` \regex_if_match ` in TeX Live 2025.
3586             \regex_match:NnT % noqa: w202
3587             \l_tmpa_regex
3588             { ##1 }
3589             {
3590                 \seq_put_right:Nn
3591                 #3
3592                 { ##1 }
3593             }
3594         }
3595         \clist_set_from_seq:NN
3596         \l_tmpa_clist
3597         #3
3598         \prop_gput:NeV
3599         \g_@@_glob_cache_prop
3600         { #2 / \l_@@_current_glob_tl }
3601         \l_tmpa_clist
3602     }
3603 }
3604 \cs_generate_variant:Nn
3605     \regex_set:Nn
3606     { NV }
3607 \cs_generate_variant:Nn

```

```

3608 \prop_gput:Nnn
3609 { NeV }

```

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

```

3610 \str_if_eq:VVT
3611 \c_@@_top_layer_tl
3612 \c_@@_option_layer_plain_tex_tl
3613 {
3614   \@@_define_renderers:
3615 }
3616 \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 [3] for the module `markdown/jekyllData`.

```

3617 \ExplSyntaxOn
3618 \keys_define:nn
3619 { markdown/jekyllData }
3620 { }
3621 \ExplSyntaxOff

```

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

```

3622 \ExplSyntaxOn
3623 \@@_with_various_cases:nn
3624 { jekyllDataRenderers }
3625 {
3626   \keys_define:nn
3627   { markdown/options }
3628   {
3629     #1 .code:n = {
3630       \tl_set:Nn
3631       \l_tmpa_tl
3632       { ##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.

```

3633       \tl_replace_all:NnV
3634       \l_tmpa_tl

```



```

3635         { / }
3636         \c_backslash_str
3637         \keys_set:nV
3638         { markdown/options/jekyll-data-renderers }
3639         \l_tmpa_tl
3640     },
3641 }
3642 }
3643 \keys_define:nn
3644 { markdown/options/jekyll-data-renderers }
3645 {
3646     unknown .code:n = {
3647         \tl_set_eq:NN
3648         \l_tmpa_tl
3649         \l_keys_key_str
3650         \tl_replace_all:NVn
3651         \l_tmpa_tl
3652         \c_backslash_str
3653         { / }
3654         \tl_put_right:Nn
3655         \l_tmpa_tl
3656         {
3657             .code:n = { #1 }
3658         }
3659         \keys_define:nV
3660         { markdown/jekyllData }
3661         \l_tmpa_tl
3662     }
3663 }
3664 \ExplSyntaxOff

```

For complex YAML metadata, the option `jekyllDataKeyValue=<module>` [14] can be used to route the processing of all YAML metadata in the current  $\text{\TeX}$  group to the key-values from  $\langle module \rangle$ .

### 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` de-

unesprotected functions, which are easier to expand and may be preferable for programming.

```

3665 \ExplSyntaxOn
3666 \cs_new:Nn \@@_define_renderer_prototypes:
3667 {
3668   \seq_map_inline:Nn
3669     \g_@@_renderers_seq
3670     {
3671       \@@_define_renderer_prototype:n
3672       { #1 }
3673     }
3674 }
3675 \cs_new:Nn \@@_define_renderer_prototype:n
3676 {
3677   \@@_renderer_prototype_tl_to_csname:nN
3678   { #1 }
3679   \l_tmpa_tl
3680   \prop_get:NnN
3681     \g_@@_renderer_arities_prop
3682     { #1 }
3683     \l_tmpb_tl
3684   \@@_define_renderer_prototype:ncV
3685   { #1 }
3686   { \l_tmpa_tl }
3687   \l_tmpb_tl
3688 }
3689 \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
3690 {
3691   \tl_set:Nn
3692     \l_tmpa_tl
3693     { \str_uppercase:n { #1 } }
3694   \tl_set:Nx
3695     #2
3696     {
3697       markdownRenderer
3698       \tl_head:f { \l_tmpa_tl }
3699       \tl_tail:n { #1 }
3700       Prototype
3701     }
3702 }
3703 \tl_new:N
3704   \l_@@_renderer_prototype_definition_tl
3705 \bool_new:N
3706   \g_@@_appending_renderer_prototype_bool
3707 \bool_new:N
3708   \g_@@_unprotected_renderer_prototype_bool
3709 \cs_new:Nn \@@_define_renderer_prototype:nNn

```

```

3710 {
3711   \keys_define:nn
3712     { markdown/options/renderer-prototypes }
3713     {
3714       #1 .code:n = {
3715         \tl_set:Nn
3716           \l_@@_renderer_prototype_definition_tl
3717           { ##1 }
3718         \regex_replace_all:nnN
3719           { \cP\#0 }
3720           { #1 }
3721         \l_@@_renderer_prototype_definition_tl
3722         \bool_if:NT
3723           \g_@@_appending_renderer_prototype_bool
3724           {
3725             \@@_tl_set_from_cs:NNn
3726             \l_tmpa_tl
3727             #2
3728             { #3 }
3729             \tl_put_left:NV
3730               \l_@@_renderer_prototype_definition_tl
3731               \l_tmpa_tl
3732           }
3733         \bool_if:NTF
3734           \g_@@_unprotected_renderer_prototype_bool
3735           {
3736             \tl_set:Nn
3737               \l_tmpa_tl
3738               { \cs_set:Npn }
3739           }
3740           {
3741             \tl_set:Nn
3742               \l_tmpa_tl
3743               { \cs_set_protected:Npn }
3744           }
3745         \exp_last_unbraced:NNV
3746           \cs_generate_from_arg_count:NNnV
3747           #2
3748           \l_tmpa_tl
3749           { #3 }
3750         \l_@@_renderer_prototype_definition_tl
3751       },
3752     }

```

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.

```

3753     \str_if_eq:nnF
3754         { #1 }
3755         { jekyllDataString }
3756         {
3757             \cs_if_free:NT
3758                 #2
3759                 {
3760                     \cs_generate_from_arg_count:NNnn
3761                         #2
3762                     \cs_gset_protected:Npn
3763                         { #3 }
3764                         { }
3765                 }
3766         }
3767     }
3768 \cs_generate_variant:Nn
3769     \@@_define_renderer_prototype:nNn
3770     { ncV }

```

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

```

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

```

```

3771 \keys_define:nn
3772     { markdown/options/renderer-prototypes }
3773     {
3774         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 += {...},
    },
  },
},
}

```

```

3775 % TODO: Use `\\regex_if_match` in TeX Live 2025.
3776 \\regex_match:NVTf
3777 \\c_@@_appending_key_regex
3778 \\l_keys_key_str
3779 {
3780   \\bool_gset_true:N
3781   \\g_@@_appending_renderer_prototype_bool
3782   \\tl_set:Nv
3783   \\l_tmpa_tl
3784   \\l_keys_key_str
3785   \\regex_replace_once:NnN
3786   \\c_@@_appending_key_regex
3787   { }
3788   \\l_tmpa_tl
3789   \\tl_set:Nx
3790   \\l_tmpb_tl
3791   { { \\l_tmpa_tl } = }
3792   \\tl_put_right:Nn
3793   \\l_tmpb_tl
3794   { { #1 } }
3795   \\keys_set:nV
3796   { markdown/options/renderer-prototypes }
3797   \\l_tmpb_tl
3798   \\bool_gset_false:N
3799   \\g_@@_appending_renderer_prototype_bool

```

```
3800 }
```

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.
    jeekyllData(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.
}
```

```
3801 {
3802   \@@_glob_seq:VnN
3803   \l_keys_key_str
3804   { g_@@_renderers_seq }
3805   \l_@@_renderer_glob_results_seq
3806   \seq_if_empty:NTF
3807   \l_@@_renderer_glob_results_seq
3808   {
3809     \msg_error:nnV
3810     { markdown }
3811     { undefined-renderer-prototype }
3812     \l_keys_key_str
```

```

3813     }
3814     {
3815         \tl_set:Nn
3816         \l_@@_renderer_glob_definition_tl
3817         { \exp_not:n { #1 } }
3818         \seq_map_inline:Nn
3819         \l_@@_renderer_glob_results_seq
3820         {
3821             \tl_set:Nn
3822             \l_tmpa_tl
3823             { { ##1 } = }
3824             \tl_put_right:Nx
3825             \l_tmpa_tl
3826             { { \l_@@_renderer_glob_definition_tl } }
3827             \keys_set:nV
3828             { markdown/options/renderer-prototypes }
3829             \l_tmpa_tl
3830         }
3831     }
3832 }
3833 },
3834 }
3835 \msg_new:nnn
3836 { markdown }
3837 { undefined-renderer-prototype }
3838 {
3839     Renderer~prototype~#1~is~undefined.
3840 }
3841 \@@_with_various_cases:nn
3842 { rendererPrototypes }
3843 {
3844     \keys_define:nn
3845     { markdown/options }
3846     {
3847         #1 .code:n = {
3848             \bool_gset_false:N
3849             \g_@@_unprotected_renderer_prototype_bool
3850             \keys_set:nn
3851             { markdown/options/renderer-prototypes }
3852             { ##1 }
3853         },
3854     }
3855 }
3856 \@@_with_various_cases:nn
3857 { unprotectedRendererPrototypes }
3858 {
3859     \keys_define:nn

```

```

3860     { markdown/options }
3861     {
3862       #1 .code:n = {
3863         \bool_gset_true:N
3864         \g_@@_unprotected_renderer_prototype_bool
3865         \keys_set:nn
3866         { markdown/options/renderer-prototypes }
3867         { ##1 }
3868       },
3869     }
3870   }

```

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.

```

3871 \str_if_eq:VVT
3872   \c_@@_top_layer_tl
3873   \c_@@_option_layer_plain_tex_tl
3874   {
3875     \@@_define_renderer_prototypes:
3876   }
3877 \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*.

```

3878 \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.

```
3879 \let\markdownReadAndConvert\relax
3880 \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*.

```
3881 \catcode`\|=0\catcode`\=12%
3882 |gdef|markdownBegin{%
3883   |markdownReadAndConvert{\markdownEnd}%
3884                               {\|markdownEnd}}%
3885 |gdef|yamlBegin{%
3886   |begingroup
3887   |yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
3888   |markdownReadAndConvert{\yamlEnd}%
3889                               {\|yamlEnd}}%
3890 |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.

```
3891 \ExplSyntaxOn
3892 \keys_define:nn
3893   { markdown/options }
3894   {
3895     code .code:n = { #1 },
3896   }
3897 \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.

```
3898 \ExplSyntaxOn
```

```

3899 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
3900 \cs_generate_variant:Nn
3901   \tl_const:Nn
3902   { NV }
3903 \tl_if_exist:NF
3904   \c_@@_top_layer_tl
3905   {
3906     \tl_const:NV
3907       \c_@@_top_layer_tl
3908       \c_@@_option_layer_latex_tl
3909   }
3910 \ExplSyntaxOff
3911 \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 L<sup>A</sup>T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> parses package options.

### 2.3.1 Typesetting Markdown

The interface exposes the `markdown`, `markdown*`, and `yaml` L<sup>A</sup>T<sub>E</sub>X environments, and redefines the `\markinline`, `\markdownInput`, and `\yamlInput` commands.

#### 2.3.1.1 Typesetting Markdown and YAML directly

The `markdown` and `markdown*` L<sup>A</sup>T<sub>E</sub>X environments are aliases for the macros `\markdownBegin` and `\markdownEnd` exposed by the plain T<sub>E</sub>X interface.

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

```

3912 \newenvironment{markdown}\relax\relax
3913 \newenvironment{markdown*}[1]\relax\relax

```

Furthermore, both environments accept L<sup>A</sup>T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X 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` L<sup>A</sup>T<sub>E</sub>X environment by using it in other environments as follows:

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

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:

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

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 T<sub>E</sub>X'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:

<code>\newenvironment{bar}{\begin{foo}}{\end{foo}}</code>
---

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

<code>\newenvironment{bar}{\markdown[some, options]}{\markdownEnd}</code>
---

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 T<sub>E</sub>X interface.

```
3914 \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 T<sub>E</sub>X. Unlike the `\yamlInput` 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 YAML document.

The following example L<sup>A</sup>T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X hooks with the Markdown package

L<sup>A</sup>T<sub>E</sub>X provides an intricate hook management system that allows users to insert extra material before and after certain T<sub>E</sub>X macros and L<sup>A</sup>T<sub>E</sub>X environments, among other things. [15, Section 3.1.2]

The Markdown package is compatible with hooks and allows the use of hooks to insert extra material before T<sub>E</sub>X commands and before/after L<sup>A</sup>T<sub>E</sub>X 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 “`markdownfoo emphasis: _bar_ baz!</markdown>`”, 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 “`markdownfoo emphasis_bar_/emphasis baz!/markdown`”, 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.

L<sup>A</sup>T<sub>E</sub>X options map directly to the options recognized by the plain T<sub>E</sub>X interface (see Section 2.2.2) and to the markdown token renderers and their prototypes recognized by the plain T<sub>E</sub>X interface (see Sections 2.2.5 and 2.2.6).

The L<sup>A</sup>T<sub>E</sub>X options may be specified when loading the L<sup>A</sup>T<sub>E</sub>X package, when using the `markdown*` L<sup>A</sup>T<sub>E</sub>X 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 [16, Section 5.1], which supports the `finalizcache` and `frozenscache` package options with similar semantics to the `finalizeCache` and `frozenCache` plain T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X document sources for distribution.

```
3915 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}  
3916 \DeclareOption{frozenscache}{\markdownSetup{frozenCache}}
```

### 2.3.3.2 Generating Plain T<sub>E</sub>X Option, Token Renderer, and Token Renderer Prototype Macros and Key-Values

If L<sup>A</sup>T<sub>E</sub>X 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.

```
3917 \ExplSyntaxOn
3918 \str_if_eq:VVT
3919   \c_@@_top_layer_tl
3920   \c_@@_option_layer_latex_tl
3921   {
3922     \@@define_option_commands_and_keyvals:
3923     \@@define_renderers:
3924     \@@define_renderer_prototypes:
3925   }
3926 \ExplSyntaxOff
```

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

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

If the L<sup>A</sup>T<sub>E</sub>X option with keys `theme` or `import` is (repeatedly) specified in the `\usepackage` macro, the loading of the theme(s) will be postponed in first-in-first-out



order until after the Markdown  $\LaTeX$  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}
```

```
3927 \newif\ifmarkdownLaTeXLoaded
3928 \markdownLaTeXLoadedfalse
```

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

```
3929 \ExplSyntaxOn
3930 \prop_new:N
3931 \g_@@_latex_built_in_themes_prop
3932 \ExplSyntaxOff
```

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

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

```
3933 \AtEndOfPackage{\markdownLaTeXLoadedtrue}
```

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

```
3934 \ExplSyntaxOn
3935 \str_if_eq:VVT
3936 \c_@@_top_layer_tl
3937 \c_@@_option_layer_latex_tl
3938 {
3939   \use:c
3940     { ExplSyntaxOff }
3941   \AtEndOfPackage
3942     {
3943       \@@_if_option:nF
3944         { noDefaults }
3945       {
```

```

3946         \@@_if_option:nTF
3947         { experimental }
3948         {
3949             \@@_setup:n
3950             { theme = witiko/markdown/defaults@experimental }
3951         }
3952         {
3953             \@@_setup:n
3954             { theme = witiko/markdown/defaults }
3955         }
3956     }
3957 }
3958 \use:c
3959 { ExplSyntaxOn }
3960 }
3961 \ExplSyntaxOff

```

Please, see Section 3.3.2 for implementation details of the built-in L<sup>A</sup>T<sub>E</sub>X 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.

```

3962 \ExplSyntaxOn
3963 \tl_const:Nn \c_@@_option_layer_context_tl { context }
3964 \cs_generate_variant:Nn
3965   \tl_const:Nn
3966   { NV }
3967 \tl_if_exist:NF
3968   \c_@@_top_layer_tl
3969   {
3970       \tl_const:NV
3971       \c_@@_top_layer_tl
3972       \c_@@_option_layer_context_tl
3973   }
3974 \ExplSyntaxOff

```

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

```

3975 \writestatus{loading}{ConTeXt User Module / markdown}%
3976 \startmodule[markdown]
3977 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
3978   \do\#\do\^\do\_ \do\% \do\~}%
3979 \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.

```
3980 \let\startmarkdown\relax
3981 \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 ConT<sub>E</sub>Xt code showcases the usage of the `\startmarkdown` and `\stopmarkdown` macros:

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

The `\startyaml` and `\stopyaml` macros are aliases for the macros `\yamlBegin` and `\yamlEnd` exposed by the plain T<sub>E</sub>X interface.

```
3982 \let\startyaml\relax
3983 \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 ConT<sub>E</sub>Xt 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 T<sub>E</sub>X interface.

```
3984 \let\inputmarkdown\relax
```

Furthermore, the `\inputmarkdown` macro also accepts ConT<sub>E</sub>Xt interface options (see Section 2.4.2) as its optional argument. These options will only influence this markdown document.

The following example ConT<sub>E</sub>Xt 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 `TeX` interface.

```
3985 \let\inputyaml\relax
```

Furthermore, the `\inputyaml` macro also accepts `ConTeXt` interface options (see Section 2.4.2) as its optional argument. These options will only influence this `YAML` document.

The following example `ConTeXt` 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.yml}
\stoptext

```

## 2.4.2 Options

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

`ConTeXt` options map directly to the options recognized by the plain `TeX` interface (see Section 2.2.2).

The `ConTeXt` 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}`.

```

3986 \ExplSyntaxOn
3987 \cs_new:Npn
3988   \setupmarkdown
3989   [ #1 ]
3990   {
3991     \@@_setup:n

```

```

3992     { #1 }
3993 }

```

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

```

3994 \cs_gset_eq:NN
3995   \setupyaml
3996   \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.

```

3997 \cs_new:Nn \@@_caseless:N
3998 {
3999   \regex_replace_all:nnN
4000     { ([a-z])([A-Z]) }
4001     { \1 \c { str_lowercase:n } \cB\{ \2 \cE\} }
4002     #1
4003   \tl_set:Nx
4004     #1
4005     { #1 }
4006 }
4007 \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.

```

4008 \str_if_eq:VVT
4009   \c_@@_top_layer_tl
4010   \c_@@_option_layer_context_tl
4011 {
4012   \@@_define_option_commands_and_keyvals:
4013   \@@_define_renderers:
4014   \@@_define_renderer_prototypes:
4015 }

```

#### 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.

```
4016 \prop_new:N
4017 \g_@@_context_built_in_themes_prop
4018 \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.

```
4019 \startmodule[markdownthemewitiko_markdown_defaults]
4020 \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).

Furthermore, here are some abbreviations that we inherited from the Lunamark library and which are used throughout the Lua implementation.

```
4021 local upper, format, length =
4022     string.upper, string.format, string.len
4023 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, Cp, any =
4024     lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
4025     lpeg.Cmt, lpeg.Cc, lpeg.Ct, lpeg.B, lpeg.Cs, lpeg.Cp, lpeg.P(1)
```

### 3.1.1 Unicode Support

To start off, we load a Lua file named `markdown-unicode-data.lua` that contains our implementation of various Unicode algorithms.

```
4026 local early_warnings = {}
4027 local unicode_data = require("markdown-unicode-data")
4028 if metadata.version ~= unicode_data.metadata.version then
4029     table.insert(
4030         early_warnings,
4031         "markdown.lua " .. metadata.version .. " used with " ..
4032         "markdown-unicode-data.lua " .. unicode_data.metadata.version .. "."
4033     )
4034 end
```

In the following subsections, we'll write a second-order file named `markdown-unicode-data-generator.lua`. The code from this file will be executed during the compilation of the Markdown package and the standard output will be stored in a generated file named `markdown-unicode-data.lua`. First, let's define a couple helper functions.

The function `depth_first_search` performs the depth first search algorithm on a tree with nodes being tables with the key `_type` equal to either `intermediate` or `leaf` and with edges labeled with bytes.

Since the algorithm is implemented using recursion, it should only be used for the traversal of shallow trees to prevent exceeding the maximum recursion depth (usually 100).

```
4035 local function depth_first_search(node, path, visit, leave)
4036     assert(node._type == "intermediate")
4037     visit(node, path)
4038     for label, child in pairs(node) do
```



```

4039     if label == "_type" then
4040         goto continue
4041     end
4042     if child._type == "intermediate" then
4043         depth_first_search(child, path .. label, visit, leave)
4044     else
4045         assert(child._type == "leaf")
4046         visit(child, path)
4047     end
4048     ::continue::
4049 end
4050 leave(node, path)
4051 end

```

The function `serialize_byte_parser` produces the Lua code for a PEG parser that matches a single byte.

```

4052 local function serialize_byte_parser(byte)
4053     if byte == '"' then
4054         return "P('' .. byte .. '')"
4055     elseif byte == "\\" then
4056         return "P(\"\\\\\\\")"
4057     else
4058         return "P('' .. byte .. '')"
4059     end
4060 end

```

The function `serialize_byte_range_parser` produces the Lua code for a PEG parser that matches a contiguous range of bytes.

```

4061 local function serialize_byte_range_parser(start_byte, end_byte)
4062     assert(start_byte <= end_byte)
4063     if start_byte == "\\" then
4064         start_byte = "\\\\"
4065     end
4066     if end_byte == "\\" then
4067         end_byte = "\\\\"
4068     end
4069     if start_byte == '"' or end_byte == '"' then
4070         return "R('' .. start_byte .. end_byte .. '')"
4071     else
4072         return "R('' .. start_byte .. end_byte .. '')"
4073     end
4074 end

```

The function `serialize_replacement` produces the Lua code for a string literal with one or more UTF-8-encoded Unicode characters.

```

4075 local function serialize_replacement(codepoints)
4076     assert(#codepoints > 0)
4077     local buffer = {''''}

```

```

4078 for _, codepoint in ipairs(codepoints) do
4079     local code = utf8.char(codepoint)
4080     for i = 1, #code do
4081         local byte = code:sub(i, i)
4082         if byte == '"' then
4083             table.insert(buffer, '\\\\"')
4084         elseif byte == "\\" then
4085             table.insert(buffer, "\\\\"")
4086         else
4087             table.insert(buffer, byte)
4088         end
4089     end
4090 end
4091 table.insert(buffer, '')
4092 return table.concat(buffer)
4093 end

```

The function `read_decompositions` is an iterator that reads a file `UnicodeData.txt` that has previously been opened for reading and yields all canonical and compatibility decompositions from that file.

```

4094 local function read_decompositions(file)
4095     return function()
4096         local from_codepoint, mapping
4097         for line in file:lines() do
4098             from_codepoint, mapping
4099                 = line:match("^(%x+);[~;]*;%a*;%d+;%a*;( [<%a>%x ]*)")
4100             assert(from_codepoint ~= nil)
4101             if #mapping > 0 then
4102                 break
4103             end
4104         end
4105         if #mapping == 0 then
4106             return nil
4107         end
4108         from_codepoint = tonumber(from_codepoint, 16)
4109         local to_codepoints, decomposition_type = {}, "canonical"
4110         for raw_codepoint in mapping:gmatch("%S+") do
4111             assert(#raw_codepoint > 0)
4112             if raw_codepoint:sub(1, 1) == "<" then
4113                 decomposition_type = "compatibility"
4114             else
4115                 local codepoint = tonumber(raw_codepoint, 16)
4116                 table.insert(to_codepoints, codepoint)
4117             end
4118         end
4119         assert(#to_codepoints > 0)
4120         return decomposition_type, from_codepoint, to_codepoints

```

```

4121     end
4122 end

```

Next, let's define some aliases in the generated file.

```

4123 print("local P, R, Cc, C = lpeg.P, lpeg.R, lpeg.Cc, lpeg.C")
4124 print("local fail = P(false)")
4125 print("-- luacheck: push no max line length")

```

### 3.1.1.1 Canonical and Compatibility Decomposition

Low-level parsers that decompose UTF-8-encoded Unicode characters using either the canonical or the compatibility decomposition [17, Section 3.7] are organized in table `unicode_data.decomposition_mapping` based on the number of bytes they occupy after conversion to UTF-8.

First, let's read the file `UnicodeData.txt`.

```

4126 ;(function()
4127     local file = assert(io.open("UnicodeData.txt", "r"),
4128         [[Could not open file "UnicodeData.txt"]])

```

In order to minimize the size and speed of the parser, we will first construct prefix trees of UTF-8 encodings for all codepoints of a given decomposition type and code length.

```

4129     local decomposition_types = {"canonical", "compatibility"}
4130     local prefix_trees = {}
4131     for _, decomposition_type in ipairs(decomposition_types) do
4132         prefix_trees[decomposition_type] = {}
4133         for char_length = 1, 4 do
4134             prefix_trees[decomposition_type][char_length]
4135                 = {_type = "intermediate"}
4136         end
4137     end
4138     for decomposition_type, from_codepoint, to_codepoints
4139         in read_decompositions(file) do
4140         local from_code = utf8.char(from_codepoint)
4141         local node = prefix_trees[decomposition_type][#from_code]
4142         for i = 1, #from_code do
4143             local from_byte = from_code:sub(i, i)
4144             if i < #from_code then
4145                 if node[from_byte] == nil then
4146                     node[from_byte] = {_type = "intermediate"}
4147                 end
4148                 node = node[from_byte]
4149             else
4150                 table.insert(node, {from_byte, to_codepoints, _type = "leaf"})
4151             end
4152         end
4153     end
4154     assert(file:close())

```

Next, we will construct parsers out of the prefix trees.

```
4155 print("M.decomposition_mapping = {}")
4156 for _, decomposition_type in ipairs(decomposition_types) do
4157     print("M.decomposition_mapping." .. decomposition_type .. " = {}")
4158     for length, prefix_tree in pairs(prefix_trees[decomposition_type])
4159     do
4160         local subparsers = {}
4161         depth_first_search(prefix_tree, "", function(node, path)
4162             if node._type == "leaf" then
4163                 local from_byte, to_codepoints = table.unpack(node)
4164                 local suffix = serialize_byte_parser(from_byte)
4165                 .. " / " .. serialize_replacement(to_codepoints)
4166                 if subparsers[path] ~= nil then
4167                     subparsers[path] = subparsers[path] .. " + " .. suffix
4168                 else
4169                     subparsers[path] = suffix
4170                 end
4171             end
4172         end, function(_, path)
4173             if #path > 0 then
4174                 local byte = path:sub(#path, #path)
4175                 local parent_path = path:sub(1, #path-1)
4176                 local prefix = serialize_byte_parser(byte)
4177                 local suffix
4178                 if subparsers[path]:find(" %+ ") then
4179                     suffix = prefix .. " * (" .. subparsers[path] .. ")"
4180                 else
4181                     suffix = prefix .. " * " .. subparsers[path]
4182                 end
4183                 if subparsers[parent_path] ~= nil then
4184                     subparsers[parent_path] = subparsers[parent_path]
4185                     .. " + " .. suffix
4186                 else
4187                     subparsers[parent_path] = suffix
4188                 end
4189             else
4190                 print(
4191                     "M.decomposition_mapping." .. decomposition_type
4192                     .. "[" .. length .. "]" = " .. (subparsers[path] or "fail")
4193                 )
4194             end
4195         end)
4196     end
4197 end
4198 end)()
```

### 3.1.1.2 Hangul Syllable Decomposition

Low-level parsers that decompose UTF-8-encoded Unicode characters using the Hangul syllable decomposition [17, Section 3.12.2] are also organized in table [unicode\\_data.decomposition\\_mapping](#), previously defined in Section 3.1.1.1 based on the number of bytes they occupy after conversion to UTF-8.

First, let's read the file [HangulSyllableType.txt](#).

```
4199 ;(function()
4200   local file = assert(io.open("HangulSyllableType.txt", "r"),
4201     [[Could not open file "HangulSyllableType.txt"]])
4202   local syllable_types = {"LV", "LVT"}
4203   local prefix_trees = {}
4204   for _, syllable_type in ipairs(syllable_types) do
4205     prefix_trees[syllable_type] = {}
4206     for char_length = 1, 4 do
4207       prefix_trees[syllable_type][char_length]
4208         = {_type = "intermediate"}
4209     end
4210   end
4211   for line in file:lines() do
4212     if #line == 0 or line:sub(1, 1) == "#" then
4213       goto continue
4214     end
4215     local codepoint, syllable_type = line:match("^([%x.]+)%s*;%s*(%a*)")
4216     assert(codepoint ~= nil)
4217     if prefix_trees[syllable_type] == nil then
4218       goto continue
4219     end
4220     local codepoint_start, codepoint_end
4221     if codepoint:find("%.%.") then
4222       codepoint_start, codepoint_end
4223         = codepoint:match("^(%x+)%.%.(%x+)$")
4224     else
4225       codepoint_start, codepoint_end = codepoint, codepoint
4226     end
4227     codepoint_start = tonumber(codepoint_start, 16)
4228     codepoint_end = tonumber(codepoint_end, 16)
4229     local prev_code, prev_leaf_node
4230     for codepoint = codepoint_start, codepoint_end do
4231       local code = utf8.char(codepoint)
4232       local node = prefix_trees[syllable_type][#code]
4233       for i = 1, #code do
4234         local byte = code:sub(i, i)
4235         if i < #code then
4236           if node[byte] == nil then
```

```

4237         node[byte] = {_type = "intermediate"}
4238     end
4239     node = node[byte]
4240 else
4241     local leaf_node
4242     if (
4243         prev_code ~= nil
4244         and #prev_code == #code
4245         and code:sub(1, #code - 1)
4246             == prev_code:sub(1, #code - 1)
4247     ) then
4248         assert(prev_leaf_node ~= nil)
4249         leaf_node = prev_leaf_node
4250         leaf_node[2] = byte
4251     else
4252         leaf_node = {byte, byte, _type = "leaf"}
4253         table.insert(node, leaf_node)
4254     end
4255     prev_code, prev_leaf_node = code, leaf_node
4256 end
4257 end
4258 end
4259 ::continue::
4260 end
4261 assert(file:close())

```

Next, we will define constants and functions with the Hangul syllable (de)composition algorithm.

```

4262 print(string.format("local s_base = %d", tonumber("AC00", 16)))
4263 print(string.format("local l_base = %d", tonumber("1100", 16)))
4264 print(string.format("local v_base = %d", tonumber("1161", 16)))
4265 print(string.format("local t_base = %d", tonumber("11A7", 16)))
4266 print(string.format("local l_count = %d", 19))
4267 print(string.format("local v_count = %d", 21))
4268 print(string.format("local t_count = %d", 28))
4269 print("local n_count = v_count * t_count")
4270 print("local s_count = l_count * n_count")
4271
4272 print("local function hangul_decompose_LV(byte)")
4273 print("  local s = utf8.codepoint(byte)")
4274 print("  local s_index = s - s_base")
4275 print("  local l_index = s_index // n_count")
4276 print("  local v_index = (s_index % n_count) // t_count")
4277 print("  local l_part = l_base + l_index")
4278 print("  local v_part = v_base + v_index")
4279 print("  return utf8.char(l_part) .. utf8.char(v_part)")
4280 print("end")

```

```

4281
4282 print("local function hangul_decompose_LVT(byte)")
4283 print("  local s = utf8.codepoint(byte)")
4284 print("  local s_index = s - s_base")
4285 print("  local lv_index = (s_index // t_count) * t_count")
4286 print("  local t_index = s_index % t_count")
4287 print("  local lv_part = s_base + lv_index")
4288 print("  local t_part = t_base + t_index")
4289 print("  return utf8.char(lv_part) .. utf8.char(t_part)")
4290 print("end")
4291
4292 print("function M.hangul_compose(first_byte, second_byte)")
4293 print("  local last = utf8.codepoint(first_byte)")
4294 print("  local ch = utf8.codepoint(second_byte)")
4295 print("  local l_index = last - l_base")
4296 print("  if 0 <= l_index and l_index < l_count then")
4297 print("    local v_index = ch - v_base")
4298 print("    if 0 <= v_index and v_index < v_count then")
4299 print("      return utf8.char("
4300         s_base + (l_index * v_count + v_index) * t_count")
4301     ")")
4302 print("  end")
4303 print("end")
4304 print("  local s_index = last - s_base")
4305 print("  if 0 <= s_index and s_index < s_count")
4306 print("    and s_index % t_count == 0 then")
4307 print("    local t_index = ch - t_base")
4308 print("    if 0 < t_index and t_index < t_count then")
4309 print("      return utf8.char(last + t_index)")
4310 print("    end")
4311 print("  end")
4312 print("  return nil")
4313 print("end")

```

Next, we will construct parsers out of the prefix trees.

```

4314 print("M.decomposition_mapping.hangul = {}")
4315 for _, syllable_type in ipairs(syllable_types) do
4316   print("M.decomposition_mapping.hangul." .. syllable_type .. " = {}")
4317   for length, prefix_tree in pairs(prefix_trees[syllable_type]) do
4318     local subparsers = {}
4319     depth_first_search(prefix_tree, "", function(node, path)
4320       if node._type == "leaf" then
4321         local start_byte, end_byte = table.unpack(node)
4322         local suffix
4323         if start_byte == end_byte then
4324           suffix = serialize_byte_parser(start_byte)
4325         else
4326           assert(start_byte < end_byte)

```

```

4327         suffix = serialize_byte_range_parser(start_byte, end_byte)
4328     end
4329     if subparsers[path] ~= nil then
4330         subparsers[path] = subparsers[path] .. " + " .. suffix
4331     else
4332         subparsers[path] = suffix
4333     end
4334 end
4335 end, function(_, path)
4336     if #path > 0 then
4337         local byte = path:sub(#path, #path)
4338         local parent_path = path:sub(1, #path-1)
4339         local prefix = serialize_byte_parser(byte)
4340         local suffix
4341         if subparsers[path]:find(" %+ ") then
4342             suffix = prefix .. " * (" .. subparsers[path] .. ")"
4343         else
4344             suffix = prefix .. " * " .. subparsers[path]
4345         end
4346         if subparsers[parent_path] ~= nil then
4347             subparsers[parent_path] = subparsers[parent_path]
4348                 .. " + " .. suffix
4349         else
4350             subparsers[parent_path] = suffix
4351         end
4352     else
4353         if subparsers[path] then
4354             print(
4355                 "M.decomposition_mapping.hangul." .. syllable_type
4356                 .. "[" .. length .. "] = C(" .. subparsers[path] .. ")"
4357                 .. " / hangul_decompose_" .. syllable_type
4358             )
4359         else
4360             print(
4361                 "M.decomposition_mapping.hangul." .. syllable_type
4362                 .. "[" .. length .. "] = fail"
4363             )
4364         end
4365     end
4366 end)
4367 end
4368 end
4369 end)()

```

### 3.1.1.3 Canonical Composition

Low-level parsers that map pairs of UTF-8-encoded Unicode characters from a



canonical or compatibility decomposition into their primary composites [17, Section 3.11.6] are organized in table [unicode\\_data.composition\\_mapping](#) based on the number of bytes the characters occupy after conversion to UTF-8.

First, let's read the file [DerivedNormalizationProps.txt](#) and record all canonical decomposable characters that are not full composition exclusions.

```

4370 ;(function()
4371   local file = assert(io.open("DerivedNormalizationProps.txt", "r"),
4372     [[Could not open file "DerivedNormalizationProps.txt"]])
4373   local full_composition_exclusions = {}
4374   for line in file:lines() do
4375     if #line == 0 or line:sub(1, 1) == "#" then
4376       goto continue
4377     end
4378     local codepoint, property = line:match("^([%x.]+)%s*;%s*([%a_]+)")
4379     assert(codepoint ~= nil)
4380     if property ~= "Full_Composition_Exclusion" then
4381       goto continue
4382     end
4383     local codepoint_start, codepoint_end
4384     if codepoint:find("%.%.") then
4385       codepoint_start, codepoint_end
4386       = codepoint:match("^(%x+)%.%.(%x+)$")
4387     else
4388       codepoint_start, codepoint_end = codepoint, codepoint
4389     end
4390     codepoint_start = tonumber(codepoint_start, 16)
4391     codepoint_end = tonumber(codepoint_end, 16)
4392     for codepoint = codepoint_start, codepoint_end do
4393       full_composition_exclusions[codepoint] = true
4394     end
4395     ::continue::
4396   end
4397   assert(file:close())

```

Next, let's also read the file [UnicodeData.txt](#).

```

4398   file = assert(io.open("UnicodeData.txt", "r"),
4399     [[Could not open file "UnicodeData.txt"]])

```

In order to minimize the size and speed of the parser, we will first construct prefix trees of UTF-8 encodings for all pairs of codepoints of given code lengths.

```

4400   local prefix_trees = {starters = {}, both = {}}
4401   for first_char_length = 1, 4 do
4402     prefix_trees.starters[first_char_length]
4403     = {_type = "intermediate"}
4404     prefix_trees.both[first_char_length] = {}
4405     for second_char_length = 1, 4 do
4406       prefix_trees.both[first_char_length][second_char_length]

```

```

4407         = {_type = "intermediate"}
4408     end
4409 end
4410 local seen_starter_codes = {}
4411 for decomposition_type, to_codepoint, from_codepoints
4412     in read_decompositions(file) do
4413     if (
4414         decomposition_type ~= "canonical"
4415         or #from_codepoints ~= 2
4416         or full_composition_exclusions[to_codepoint]
4417     ) then
4418         goto continue
4419     end
4420     local starter_code = utf8.char(from_codepoints[1])
4421     local combining_character_code = utf8.char(from_codepoints[2])
4422     local starter_node = prefix_trees.starters[#starter_code]
4423     local both_node
4424         = prefix_trees.both[#starter_code][#combining_character_code]
4425     for i = 1, #starter_code do
4426         local from_byte = starter_code:sub(i, i)
4427         if both_node[from_byte] == nil then
4428             both_node[from_byte] = {_type = "intermediate"}
4429         end
4430         both_node = both_node[from_byte]
4431         if i < #starter_code then
4432             if starter_node[from_byte] == nil then
4433                 starter_node[from_byte] = {_type = "intermediate"}
4434             end
4435             starter_node = starter_node[from_byte]
4436         elseif seen_starter_codes[starter_code] == nil then
4437             seen_starter_codes[starter_code] = true
4438             table.insert(starter_node, {from_byte, _type = "leaf"})
4439         end
4440     end
4441     for i = 1, #combining_character_code do
4442         local from_byte = combining_character_code:sub(i, i)
4443         if i < #combining_character_code then
4444             if both_node[from_byte] == nil then
4445                 both_node[from_byte] = {_type = "intermediate"}
4446             end
4447             both_node = both_node[from_byte]
4448         else
4449             table.insert(
4450                 both_node,
4451                 {from_byte, to_codepoint, _type = "leaf"}
4452             )
4453         end

```

```

4454     end
4455     ::continue::
4456 end
4457 assert(file:close())

```

Next, we will construct parsers out of the prefix trees.

```

4458 print("M.composition_mapping = {starters = {}, both = {}}")
4459 for first_char_length = 1, 4 do
4460     local prefix_tree = prefix_trees.starters[first_char_length]
4461     local subparsers = {}
4462     depth_first_search(prefix_tree, "", function(node, path)
4463         if node._type == "leaf" then
4464             local from_byte = table.unpack(node)
4465             local suffix = serialize_byte_parser(from_byte)
4466             if subparsers[path] ~= nil then
4467                 subparsers[path] = subparsers[path] .. " + " .. suffix
4468             else
4469                 subparsers[path] = suffix
4470             end
4471         end
4472     end, function(_, path)
4473         if #path > 0 then
4474             local byte = path:sub(#path, #path)
4475             local parent_path = path:sub(1, #path-1)
4476             local prefix = serialize_byte_parser(byte)
4477             local suffix
4478             if subparsers[path]:find(" %+ ") then
4479                 suffix = prefix .. " * (" .. subparsers[path] .. ")"
4480             else
4481                 suffix = prefix .. " * " .. subparsers[path]
4482             end
4483             if subparsers[parent_path] ~= nil then
4484                 subparsers[parent_path] = subparsers[parent_path]
4485                     .. " + " .. suffix
4486             else
4487                 subparsers[parent_path] = suffix
4488             end
4489         else
4490             print(
4491                 "M.composition_mapping.starters["
4492                 .. first_char_length .. "] = " .. (subparsers[path] or "fail")
4493             )
4494         end
4495     end)
4496 print(
4497     string.format(
4498         "M.composition_mapping.both[%d] = {}",
4499         first_char_length

```

```

4500     )
4501 )
4502 for second_char_length = 1, 4 do
4503     prefix_tree
4504     = prefix_trees.both[first_char_length][second_char_length]
4505     subparsers = {}
4506     depth_first_search(prefix_tree, "", function(node, path)
4507         if node._type == "leaf" then
4508             local from_byte, to_codepoint = table.unpack(node)
4509             local suffix = serialize_byte_parser(from_byte)
4510             .. " / " .. serialize_replacement({to_codepoint})
4511             if subparsers[path] ~= nil then
4512                 subparsers[path] = subparsers[path] .. " + " .. suffix
4513             else
4514                 subparsers[path] = suffix
4515             end
4516         end
4517     end, function(_, path)
4518         if #path > 0 then
4519             local byte = path:sub(#path, #path)
4520             local parent_path = path:sub(1, #path-1)
4521             local prefix = serialize_byte_parser(byte)
4522             local suffix
4523             if subparsers[path]:find(" %+ ") then
4524                 suffix = prefix .. " * (" .. subparsers[path] .. ")"
4525             else
4526                 suffix = prefix .. " * " .. subparsers[path]
4527             end
4528             if subparsers[parent_path] ~= nil then
4529                 subparsers[parent_path] = subparsers[parent_path]
4530                     .. " + " .. suffix
4531             else
4532                 subparsers[parent_path] = suffix
4533             end
4534         else
4535             print(
4536                 "M.composition_mapping.both[" .. first_char_length .. "]["
4537                     .. second_char_length .. "] = "
4538                     .. (subparsers[path] or "fail")
4539             )
4540         end
4541     end)
4542 end
4543 end
4544 end)()

```

### 3.1.1.4 Case Folding

Low-level parsers that case-fold UTF-8-encoded Unicode characters using the full mapping (C and F) [17, Section 3.13.3] are organized in table `unicode_data.casefold_mapping` based on the number of bytes they occupy after conversion to UTF-8.

First, let's read the file `CaseFolding.txt`.

```

4545 ;(function()
4546   local file = assert(io.open("CaseFolding.txt", "r"),
4547     [[Could not open file "CaseFolding.txt"]])
In order to minimize the size and speed of the parser, we will first construct prefix
trees of UTF-8 encodings for all codepoints of a given code length.
4548   local prefix_trees = {}
4549   for char_length = 1, 4 do
4550     prefix_trees[char_length] = {_type = "intermediate"}
4551   end
4552   for line in file:lines() do
4553     if #line == 0 or line:sub(1, 1) == "#" then
4554       goto continue
4555     end
4556     local raw_from_codepoint, status, raw_to_codepoints
4557       = line:match("^(%x+); ([CFST]); ([%x ]+);")
4558     assert(raw_from_codepoint ~= nil)
4559     assert(status ~= nil)
4560     assert(raw_to_codepoints ~= nil)
4561     if status ~= "C" and status ~= "F" then
4562       goto continue
4563     end
4564     local from_codepoint = tonumber(raw_from_codepoint, 16)
4565     local to_codepoints = {}
4566     for raw_codepoint in raw_to_codepoints:gmatch('%x+') do
4567       local codepoint = tonumber(raw_codepoint, 16)
4568       table.insert(to_codepoints, codepoint)
4569     end
4570     local from_code = utf8.char(from_codepoint)
4571     local node = prefix_trees[#from_code]
4572     for i = 1, #from_code do
4573       local from_byte = from_code:sub(i, i)
4574       if i < #from_code then
4575         if node[from_byte] == nil then
4576           node[from_byte] = {_type = "intermediate"}
4577         end
4578         node = node[from_byte]
4579       else
4580         table.insert(node, {from_byte, to_codepoints, _type = "leaf"})
4581       end
4582     end
4583     ::continue::

```

```

4584 end
4585 assert(file:close())

```

Next, we will construct parsers out of the prefix trees.

```

4586 print("M.casefold_mapping = {}")
4587 for length, prefix_tree in pairs(prefix_trees) do
4588     local subparsers = {}
4589     depth_first_search(prefix_tree, "", function(node, path)
4590         if node._type == "leaf" then
4591             local from_byte, to_codepoints = table.unpack(node)
4592             local suffix = serialize_byte_parser(from_byte)
4593             .. " / " .. serialize_replacement(to_codepoints)
4594             if subparsers[path] ~= nil then
4595                 subparsers[path] = subparsers[path] .. " + " .. suffix
4596             else
4597                 subparsers[path] = suffix
4598             end
4599         end
4600     end, function(_, path)
4601         if #path > 0 then
4602             local byte = path:sub(#path, #path)
4603             local parent_path = path:sub(1, #path-1)
4604             local prefix = serialize_byte_parser(byte)
4605             local suffix
4606             if subparsers[path]:find(" %+ ") then
4607                 suffix = prefix .. " * (" .. subparsers[path] .. ")"
4608             else
4609                 suffix = prefix .. " * " .. subparsers[path]
4610             end
4611             if subparsers[parent_path] ~= nil then
4612                 subparsers[parent_path] = subparsers[parent_path]
4613                 .. " + " .. suffix
4614             else
4615                 subparsers[parent_path] = suffix
4616             end
4617         else
4618             print(
4619                 "M.casefold_mapping[" .. length .. "] = "
4620                 .. (subparsers[path] or "fail")
4621             )
4622         end
4623     end)
4624 end
4625 end)()

```

### 3.1.1.5 Character Categories

Low-level parsers of UTF-8-encoded Unicode characters from different general categories [17, Section 4.5] are organized in table `unicode_data.categories` based on the number of bytes they occupy after conversion to UTF-8.

First, let's read the file `UnicodeData.txt`.

```
4626 ;(function()
4627   local file = assert(io.open("UnicodeData.txt", "r"),
4628     [[Could not open file "UnicodeData.txt"]])
```

In order to minimize the size and speed of the parser, we will first construct prefix trees of UTF-8 encodings for all codepoints of a given Unicode category and code length.

```
4629   local categories = {"L", "N", "P", "Pc", "S", "Z"}
4630   local prefix_trees = {}
4631   for _, category in ipairs(categories) do
4632     prefix_trees[category] = {}
4633     for char_length = 1, 4 do
4634       prefix_trees[category][char_length] = {_type = "intermediate"}
4635     end
4636   end
4637   for line in file:lines() do
4638     local codepoint, full_category = line:match("^(%x+);[~;]*;(%a*)")
4639     assert(#full_category >= 1)
4640     local major_category = full_category:sub(1, 1)
4641     for _, category in ipairs({full_category, major_category}) do
4642       if prefix_trees[category] == nil then
4643         goto continue
4644       end
4645       local code = utf8.char(tonumber(codepoint, 16))
4646       local node = prefix_trees[category][#code]
4647       for i = 1, #code do
4648         local byte = code:sub(i, i)
4649         if i < #code then
4650           if node[byte] == nil then
4651             node[byte] = {_type = "intermediate"}
4652           end
4653           node = node[byte]
4654         else
4655           table.insert(node, {byte, _type = "leaf"})
4656         end
4657       end
4658       ::continue::
4659     end
4660   end
4661   assert(file:close())
```

Next, we will construct parsers out of the prefix trees.

```
4662   print("M.categories = {}")
```

```

4663 for _, category in ipairs(categories) do
4664     print("M.categories." .. category .. " = {}".format({}))
4665     for length, prefix_tree in pairs(prefix_trees[category]) do
4666         local subparsers = {}
4667         depth_first_search(prefix_tree, "", function(node, path)
4668             if node._type == "leaf" then
4669                 local byte = node[1]
4670                 local suffix = serialize_byte_parser(byte)
4671                 if subparsers[path] ~= nil then
4672                     subparsers[path] = subparsers[path] .. " + " .. suffix
4673                 else
4674                     subparsers[path] = suffix
4675                 end
4676             end
4677         end, function(_, path)
4678             if #path > 0 then
4679                 local byte = path:sub(#path, #path)
4680                 local parent_path = path:sub(1, #path-1)
4681                 local prefix = serialize_byte_parser(byte)
4682                 local suffix
4683                 if subparsers[path]:find(" %+ ") then
4684                     suffix = prefix .. " * (" .. subparsers[path] .. ")"
4685                 else
4686                     suffix = prefix .. " * " .. subparsers[path]
4687                 end
4688                 if subparsers[parent_path] ~= nil then
4689                     subparsers[parent_path] = subparsers[parent_path]
4690                         .. " + " .. suffix
4691                 else
4692                     subparsers[parent_path] = suffix
4693                 end
4694             else
4695                 print(
4696                     "M.categories." .. category .. "[" .. length .. "] = "
4697                     .. (subparsers[path] or "fail")
4698                 )
4699             end
4700         end)
4701     end
4702 end
4703 end()

```

### 3.1.1.6 Canonical Ordering Classes

Low-level parsers of UTF-8-encoded Unicode characters from different character classes [17, Section 3.11] are organized in table `unicode_data.ccc` based on the number of bytes they occupy after conversion to UTF-8.



First, let's read the file `UnicodeData.txt`.

```
4704 ;(function()
4705   local file = assert(io.open("UnicodeData.txt", "r"),
4706     [[Could not open file "UnicodeData.txt"]])
```

In order to minimize the size and speed of the parser, we will first construct prefix trees of UTF-8 encodings for all codepoints of a given Unicode combining class and code length.

```
4707   local prefix_trees = {}
4708   for char_length = 1, 4 do
4709     prefix_trees[char_length] = {_type = "intermediate"}
4710   end
4711   for line in file:lines() do
4712     local codepoint, combining_class
4713     = line:match("^(%x+);[~;]*;%a*;%d+")
4714     combining_class = tonumber(combining_class)
4715     if combining_class == 0 then
4716       goto continue
4717     end
4718     local code = utf8.char(tonumber(codepoint, 16))
4719     local node = prefix_trees[#code]
4720     for i = 1, #code do
4721       local byte = code:sub(i, i)
4722       if i < #code then
4723         if node[byte] == nil then
4724           node[byte] = {_type = "intermediate"}
4725         end
4726         node = node[byte]
4727       else
4728         table.insert(node, {byte, combining_class, _type = "leaf"})
4729       end
4730     end
4731     ::continue::
4732   end
4733   assert(file:close())
```

Next, we will construct parsers out of the prefix trees.

```
4734   print("M.ccc = {}")
4735   for length, prefix_tree in pairs(prefix_trees) do
4736     local subparsers = {}
4737     depth_first_search(prefix_tree, "", function(node, path)
4738       if node._type == "leaf" then
4739         local byte, combining_class = table.unpack(node)
4740         local suffix = serialize_byte_parser(byte)
4741         .. " * Cc(" .. tostring(combining_class) .. ")"
4742         if subparsers[path] ~= nil then
4743           subparsers[path] = subparsers[path] .. " + " .. suffix
```

```

4744         else
4745             subparsers[path] = suffix
4746         end
4747     end
4748 end, function(_, path)
4749     if #path > 0 then
4750         local byte = path:sub(#path, #path)
4751         local parent_path = path:sub(1, #path-1)
4752         local prefix = serialize_byte_parser(byte)
4753         local suffix
4754         if subparsers[path]:find(" %+ ") then
4755             suffix = prefix .. " * (" .. subparsers[path] .. ")"
4756         else
4757             suffix = prefix .. " * " .. subparsers[path]
4758         end
4759         if subparsers[parent_path] ~= nil then
4760             subparsers[parent_path] = subparsers[parent_path]
4761                                     .. " + " .. suffix
4762         else
4763             subparsers[parent_path] = suffix
4764         end
4765     else
4766         print(
4767             "M.ccc[" .. length .. "] = " .. (subparsers[path] or "fail")
4768         )
4769     end
4770 end)
4771 end
4772 end)()
4773 print("-- luacheck: pop")
4774 print("return M")

```

### 3.1.2 Utility Functions

This section documents the utility functions back in the file [markdown-parser.lua](#) used by the plain TeX writer and the markdown reader. These functions are encapsulated in the [util](#) object. The functions were originally located in the [lunamark/util.lua](#) file in the Lunamark Lua module.

```

4775 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.

```

4776 function util.err(msg, exit_code)
4777     io.stderr:write("markdown.lua: " .. msg .. "\n")
4778     os.exit(exit_code or 1)
4779 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.

```

4780 function util.cache(dir, string, salt, transform, suffix)
4781   local digest = md5.sumhexa(string .. (salt or ""))
4782   local name = util.pathname(dir, digest .. suffix)
4783   local file = io.open(name, "r")
4784   local result = nil
4785   if file == nil then -- If no cache entry exists, create a new one.
4786     file = assert(io.open(name, "w"),
4787       [[Could not open file ]] .. name .. [[ for writing]])
4788     result = string
4789     if transform ~= nil then
4790       result = transform(result)
4791     end
4792     assert(file:write(result))
4793     assert(file:close())
4794   end
4795   return name, result
4796 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.

```

4797 function util.cache_verbatim(dir, string)
4798   local name = util.cache(dir, string, nil, nil, ".verbatim")
4799   return name
4800 end

```

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

```

4801 function util.table_copy(t)
4802   local u = { }
4803   for k, v in pairs(t) do u[k] = v end
4804   return setmetatable(u, getmetatable(t))
4805 end

```

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

```

4806 function util.encode_json_string(s)
4807   s = s:gsub([[\\]], [[\\]])
4808   s = s:gsub([[\"]], [[\"]])
4809   return [[ ]] .. s .. [[ ]]
4810 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 [18, Chapter 21].

```
4811 function util.expand_tabs_in_line(s, tabstop)
4812   local tab = tabstop or 4
4813   local corr = 0
4814   return (s:gsub("()\t", function(p)
4815     local sp = tab - (p - 1 + corr) % tab
4816     corr = corr - 1 + sp
4817     return string.rep(" ", sp)
4818   end))
4819 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.

```
4820 function util.walk(t, f)
4821   local typ = type(t)
4822   if typ == "string" then
4823     f(t)
4824   elseif typ == "table" then
4825     local i = 1
4826     local n
4827     n = t[i]
4828     while n do
4829       util.walk(n, f)
4830       i = i + 1
4831       n = t[i]
4832     end
4833   elseif typ == "function" then
4834     local ok, val = pcall(t)
4835     if ok then
4836       util.walk(val, f)
4837     end
4838   else
4839     f(tostring(t))
4840   end
4841 end
```

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

```
4842 function util.flatten(ary)
4843   local new = {}
4844   for _,v in ipairs(ary) do
4845     if type(v) == "table" then
4846       for _,w in ipairs(util.flatten(v)) do
4847         new[#new + 1] = w
4848       end
4849     end
4850   end
4851   return new
4852 end
```

```

4848     end
4849     else
4850         new[#new + 1] = v
4851     end
4852 end
4853 return new
4854 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.

```

4855 function util.rope_to_string(rope)
4856     local buffer = {}
4857     util.walk(rope, function(x) buffer[#buffer + 1] = x end)
4858     return table.concat(buffer)
4859 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.

```

4860 function util.rope_last(rope)
4861     if #rope == 0 then
4862         return nil
4863     else
4864         local l = rope[#rope]
4865         if type(l) == "table" then
4866             return util.rope_last(l)
4867         else
4868             return l
4869         end
4870     end
4871 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  $1 \leq i \leq \#ary$ .

```

4872 function util.intersperse(ary, x)
4873     local new = {}
4874     local l = #ary
4875     for i,v in ipairs(ary) do
4876         local n = #new
4877         new[n + 1] = v
4878         if i ~= l then
4879             new[n + 2] = x
4880         end
4881     end
4882     return new
4883 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  $1 \leq i \leq \#ary$ .

```
4884 function util.map(ary, f)
4885   local new = {}
4886   for i,v in ipairs(ary) do
4887     new[i] = f(v)
4888   end
4889   return new
4890 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.

```
4891 function util.escaper(char_escapes, string_escapes)
```

Build a string of escapable characters.

```
4892   local char_escapes_list = ""
4893   for i,_ in pairs(char_escapes) do
4894     char_escapes_list = char_escapes_list .. i
4895   end
```

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

```
4896   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.

```
4897   if string_escapes then
4898     for k,v in pairs(string_escapes) do
4899       escapable = P(k) / v + escapable
4900     end
4901   end
```

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

```
4902   local escape_string = Cs((escapable + any)^0)
```

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

```

4903   return function(s)
4904       return lpeg.match(escape_string, s)
4905   end
4906 end

```

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

```

4907 function util.pathname(dir, file)
4908     if #dir == 0 then
4909         return file
4910     else
4911         return dir .. "/" .. file
4912     end
4913 end

```

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

```

4914 function util.salt(options)
4915     local opt_string = {}
4916     for k, _ in pairs(defaultOptions) do
4917         local v = options[k]
4918         if type(v) == "table" then
4919             for _, i in ipairs(v) do
4920                 opt_string[#opt_string+1] = k .. "=" .. tostring(i)
4921             end
4922         end
4923     end
4924 end

```

The `cacheDir` option is disregarded.

```

4922     elseif k ~= "cacheDir" then
4923         opt_string[#opt_string+1] = k .. "=" .. tostring(v)
4924     end
4925 end
4926 table.sort(opt_string)
4927 local salt = table.concat(opt_string, ",")
4928             .. "," .. metadata.version
4929 return salt
4930 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.

```

4931 util.warning = (function()
4932     local function warning(s)
4933         io.stderr:write("Warning: " .. s .. "\n")
4934     end
4935     for _, message in ipairs(early_warnings) do
4936         warning(message)
4937     end
4938     return warning
4939 end)()

```

The `util.casefold` method performs a full case-folding of a UTF-8-encoded Unicode string `s` based on the low-level parsers in `unicode_data.casefold_mapping`, defined in Section 3.1.1.4. Unlike the low-level parsers, the high-level function is invariant to the number of bytes the Unicode characters occupy after conversion to UTF-8.

```
4940 util.casefold = (function()
4941     local fail, any = P(false), P(1)
4942     local eof = -any
```

First, define a parser that will case-fold a character.

```
4943     local fold_character = fail
4944     for n = 1, 4 do
4945         fold_character
4946         = fold_character
4947         + unicode_data.casefold_mapping[n]
4948     end
4949     fold_character
4950     = fold_character
4951     + C(any)
```

Next, define a parser that will case-fold a string.

```
4952     local fold_string = Ct(fold_character^0) * eof
4953     return function(s, form)
4954         local result = table.concat(lpeg.match(fold_string, s))
4955         assert(result ~= nil)
```

For NFD and NFKD normalization forms, normalize the case-folded string and then repeat the fold-and-normalize operation.

```
4956         if form == "nfd" or form == "nfkd" then
4957             result = util.normalize(result, form)
4958             result = table.concat(lpeg.match(fold_string, result))
4959             assert(result ~= nil)
4960             result = util.normalize(result, form)
4961         end
4962         return result
4963     end
4964 end)()
```

The `util.canonically_order` method performs a Unicode canonical ordering of a string UTF-8-encoded Unicode `s` based on the low-level parsers in `unicode_data.ccc`, defined in Section 3.1.1.6. Unlike the low-level parsers, the high-level function is invariant to the number of bytes the Unicode characters occupy after conversion to UTF-8.

```
4965 util.canonically_order = (function()
4966     local fail, any = P(false), P(1)
4967     local eof = -any
4968     local cont = R("\128\191")
4969     local utf8_character
```



```

4970     = R("\0\127")
4971     + R("\194\223") * cont
4972     + R("\224\239") * cont * cont
4973     + R("\240\244") * cont * cont * cont

```

First, define a parser that will determine the combining class of a character.

```

4974     local classify_character = fail
4975     for n = 1, 4 do
4976         classify_character
4977         = classify_character
4978         + unicode_data.ccc[n]
4979     end
4980     classify_character
4981     = classify_character
4982     + utf8_character * Cc(0)

```

Next, define a parser that will determine the combining classes of all characters in a string.

```

4983     local classify_string = Ct(classify_character^0) * eof

```

When the function is called, first check whether the string is trivially ordered. If it is, return it without any changes.

```

4984     return function(s)
4985         local s_len = utf8.len(s)
4986         if s == false or s_len <= 1 then
4987             return s
4988         end

```

Otherwise, determine the combining classes of all characters in the string. If the string cannot be decoded with UTF-8, return it unchanged.

```

4989         local classes = lpeg.match(classify_string, s)
4990         if classes == nil then
4991             return s
4992         end
4993         assert(#classes == s_len)

```

Again, check whether the string is trivially ordered. If it is, return it without any changes. Otherwise, construct a list of ranges of non-starter characters that must be ordered.

```

4994         local non_starter_ranges = {}
4995         local first_non_starter, last_non_starter = nil, nil
4996         for i = 1, #classes do
4997             if first_non_starter == nil then
4998                 if classes[i] ~= 0 then
4999                     first_non_starter, last_non_starter = i, i
5000                 end
5001             else
5002                 if classes[i] == 0 then

```

```

5003         table.insert(
5004             non_starter_ranges,
5005             {first_non_starter, last_non_starter}
5006         )
5007         first_non_starter, last_non_starter = nil, nil
5008     else
5009         last_non_starter = i
5010     end
5011 end
5012 end
5013 if first_non_starter ~= nil then
5014     table.insert(
5015         non_starter_ranges,
5016         {first_non_starter, last_non_starter}
5017     )
5018 end
5019 if #non_starter_ranges == 0 then
5020     return s
5021 end
5022 local max_range_length = 0
5023 for _, range in ipairs(non_starter_ranges) do
5024     local range_start, range_end = table.unpack(range)
5025     local range_length = range_end - range_start + 1
5026     if range_length > max_range_length then
5027         max_range_length = range_length
5028     end
5029 end
5030 if max_range_length <= 1 then
5031     return s
5032 end

```

Then, construct a buffer of all characters in the string.

```

5033     local buffer = {}
5034     for _, code in utf8.codes(s) do
5035         local char = utf8.char(code)
5036         table.insert(buffer, char)
5037     end
5038     assert(#buffer == s_len)

```

Next, perform a local bubble sort over the ranges of non-starter characters.

```

5039     for _, range in ipairs(non_starter_ranges) do
5040         local range_start, range_end = table.unpack(range)
5041         local range_length = range_end - range_start + 1
5042         for _ = 1, range_length - 1 do
5043             local swapped = false
5044             for i = range_start, range_end - 1 do
5045                 local j = i + 1
5046                 if classes[i] > classes[j] then

```

```

5047         classes[i], classes[j] = classes[j], classes[i]
5048         buffer[i], buffer[j] = buffer[j], buffer[i]
5049         swapped = true
5050     end
5051 end
5052 if not swapped then
5053     break
5054 end
5055 end
5056 end

```

Finally, concatenate the buffer and return an ordered string.

```

5057     return table.concat(buffer, "")
5058 end
5059 end)()

```

The `util.decompose` method performs either the canonical or the compatibility decomposition of a UTF-8-encoded Unicode string `s` based on the low-level parsers in `unicode_data.decomposition_mapping`, defined in sections 3.1.1.1 and 3.1.1.2. Unlike the low-level parsers, the high-level function is invariant to the number of bytes the Unicode characters occupy after conversion to UTF-8.

```

5060 util.decompose = (function()
5061     local fail, any = P(false), P(1)
5062     local eof = -any
5063     local decomposition_types = {"canonical", "compatibility"}

```

First, define parsers that will decompose a character.

```

5064     local decompose_character = {}
5065     for _, decomposition_type in ipairs(decomposition_types) do
5066         decompose_character[decomposition_type] = fail
5067         for n = 1, 4 do
5068             decompose_character[decomposition_type]
5069                 = decompose_character[decomposition_type]
5070                 + unicode_data.decomposition_mapping[decomposition_type][n]
5071         end
5072         decompose_character[decomposition_type]
5073             = decompose_character[decomposition_type]
5074             + C(any)
5075     end
5076     local hangul = unicode_data.decomposition_mapping.hangul
5077     decompose_character.hangul = {}
5078     for syllable_type, _ in pairs(hangul) do
5079         decompose_character.hangul[syllable_type] = fail
5080         for n = 1, 4 do
5081             decompose_character.hangul[syllable_type]
5082                 = decompose_character.hangul[syllable_type]
5083                 + hangul[syllable_type][n]
5084         end

```

```

5085     decompose_character.hangul[syllable_type]
5086     = decompose_character.hangul[syllable_type]
5087     + C(any)
5088 end

```

Next, define a parser that will decompose a string.

```

5089 local decompose_string = {}
5090 for _, decomposition_type in ipairs(decomposition_types) do
5091     decompose_string[decomposition_type]
5092     = Ct(decompose_character[decomposition_type]^0) * eof
5093 end
5094 decompose_string.hangul = {}
5095 for syllable_type, _ in pairs(hangul) do
5096     decompose_string.hangul[syllable_type]
5097     = Ct(decompose_character.hangul[syllable_type]^0) * eof
5098 end
5099 local function _decompose(s, parser)
5100     assert(s ~= nil)
5101     local result = table.concat(lpeg.match(parser, s), "")
5102     assert(result ~= nil)
5103     return result
5104 end
5105 return function(s, decomposition_type)
5106     assert(
5107         decomposition_type == "canonical"
5108         or decomposition_type == "compatibility"
5109     )
5110     local prev_s
5111     local next_s = s
5112     repeat
5113         prev_s = next_s
5114         local function decompose(...) next_s = _decompose(next_s, ...) end
5115         decompose(decompose_string.canonical)
5116         if decomposition_type == "compatibility" then
5117             decompose(decompose_string.compatibility)
5118         end
5119         decompose(decompose_string.hangul.LVT)
5120         decompose(decompose_string.hangul.LV)
5121     until prev_s == next_s
5122     return util.canonically_order(next_s)
5123 end
5124 end)()

```

The `util.compose` method performs the canonical composition of a UTF-8-encoded canonically ordered Unicode string `s` based on the low-level parsers in `unicode_data.composition_mapping`, defined in Section 3.1.1.3, and definitions from the Hangul syllable (de)composition algorithm, defined in Section 3.1.1.2. Un-

like the low-level parsers, this high-level function is invariant to the number of bytes the Unicode characters occupy after conversion to UTF-8.

```
5125 util.compose = (function()
5126   local fail, any = P(false), P(1)
5127   local eof = -any
5128   local cont = R("\128\191")
5129   local utf8_character
5130     = R("\0\127")
5131     + R("\194\223") * cont
5132     + R("\224\239") * cont * cont
5133     + R("\240\244") * cont * cont * cont
```

First, define a parser that will determine the combining class of a character.

```
5134   local classify_character = fail
5135   for n = 1, 4 do
5136     classify_character
5137       = classify_character
5138       + unicode_data.ccc[n]
5139   end
5140   classify_character
5141     = classify_character
5142     + utf8_character * Cc(0)
```

Next, define a parser that will determine the combining classes of all characters in a string.

```
5143   local classify_string = Ct(classify_character^0) * eof
```

First, define parsers that will compose a pair of UTF-8-encoded Unicode characters into their primary composite.

```
5144   local compose_characters = fail
5145   for m = 1, 4 do
5146     local starter = #unicode_data.composition_mapping.starters[m]
5147     local both = fail
5148     for n = 1, 4 do
5149       both = (
5150         both
5151         + #unicode_data.composition_mapping.both[m][n]
5152         * unicode_data.composition_mapping.both[m][n]
5153       )
5154     end
5155     compose_characters = compose_characters + starter * both
5156   end
```

When the function is called, first check whether the string is trivially ordered. If it is, return it without any changes.

```
5157   return function(s)
5158     local s_len = utf8.len(s)
5159     if s == false or s_len <= 1 then
```

```

5160     return s
5161 end

```

Otherwise, determine the combining classes of all characters in the string. If the string cannot be decoded with UTF-8, return it unchanged.

```

5162     local classes = lpeg.match(classify_string, s)
5163     if classes == nil then
5164         return s
5165     end
5166     assert(#classes == s_len)

```

Otherwise, construct a buffer of all characters in the string.

```

5167     local buffer = {}
5168     for _, code in utf8.codes(s) do
5169         local char = utf8.char(code)
5170         table.insert(buffer, char)
5171     end
5172     assert(#buffer == s_len)

```

Finally, implement the composition algorithm.

First, find the first starter character in the string.

```

5173     local starter = 1
5174     while starter <= s_len and classes[starter] ~= 0 do
5175         starter = starter + 1
5176     end
5177     local candidate_combining_mark = starter + 1

```

Next, apply the composition rules until we reach the end of the string.

```

5178     while candidate_combining_mark <= s_len do
5179         local L = buffer[starter]
5180         local C = buffer[candidate_combining_mark]
5181         local P = lpeg.match(compose_characters, L .. C)
5182         if P ~= nil then
5183             buffer[starter] = P
5184             buffer[candidate_combining_mark] = ""
5185         else
5186             if classes[candidate_combining_mark] == 0 then
5187                 starter = candidate_combining_mark
5188             end
5189             candidate_combining_mark = candidate_combining_mark + 1
5190         end
5191         assert(starter <= s_len)
5192     end

```

Next, iterate over the string once more and compose Hangul syllables.

```

5193     for i = 1, s_len - 1 do
5194         local last, ch = buffer[i], buffer[i + 1]
5195         if last ~= "" and ch ~= "" then
5196             local composite = unicode_data.hangul_compose(last, ch)

```

```

5197         if composite ~= nil then
5198             buffer[i] = ""
5199             buffer[i + 1] = composite
5200         end
5201     end
5202 end

```

Finally, concatenate the buffer and return an ordered string.

```

5203     return table.concat(buffer, "")
5204 end
5205 end)()

```

The `util.normalize` method normalizes a UTF-8-encoded canonically ordered Unicode string `s` using the normalization form `form`.

```

5206 function util.normalize(s, form)
5207     if form == "nfd" then
5208         return util.decompose(s, "canonical")
5209     elseif form == "nfkd" then
5210         return util.decompose(s, "compatibility")
5211     elseif form == "nfc" then
5212         return util.compose(util.decompose(s, "canonical"))
5213     elseif form == "nfkc" then
5214         return util.compose(util.decompose(s, "compatibility"))
5215     else
5216         error(string.format('Unexpected normal form "%s"', form))
5217     end
5218 end

```

The `util.find_file` and `util.find_files` method find the first or all locations of a file, respectively, according to either the resolvers API [1, Section 11.5] from the ConTeXt format or the Kpathsea library.

```

5219 function util.find_file(filename)
5220     if resolvers ~= nil then
5221         return resolvers.findfile(filename)
5222     else
5223         return kpse.find_file(filename)
5224     end
5225 end
5226 function util.find_files(filename)
5227     if resolvers ~= nil then
5228         return resolvers.findfiles(filename)
5229     else
5230         return {kpse.lookup(filename, {all=true})}
5231     end
5232 end

```

### 3.1.3 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.

```
5233 local entities = {}
5234
5235 local character_entities = {
5236     ["Tab"] = 9,
5237     ["NewLine"] = 10,
5238     ["excl"] = 33,
5239     ["QUOT"] = 34,
5240     ["quot"] = 34,
5241     ["num"] = 35,
5242     ["dollar"] = 36,
5243     ["percent"] = 37,
5244     ["AMP"] = 38,
5245     ["amp"] = 38,
5246     ["apos"] = 39,
5247     ["lpar"] = 40,
5248     ["rpar"] = 41,
5249     ["ast"] = 42,
5250     ["midast"] = 42,
5251     ["plus"] = 43,
5252     ["comma"] = 44,
5253     ["period"] = 46,
5254     ["sol"] = 47,
5255     ["colon"] = 58,
5256     ["semi"] = 59,
5257     ["LT"] = 60,
5258     ["lt"] = 60,
5259     ["nvlt"] = {60, 8402},
5260     ["bne"] = {61, 8421},
5261     ["equals"] = 61,
5262     ["GT"] = 62,
5263     ["gt"] = 62,
5264     ["nvgt"] = {62, 8402},
5265     ["quest"] = 63,
5266     ["commat"] = 64,
5267     ["lbrack"] = 91,
5268     ["lsqb"] = 91,
5269     ["bsol"] = 92,
5270     ["rbrack"] = 93,
5271     ["rsqb"] = 93,
5272     ["Hat"] = 94,
5273     ["UnderBar"] = 95,
5274     ["lowbar"] = 95,
```



```

5275 ["DiacriticalGrave"] = 96,
5276 ["grave"] = 96,
5277 ["fjlig"] = {102, 106},
5278 ["lbrace"] = 123,
5279 ["lcub"] = 123,
5280 ["VerticalLine"] = 124,
5281 ["verbar"] = 124,
5282 ["vert"] = 124,
5283 ["rbrace"] = 125,
5284 ["rcub"] = 125,
5285 ["NonBreakingSpace"] = 160,
5286 ["nbsp"] = 160,
5287 ["iexcl"] = 161,
5288 ["cent"] = 162,
5289 ["pound"] = 163,
5290 ["curren"] = 164,
5291 ["yen"] = 165,
5292 ["brvbar"] = 166,
5293 ["sect"] = 167,
5294 ["Dot"] = 168,
5295 ["DoubleDot"] = 168,
5296 ["die"] = 168,
5297 ["uml"] = 168,
5298 ["COPY"] = 169,
5299 ["copy"] = 169,
5300 ["ordf"] = 170,
5301 ["laquo"] = 171,
5302 ["not"] = 172,
5303 ["shy"] = 173,
5304 ["REG"] = 174,
5305 ["circledR"] = 174,
5306 ["reg"] = 174,
5307 ["macr"] = 175,
5308 ["strns"] = 175,
5309 ["deg"] = 176,
5310 ["PlusMinus"] = 177,
5311 ["plusmn"] = 177,
5312 ["pm"] = 177,
5313 ["sup2"] = 178,
5314 ["sup3"] = 179,
5315 ["DiacriticalAcute"] = 180,
5316 ["acute"] = 180,
5317 ["micro"] = 181,
5318 ["para"] = 182,
5319 ["CenterDot"] = 183,
5320 ["centerdot"] = 183,
5321 ["middot"] = 183,

```

5322 ["Cedilla"] = 184,  
 5323 ["cedil"] = 184,  
 5324 ["sup1"] = 185,  
 5325 ["ordm"] = 186,  
 5326 ["raquo"] = 187,  
 5327 ["frac14"] = 188,  
 5328 ["frac12"] = 189,  
 5329 ["half"] = 189,  
 5330 ["frac34"] = 190,  
 5331 ["iquest"] = 191,  
 5332 ["Agrave"] = 192,  
 5333 ["Aacute"] = 193,  
 5334 ["Acirc"] = 194,  
 5335 ["Atilde"] = 195,  
 5336 ["Auml"] = 196,  
 5337 ["Aring"] = 197,  
 5338 ["angst"] = 197,  
 5339 ["AElig"] = 198,  
 5340 ["Ccedil"] = 199,  
 5341 ["Egrave"] = 200,  
 5342 ["Eacute"] = 201,  
 5343 ["Ecirc"] = 202,  
 5344 ["Euml"] = 203,  
 5345 ["Igrave"] = 204,  
 5346 ["Iacute"] = 205,  
 5347 ["Icirc"] = 206,  
 5348 ["Iuml"] = 207,  
 5349 ["ETH"] = 208,  
 5350 ["Ntilde"] = 209,  
 5351 ["Ograve"] = 210,  
 5352 ["Oacute"] = 211,  
 5353 ["Ocirc"] = 212,  
 5354 ["Otilde"] = 213,  
 5355 ["Ouml"] = 214,  
 5356 ["times"] = 215,  
 5357 ["Oslash"] = 216,  
 5358 ["Ugrave"] = 217,  
 5359 ["Uacute"] = 218,  
 5360 ["Ucirc"] = 219,  
 5361 ["Uuml"] = 220,  
 5362 ["Yacute"] = 221,  
 5363 ["THORN"] = 222,  
 5364 ["szlig"] = 223,  
 5365 ["agrave"] = 224,  
 5366 ["aacute"] = 225,  
 5367 ["acirc"] = 226,  
 5368 ["atilde"] = 227,

5369 ["auml"] = 228,  
 5370 ["aring"] = 229,  
 5371 ["aelig"] = 230,  
 5372 ["ccedil"] = 231,  
 5373 ["egrave"] = 232,  
 5374 ["eacute"] = 233,  
 5375 ["ecirc"] = 234,  
 5376 ["euml"] = 235,  
 5377 ["igrave"] = 236,  
 5378 ["iacute"] = 237,  
 5379 ["icirc"] = 238,  
 5380 ["iuml"] = 239,  
 5381 ["eth"] = 240,  
 5382 ["ntilde"] = 241,  
 5383 ["ograve"] = 242,  
 5384 ["oacute"] = 243,  
 5385 ["ocirc"] = 244,  
 5386 ["otilde"] = 245,  
 5387 ["ouml"] = 246,  
 5388 ["div"] = 247,  
 5389 ["divide"] = 247,  
 5390 ["oslash"] = 248,  
 5391 ["ugrave"] = 249,  
 5392 ["uacute"] = 250,  
 5393 ["ucirc"] = 251,  
 5394 ["uuml"] = 252,  
 5395 ["yacute"] = 253,  
 5396 ["thorn"] = 254,  
 5397 ["yuml"] = 255,  
 5398 ["Amacr"] = 256,  
 5399 ["amacr"] = 257,  
 5400 ["Abreve"] = 258,  
 5401 ["abreve"] = 259,  
 5402 ["Aogon"] = 260,  
 5403 ["aogon"] = 261,  
 5404 ["Cacute"] = 262,  
 5405 ["cacute"] = 263,  
 5406 ["Ccirc"] = 264,  
 5407 ["ccirc"] = 265,  
 5408 ["Cdot"] = 266,  
 5409 ["cdot"] = 267,  
 5410 ["Ccaron"] = 268,  
 5411 ["ccaron"] = 269,  
 5412 ["Dcaron"] = 270,  
 5413 ["dcaron"] = 271,  
 5414 ["Dstrok"] = 272,  
 5415 ["dstrok"] = 273,

```

5416 ["Emacr"] = 274,
5417 ["emacr"] = 275,
5418 ["Edot"] = 278,
5419 ["edot"] = 279,
5420 ["Eogon"] = 280,
5421 ["eogon"] = 281,
5422 ["Ecaron"] = 282,
5423 ["ecaron"] = 283,
5424 ["Gcirc"] = 284,
5425 ["gcirc"] = 285,
5426 ["Gbreve"] = 286,
5427 ["gbreve"] = 287,
5428 ["Gdot"] = 288,
5429 ["gdot"] = 289,
5430 ["Gcedil"] = 290,
5431 ["Hcirc"] = 292,
5432 ["hcirc"] = 293,
5433 ["Hstrook"] = 294,
5434 ["hstrook"] = 295,
5435 ["Itilde"] = 296,
5436 ["itilde"] = 297,
5437 ["Imacr"] = 298,
5438 ["imacr"] = 299,
5439 ["Iogon"] = 302,
5440 ["iogon"] = 303,
5441 ["Idot"] = 304,
5442 ["imath"] = 305,
5443 ["inodot"] = 305,
5444 ["IJlig"] = 306,
5445 ["ijlig"] = 307,
5446 ["Jcirc"] = 308,
5447 ["jcirc"] = 309,
5448 ["Kcedil"] = 310,
5449 ["kcedil"] = 311,
5450 ["kgreen"] = 312,
5451 ["Lacute"] = 313,
5452 ["lacute"] = 314,
5453 ["Lcedil"] = 315,
5454 ["lcedil"] = 316,
5455 ["Lcaron"] = 317,
5456 ["lcaron"] = 318,
5457 ["Lmidot"] = 319,
5458 ["lmidot"] = 320,
5459 ["Lstrook"] = 321,
5460 ["lstrook"] = 322,
5461 ["Nacute"] = 323,
5462 ["nacute"] = 324,

```

5463 ["Ncedil"] = 325,  
 5464 ["ncedil"] = 326,  
 5465 ["Ncaron"] = 327,  
 5466 ["ncaron"] = 328,  
 5467 ["napos"] = 329,  
 5468 ["ENG"] = 330,  
 5469 ["eng"] = 331,  
 5470 ["Omacr"] = 332,  
 5471 ["omacr"] = 333,  
 5472 ["Odblac"] = 336,  
 5473 ["odblac"] = 337,  
 5474 ["OElig"] = 338,  
 5475 ["oelig"] = 339,  
 5476 ["Racute"] = 340,  
 5477 ["racute"] = 341,  
 5478 ["Rcedil"] = 342,  
 5479 ["rcedil"] = 343,  
 5480 ["Rcaron"] = 344,  
 5481 ["rcaron"] = 345,  
 5482 ["Sacute"] = 346,  
 5483 ["sacute"] = 347,  
 5484 ["Scirc"] = 348,  
 5485 ["scirc"] = 349,  
 5486 ["Scedil"] = 350,  
 5487 ["scedil"] = 351,  
 5488 ["Scaron"] = 352,  
 5489 ["scaron"] = 353,  
 5490 ["Tcedil"] = 354,  
 5491 ["tcedil"] = 355,  
 5492 ["Tcaron"] = 356,  
 5493 ["tcaron"] = 357,  
 5494 ["Tstrok"] = 358,  
 5495 ["tstrok"] = 359,  
 5496 ["Utilde"] = 360,  
 5497 ["utilde"] = 361,  
 5498 ["Umacr"] = 362,  
 5499 ["umacr"] = 363,  
 5500 ["Ubreve"] = 364,  
 5501 ["ubreve"] = 365,  
 5502 ["Uring"] = 366,  
 5503 ["uring"] = 367,  
 5504 ["Udblac"] = 368,  
 5505 ["udblac"] = 369,  
 5506 ["Uogon"] = 370,  
 5507 ["uogon"] = 371,  
 5508 ["Wcirc"] = 372,  
 5509 ["wcirc"] = 373,

5510 ["Ycirc"] = 374,  
 5511 ["ycirc"] = 375,  
 5512 ["Yuml"] = 376,  
 5513 ["Zacute"] = 377,  
 5514 ["zacute"] = 378,  
 5515 ["Zdot"] = 379,  
 5516 ["zdot"] = 380,  
 5517 ["Zcaron"] = 381,  
 5518 ["zcaron"] = 382,  
 5519 ["fnof"] = 402,  
 5520 ["imped"] = 437,  
 5521 ["gacute"] = 501,  
 5522 ["jmath"] = 567,  
 5523 ["circ"] = 710,  
 5524 ["Hacek"] = 711,  
 5525 ["caron"] = 711,  
 5526 ["Breve"] = 728,  
 5527 ["breve"] = 728,  
 5528 ["DiacriticalDot"] = 729,  
 5529 ["dot"] = 729,  
 5530 ["ring"] = 730,  
 5531 ["ogon"] = 731,  
 5532 ["DiacriticalTilde"] = 732,  
 5533 ["tilde"] = 732,  
 5534 ["DiacriticalDoubleAcute"] = 733,  
 5535 ["dblac"] = 733,  
 5536 ["DownBreve"] = 785,  
 5537 ["Alpha"] = 913,  
 5538 ["Beta"] = 914,  
 5539 ["Gamma"] = 915,  
 5540 ["Delta"] = 916,  
 5541 ["Epsilon"] = 917,  
 5542 ["Zeta"] = 918,  
 5543 ["Eta"] = 919,  
 5544 ["Theta"] = 920,  
 5545 ["Iota"] = 921,  
 5546 ["Kappa"] = 922,  
 5547 ["Lambda"] = 923,  
 5548 ["Mu"] = 924,  
 5549 ["Nu"] = 925,  
 5550 ["Xi"] = 926,  
 5551 ["Omicron"] = 927,  
 5552 ["Pi"] = 928,  
 5553 ["Rho"] = 929,  
 5554 ["Sigma"] = 931,  
 5555 ["Tau"] = 932,  
 5556 ["Upsilon"] = 933,

```

5557 ["Phi"] = 934,
5558 ["Chi"] = 935,
5559 ["Psi"] = 936,
5560 ["Omega"] = 937,
5561 ["ohm"] = 937,
5562 ["alpha"] = 945,
5563 ["beta"] = 946,
5564 ["gamma"] = 947,
5565 ["delta"] = 948,
5566 ["epsi"] = 949,
5567 ["epsilon"] = 949,
5568 ["zeta"] = 950,
5569 ["eta"] = 951,
5570 ["theta"] = 952,
5571 ["iota"] = 953,
5572 ["kappa"] = 954,
5573 ["lambda"] = 955,
5574 ["mu"] = 956,
5575 ["nu"] = 957,
5576 ["xi"] = 958,
5577 ["omicron"] = 959,
5578 ["pi"] = 960,
5579 ["rho"] = 961,
5580 ["sigmaf"] = 962,
5581 ["sigmav"] = 962,
5582 ["varsigma"] = 962,
5583 ["sigma"] = 963,
5584 ["tau"] = 964,
5585 ["upsi"] = 965,
5586 ["upsilon"] = 965,
5587 ["phi"] = 966,
5588 ["chi"] = 967,
5589 ["psi"] = 968,
5590 ["omega"] = 969,
5591 ["thetasym"] = 977,
5592 ["thetav"] = 977,
5593 ["vartheta"] = 977,
5594 ["Upsi"] = 978,
5595 ["upsih"] = 978,
5596 ["phiv"] = 981,
5597 ["straightphi"] = 981,
5598 ["varphi"] = 981,
5599 ["piv"] = 982,
5600 ["varpi"] = 982,
5601 ["Gammad"] = 988,
5602 ["digamma"] = 989,
5603 ["gammad"] = 989,

```

```

5604 ["kappav"] = 1008,
5605 ["varkappa"] = 1008,
5606 ["rhov"] = 1009,
5607 ["varrho"] = 1009,
5608 ["epsiv"] = 1013,
5609 ["straightepsilon"] = 1013,
5610 ["varepsilon"] = 1013,
5611 ["backepsilon"] = 1014,
5612 ["bepsi"] = 1014,
5613 ["IOcy"] = 1025,
5614 ["DJcy"] = 1026,
5615 ["GJcy"] = 1027,
5616 ["Jukcy"] = 1028,
5617 ["DScy"] = 1029,
5618 ["Iukcy"] = 1030,
5619 ["YIcy"] = 1031,
5620 ["Jsercy"] = 1032,
5621 ["LJcy"] = 1033,
5622 ["NJcy"] = 1034,
5623 ["TSHcy"] = 1035,
5624 ["KJcy"] = 1036,
5625 ["Ubrcy"] = 1038,
5626 ["DZcy"] = 1039,
5627 ["Acy"] = 1040,
5628 ["Bcy"] = 1041,
5629 ["Vcy"] = 1042,
5630 ["Gcy"] = 1043,
5631 ["Dcy"] = 1044,
5632 ["IEcy"] = 1045,
5633 ["ZHcy"] = 1046,
5634 ["Zcy"] = 1047,
5635 ["Icy"] = 1048,
5636 ["Jcy"] = 1049,
5637 ["Kcy"] = 1050,
5638 ["Lcy"] = 1051,
5639 ["Mcy"] = 1052,
5640 ["Ncy"] = 1053,
5641 ["Ocy"] = 1054,
5642 ["Pcy"] = 1055,
5643 ["Rcy"] = 1056,
5644 ["Scy"] = 1057,
5645 ["Tcy"] = 1058,
5646 ["Ucy"] = 1059,
5647 ["Fcy"] = 1060,
5648 ["KHcy"] = 1061,
5649 ["TScy"] = 1062,
5650 ["CHcy"] = 1063,

```



```

5651 ["SHcy"] = 1064,
5652 ["SHCHcy"] = 1065,
5653 ["HARDcy"] = 1066,
5654 ["Ycy"] = 1067,
5655 ["SOFTcy"] = 1068,
5656 ["Ecy"] = 1069,
5657 ["YUcy"] = 1070,
5658 ["YAcy"] = 1071,
5659 ["acy"] = 1072,
5660 ["bcy"] = 1073,
5661 ["vcy"] = 1074,
5662 ["gcy"] = 1075,
5663 ["dcy"] = 1076,
5664 ["iecy"] = 1077,
5665 ["zhcy"] = 1078,
5666 ["zcy"] = 1079,
5667 ["icy"] = 1080,
5668 ["jcy"] = 1081,
5669 ["kcy"] = 1082,
5670 ["lcy"] = 1083,
5671 ["mcy"] = 1084,
5672 ["ncy"] = 1085,
5673 ["ocy"] = 1086,
5674 ["pcy"] = 1087,
5675 ["rcy"] = 1088,
5676 ["scy"] = 1089,
5677 ["tcy"] = 1090,
5678 ["ucy"] = 1091,
5679 ["fcy"] = 1092,
5680 ["khcy"] = 1093,
5681 ["tscy"] = 1094,
5682 ["chcy"] = 1095,
5683 ["shcy"] = 1096,
5684 ["shchcy"] = 1097,
5685 ["hardcy"] = 1098,
5686 ["ycy"] = 1099,
5687 ["softcy"] = 1100,
5688 ["ecy"] = 1101,
5689 ["yucy"] = 1102,
5690 ["yacy"] = 1103,
5691 ["iocy"] = 1105,
5692 ["djcy"] = 1106,
5693 ["gjcy"] = 1107,
5694 ["jukcy"] = 1108,
5695 ["dscy"] = 1109,
5696 ["iukcy"] = 1110,
5697 ["yicy"] = 1111,

```

```

5698 ["jsercy"] = 1112,
5699 ["ljcy"] = 1113,
5700 ["njcy"] = 1114,
5701 ["tshcy"] = 1115,
5702 ["kjcy"] = 1116,
5703 ["ubrcy"] = 1118,
5704 ["dzcyc"] = 1119,
5705 ["ensp"] = 8194,
5706 ["emsp"] = 8195,
5707 ["emsp13"] = 8196,
5708 ["emsp14"] = 8197,
5709 ["numsp"] = 8199,
5710 ["puncsp"] = 8200,
5711 ["ThinSpace"] = 8201,
5712 ["thinsp"] = 8201,
5713 ["VeryThinSpace"] = 8202,
5714 ["hairsp"] = 8202,
5715 ["NegativeMediumSpace"] = 8203,
5716 ["NegativeThickSpace"] = 8203,
5717 ["NegativeThinSpace"] = 8203,
5718 ["NegativeVeryThinSpace"] = 8203,
5719 ["ZeroWidthSpace"] = 8203,
5720 ["zwnj"] = 8204,
5721 ["zwj"] = 8205,
5722 ["lrm"] = 8206,
5723 ["rlm"] = 8207,
5724 ["dash"] = 8208,
5725 ["hyphen"] = 8208,
5726 ["ndash"] = 8211,
5727 ["mdash"] = 8212,
5728 ["horbar"] = 8213,
5729 ["Verbar"] = 8214,
5730 ["Vert"] = 8214,
5731 ["OpenCurlyQuote"] = 8216,
5732 ["lsquo"] = 8216,
5733 ["CloseCurlyQuote"] = 8217,
5734 ["rsquo"] = 8217,
5735 ["rsquor"] = 8217,
5736 ["lsquor"] = 8218,
5737 ["sbquo"] = 8218,
5738 ["OpenCurlyDoubleQuote"] = 8220,
5739 ["ldquo"] = 8220,
5740 ["CloseCurlyDoubleQuote"] = 8221,
5741 ["rdquo"] = 8221,
5742 ["rdquor"] = 8221,
5743 ["bdquo"] = 8222,
5744 ["ldquor"] = 8222,

```

```

5745 ["dagger"] = 8224,
5746 ["Dagger"] = 8225,
5747 ["ddagger"] = 8225,
5748 ["bull"] = 8226,
5749 ["bullet"] = 8226,
5750 ["nldr"] = 8229,
5751 ["hellip"] = 8230,
5752 ["mldr"] = 8230,
5753 ["permil"] = 8240,
5754 ["pertenk"] = 8241,
5755 ["prime"] = 8242,
5756 ["Prime"] = 8243,
5757 ["tprime"] = 8244,
5758 ["backprime"] = 8245,
5759 ["bprime"] = 8245,
5760 ["lsaquo"] = 8249,
5761 ["rsaquo"] = 8250,
5762 ["OverBar"] = 8254,
5763 ["oline"] = 8254,
5764 ["caret"] = 8257,
5765 ["hybull"] = 8259,
5766 ["frasl"] = 8260,
5767 ["bsemi"] = 8271,
5768 ["qprime"] = 8279,
5769 ["MediumSpace"] = 8287,
5770 ["ThickSpace"] = {8287, 8202},
5771 ["NoBreak"] = 8288,
5772 ["ApplyFunction"] = 8289,
5773 ["af"] = 8289,
5774 ["InvisibleTimes"] = 8290,
5775 ["it"] = 8290,
5776 ["InvisibleComma"] = 8291,
5777 ["ic"] = 8291,
5778 ["euro"] = 8364,
5779 ["TripleDot"] = 8411,
5780 ["tdot"] = 8411,
5781 ["DotDot"] = 8412,
5782 ["Copf"] = 8450,
5783 ["complexes"] = 8450,
5784 ["incare"] = 8453,
5785 ["gscr"] = 8458,
5786 ["HilbertSpace"] = 8459,
5787 ["Hscr"] = 8459,
5788 ["hamilt"] = 8459,
5789 ["Hfr"] = 8460,
5790 ["Poincareplane"] = 8460,
5791 ["Hopf"] = 8461,

```

```

5792 ["quaternions"] = 8461,
5793 ["planckh"] = 8462,
5794 ["hbar"] = 8463,
5795 ["hslash"] = 8463,
5796 ["planck"] = 8463,
5797 ["plankv"] = 8463,
5798 ["Iscr"] = 8464,
5799 ["imagline"] = 8464,
5800 ["Ifr"] = 8465,
5801 ["Im"] = 8465,
5802 ["image"] = 8465,
5803 ["imagpart"] = 8465,
5804 ["Laplacetrfr"] = 8466,
5805 ["Lscr"] = 8466,
5806 ["lagran"] = 8466,
5807 ["ell"] = 8467,
5808 ["Nopf"] = 8469,
5809 ["naturals"] = 8469,
5810 ["numero"] = 8470,
5811 ["copysr"] = 8471,
5812 ["weierp"] = 8472,
5813 ["wp"] = 8472,
5814 ["Popf"] = 8473,
5815 ["primes"] = 8473,
5816 ["Qopf"] = 8474,
5817 ["rationals"] = 8474,
5818 ["Rscr"] = 8475,
5819 ["realine"] = 8475,
5820 ["Re"] = 8476,
5821 ["Rfr"] = 8476,
5822 ["real"] = 8476,
5823 ["realpart"] = 8476,
5824 ["Ropf"] = 8477,
5825 ["reals"] = 8477,
5826 ["rx"] = 8478,
5827 ["TRADE"] = 8482,
5828 ["trade"] = 8482,
5829 ["Zopf"] = 8484,
5830 ["integers"] = 8484,
5831 ["mho"] = 8487,
5832 ["Zfr"] = 8488,
5833 ["zeetrf"] = 8488,
5834 ["iiota"] = 8489,
5835 ["Bernoullis"] = 8492,
5836 ["Bscr"] = 8492,
5837 ["bernou"] = 8492,
5838 ["Cayleys"] = 8493,

```

```

5839 ["Cfr"] = 8493,
5840 ["escr"] = 8495,
5841 ["Escr"] = 8496,
5842 ["expectation"] = 8496,
5843 ["Fouriertrf"] = 8497,
5844 ["Fscr"] = 8497,
5845 ["Mellintrf"] = 8499,
5846 ["Mscr"] = 8499,
5847 ["phmmat"] = 8499,
5848 ["order"] = 8500,
5849 ["orderof"] = 8500,
5850 ["oscr"] = 8500,
5851 ["alefsym"] = 8501,
5852 ["aleph"] = 8501,
5853 ["beth"] = 8502,
5854 ["gimel"] = 8503,
5855 ["daleth"] = 8504,
5856 ["CapitalDifferentialD"] = 8517,
5857 ["DD"] = 8517,
5858 ["DifferentialD"] = 8518,
5859 ["dd"] = 8518,
5860 ["ExponentialE"] = 8519,
5861 ["ee"] = 8519,
5862 ["exponentiale"] = 8519,
5863 ["ImaginaryI"] = 8520,
5864 ["ii"] = 8520,
5865 ["frac13"] = 8531,
5866 ["frac23"] = 8532,
5867 ["frac15"] = 8533,
5868 ["frac25"] = 8534,
5869 ["frac35"] = 8535,
5870 ["frac45"] = 8536,
5871 ["frac16"] = 8537,
5872 ["frac56"] = 8538,
5873 ["frac18"] = 8539,
5874 ["frac38"] = 8540,
5875 ["frac58"] = 8541,
5876 ["frac78"] = 8542,
5877 ["LeftArrow"] = 8592,
5878 ["ShortLeftArrow"] = 8592,
5879 ["larr"] = 8592,
5880 ["leftarrow"] = 8592,
5881 ["slarr"] = 8592,
5882 ["ShortUpArrow"] = 8593,
5883 ["UpArrow"] = 8593,
5884 ["uarr"] = 8593,
5885 ["uparrow"] = 8593,

```

```

5886 ["RightArrow"] = 8594,
5887 ["ShortRightArrow"] = 8594,
5888 ["rarr"] = 8594,
5889 ["rightarrow"] = 8594,
5890 ["srarr"] = 8594,
5891 ["DownArrow"] = 8595,
5892 ["ShortDownArrow"] = 8595,
5893 ["darr"] = 8595,
5894 ["downarrow"] = 8595,
5895 ["LeftRightArrow"] = 8596,
5896 ["harr"] = 8596,
5897 ["leftrightarrow"] = 8596,
5898 ["UpDownArrow"] = 8597,
5899 ["updownarrow"] = 8597,
5900 ["varr"] = 8597,
5901 ["UpperLeftArrow"] = 8598,
5902 ["nwarr"] = 8598,
5903 ["nwarrow"] = 8598,
5904 ["UpperRightArrow"] = 8599,
5905 ["nearr"] = 8599,
5906 ["nearrow"] = 8599,
5907 ["LowerRightArrow"] = 8600,
5908 ["searr"] = 8600,
5909 ["searrow"] = 8600,
5910 ["LowerLeftArrow"] = 8601,
5911 ["swarr"] = 8601,
5912 ["swarrow"] = 8601,
5913 ["nlarr"] = 8602,
5914 ["nleftarrow"] = 8602,
5915 ["nrarr"] = 8603,
5916 ["nrightarrow"] = 8603,
5917 ["nrarrw"] = {8605, 824},
5918 ["rarrw"] = 8605,
5919 ["rightsquigarrow"] = 8605,
5920 ["Larr"] = 8606,
5921 ["twoheadleftarrow"] = 8606,
5922 ["Uarr"] = 8607,
5923 ["Rarr"] = 8608,
5924 ["twoheadrightarrow"] = 8608,
5925 ["Darr"] = 8609,
5926 ["larrtl"] = 8610,
5927 ["leftarrowtail"] = 8610,
5928 ["rarrtl"] = 8611,
5929 ["rightarrowtail"] = 8611,
5930 ["LeftTeeArrow"] = 8612,
5931 ["mapstoleft"] = 8612,
5932 ["UpTeeArrow"] = 8613,

```

```

5933 ["mapstoup"] = 8613,
5934 ["RightTeeArrow"] = 8614,
5935 ["map"] = 8614,
5936 ["mapsto"] = 8614,
5937 ["DownTeeArrow"] = 8615,
5938 ["mapstodown"] = 8615,
5939 ["hookleftarrow"] = 8617,
5940 ["larrhk"] = 8617,
5941 ["hookrightarrow"] = 8618,
5942 ["rarrhk"] = 8618,
5943 ["larrlp"] = 8619,
5944 ["looparrowleft"] = 8619,
5945 ["looparrowright"] = 8620,
5946 ["rarrlp"] = 8620,
5947 ["harrw"] = 8621,
5948 ["leftrightsquigarrow"] = 8621,
5949 ["nharr"] = 8622,
5950 ["nleftrightarrow"] = 8622,
5951 ["Lsh"] = 8624,
5952 ["lsh"] = 8624,
5953 ["Rsh"] = 8625,
5954 ["rsh"] = 8625,
5955 ["ldsh"] = 8626,
5956 ["rdsh"] = 8627,
5957 ["crarr"] = 8629,
5958 ["cularr"] = 8630,
5959 ["curvearrowleft"] = 8630,
5960 ["curarr"] = 8631,
5961 ["curvearrowright"] = 8631,
5962 ["circlearrowleft"] = 8634,
5963 ["olarr"] = 8634,
5964 ["circlearrowright"] = 8635,
5965 ["orarr"] = 8635,
5966 ["LeftVector"] = 8636,
5967 ["leftharpoonup"] = 8636,
5968 ["lharu"] = 8636,
5969 ["DownLeftVector"] = 8637,
5970 ["leftharpoondown"] = 8637,
5971 ["lhard"] = 8637,
5972 ["RightUpVector"] = 8638,
5973 ["uharr"] = 8638,
5974 ["upharpoonright"] = 8638,
5975 ["LeftUpVector"] = 8639,
5976 ["uharl"] = 8639,
5977 ["upharpoonleft"] = 8639,
5978 ["RightVector"] = 8640,
5979 ["rharu"] = 8640,

```

```

5980 ["rightharpoonup"] = 8640,
5981 ["DownRightVector"] = 8641,
5982 ["rhard"] = 8641,
5983 ["rightharpoondown"] = 8641,
5984 ["RightDownVector"] = 8642,
5985 ["dharr"] = 8642,
5986 ["downharpoonright"] = 8642,
5987 ["LeftDownVector"] = 8643,
5988 ["dharl"] = 8643,
5989 ["downharpoonleft"] = 8643,
5990 ["RightArrowLeftArrow"] = 8644,
5991 ["rightleftarrows"] = 8644,
5992 ["rlarr"] = 8644,
5993 ["UpArrowDownArrow"] = 8645,
5994 ["udarr"] = 8645,
5995 ["LeftArrowRightArrow"] = 8646,
5996 ["leftrightarrows"] = 8646,
5997 ["lrarr"] = 8646,
5998 ["leftleftarrows"] = 8647,
5999 ["llarr"] = 8647,
6000 ["upuparrows"] = 8648,
6001 ["uuarr"] = 8648,
6002 ["rightrightarrows"] = 8649,
6003 ["rrarr"] = 8649,
6004 ["ddarr"] = 8650,
6005 ["downdownarrows"] = 8650,
6006 ["ReverseEquilibrium"] = 8651,
6007 ["leftrightharpoons"] = 8651,
6008 ["lrhar"] = 8651,
6009 ["Equilibrium"] = 8652,
6010 ["rightleftharpoons"] = 8652,
6011 ["rlhar"] = 8652,
6012 ["nLeftarrow"] = 8653,
6013 ["nlArr"] = 8653,
6014 ["nLeftrightarrow"] = 8654,
6015 ["nhArr"] = 8654,
6016 ["nRightarrow"] = 8655,
6017 ["nrArr"] = 8655,
6018 ["DoubleLeftArrow"] = 8656,
6019 ["Leftarrow"] = 8656,
6020 ["lArr"] = 8656,
6021 ["DoubleUpArrow"] = 8657,
6022 ["Uparrow"] = 8657,
6023 ["uArr"] = 8657,
6024 ["DoubleRightArrow"] = 8658,
6025 ["Implies"] = 8658,
6026 ["Rightarrow"] = 8658,

```



```

6027 ["rArr"] = 8658,
6028 ["DoubleDownArrow"] = 8659,
6029 ["Downarrow"] = 8659,
6030 ["dArr"] = 8659,
6031 ["DoubleLeftRightArrow"] = 8660,
6032 ["Leftrightarrow"] = 8660,
6033 ["hArr"] = 8660,
6034 ["iff"] = 8660,
6035 ["DoubleUpDownArrow"] = 8661,
6036 ["Updownarrow"] = 8661,
6037 ["vArr"] = 8661,
6038 ["nwArr"] = 8662,
6039 ["neArr"] = 8663,
6040 ["seArr"] = 8664,
6041 ["swArr"] = 8665,
6042 ["Lleftarrow"] = 8666,
6043 ["lAarr"] = 8666,
6044 ["Rrightarrow"] = 8667,
6045 ["rAarr"] = 8667,
6046 ["zigrarr"] = 8669,
6047 ["LeftArrowBar"] = 8676,
6048 ["larrb"] = 8676,
6049 ["RightArrowBar"] = 8677,
6050 ["rarrb"] = 8677,
6051 ["DownArrowUpArrow"] = 8693,
6052 ["duarr"] = 8693,
6053 ["loarr"] = 8701,
6054 ["roarr"] = 8702,
6055 ["hoarr"] = 8703,
6056 ["ForAll"] = 8704,
6057 ["forall"] = 8704,
6058 ["comp"] = 8705,
6059 ["complement"] = 8705,
6060 ["PartialD"] = 8706,
6061 ["npart"] = {8706, 824},
6062 ["part"] = 8706,
6063 ["Exists"] = 8707,
6064 ["exist"] = 8707,
6065 ["NotExists"] = 8708,
6066 ["nexist"] = 8708,
6067 ["nexists"] = 8708,
6068 ["empty"] = 8709,
6069 ["emptyset"] = 8709,
6070 ["emptyv"] = 8709,
6071 ["varnothing"] = 8709,
6072 ["Del"] = 8711,
6073 ["nabla"] = 8711,

```

```

6074 ["Element"] = 8712,
6075 ["in"] = 8712,
6076 ["isin"] = 8712,
6077 ["isinv"] = 8712,
6078 ["NotElement"] = 8713,
6079 ["notin"] = 8713,
6080 ["notinva"] = 8713,
6081 ["ReverseElement"] = 8715,
6082 ["SuchThat"] = 8715,
6083 ["ni"] = 8715,
6084 ["niv"] = 8715,
6085 ["NotReverseElement"] = 8716,
6086 ["notni"] = 8716,
6087 ["notniva"] = 8716,
6088 ["Product"] = 8719,
6089 ["prod"] = 8719,
6090 ["Coproduct"] = 8720,
6091 ["coprod"] = 8720,
6092 ["Sum"] = 8721,
6093 ["sum"] = 8721,
6094 ["minus"] = 8722,
6095 ["MinusPlus"] = 8723,
6096 ["mnplus"] = 8723,
6097 ["mp"] = 8723,
6098 ["dotplus"] = 8724,
6099 ["plusdo"] = 8724,
6100 ["Backslash"] = 8726,
6101 ["setminus"] = 8726,
6102 ["setmn"] = 8726,
6103 ["smallsetminus"] = 8726,
6104 ["ssetmn"] = 8726,
6105 ["lowast"] = 8727,
6106 ["SmallCircle"] = 8728,
6107 ["compfn"] = 8728,
6108 ["Sqrt"] = 8730,
6109 ["radic"] = 8730,
6110 ["Proportional"] = 8733,
6111 ["prop"] = 8733,
6112 ["propto"] = 8733,
6113 ["varpropto"] = 8733,
6114 ["vprop"] = 8733,
6115 ["infin"] = 8734,
6116 ["angrt"] = 8735,
6117 ["ang"] = 8736,
6118 ["angle"] = 8736,
6119 ["nang"] = {8736, 8402},
6120 ["angmsd"] = 8737,

```

```

6121 ["measuredangle"] = 8737,
6122 ["angsph"] = 8738,
6123 ["VerticalBar"] = 8739,
6124 ["mid"] = 8739,
6125 ["shortmid"] = 8739,
6126 ["smid"] = 8739,
6127 ["NotVerticalBar"] = 8740,
6128 ["nmid"] = 8740,
6129 ["nshortmid"] = 8740,
6130 ["nsmid"] = 8740,
6131 ["DoubleVerticalBar"] = 8741,
6132 ["par"] = 8741,
6133 ["parallel"] = 8741,
6134 ["shortparallel"] = 8741,
6135 ["spar"] = 8741,
6136 ["NotDoubleVerticalBar"] = 8742,
6137 ["npar"] = 8742,
6138 ["nparallel"] = 8742,
6139 ["nshortparallel"] = 8742,
6140 ["nspar"] = 8742,
6141 ["and"] = 8743,
6142 ["wedge"] = 8743,
6143 ["or"] = 8744,
6144 ["vee"] = 8744,
6145 ["cap"] = 8745,
6146 ["caps"] = {8745, 65024},
6147 ["cup"] = 8746,
6148 ["cups"] = {8746, 65024},
6149 ["Integral"] = 8747,
6150 ["int"] = 8747,
6151 ["Int"] = 8748,
6152 ["iiint"] = 8749,
6153 ["tint"] = 8749,
6154 ["ContourIntegral"] = 8750,
6155 ["conint"] = 8750,
6156 ["oint"] = 8750,
6157 ["Conint"] = 8751,
6158 ["DoubleContourIntegral"] = 8751,
6159 ["Cconint"] = 8752,
6160 ["cwint"] = 8753,
6161 ["ClockwiseContourIntegral"] = 8754,
6162 ["cwconint"] = 8754,
6163 ["CounterClockwiseContourIntegral"] = 8755,
6164 ["awconint"] = 8755,
6165 ["Therefore"] = 8756,
6166 ["there4"] = 8756,
6167 ["therefore"] = 8756,

```

```

6168 ["Because"] = 8757,
6169 ["becaus"] = 8757,
6170 ["because"] = 8757,
6171 ["ratio"] = 8758,
6172 ["Colon"] = 8759,
6173 ["Proportion"] = 8759,
6174 ["dotminus"] = 8760,
6175 ["minusd"] = 8760,
6176 ["mDDot"] = 8762,
6177 ["homtht"] = 8763,
6178 ["Tilde"] = 8764,
6179 ["nvsim"] = {8764, 8402},
6180 ["sim"] = 8764,
6181 ["thicksim"] = 8764,
6182 ["thksim"] = 8764,
6183 ["backsim"] = 8765,
6184 ["bsim"] = 8765,
6185 ["race"] = {8765, 817},
6186 ["ac"] = 8766,
6187 ["acE"] = {8766, 819},
6188 ["mstpos"] = 8766,
6189 ["acd"] = 8767,
6190 ["VerticalTilde"] = 8768,
6191 ["wr"] = 8768,
6192 ["wreath"] = 8768,
6193 ["NotTilde"] = 8769,
6194 ["nsim"] = 8769,
6195 ["EqualTilde"] = 8770,
6196 ["NotEqualTilde"] = {8770, 824},
6197 ["eqsim"] = 8770,
6198 ["esim"] = 8770,
6199 ["nesim"] = {8770, 824},
6200 ["TildeEqual"] = 8771,
6201 ["sime"] = 8771,
6202 ["simeq"] = 8771,
6203 ["NotTildeEqual"] = 8772,
6204 ["nsime"] = 8772,
6205 ["nsimeq"] = 8772,
6206 ["TildeFullEqual"] = 8773,
6207 ["cong"] = 8773,
6208 ["simne"] = 8774,
6209 ["NotTildeFullEqual"] = 8775,
6210 ["ncong"] = 8775,
6211 ["TildeTilde"] = 8776,
6212 ["ap"] = 8776,
6213 ["approx"] = 8776,
6214 ["asymp"] = 8776,

```

```

6215 ["thickapprox"] = 8776,
6216 ["thkap"] = 8776,
6217 ["NotTildeTilde"] = 8777,
6218 ["nap"] = 8777,
6219 ["napprox"] = 8777,
6220 ["ape"] = 8778,
6221 ["approxeq"] = 8778,
6222 ["apid"] = 8779,
6223 ["napid"] = {8779, 824},
6224 ["backcong"] = 8780,
6225 ["bcong"] = 8780,
6226 ["CupCap"] = 8781,
6227 ["asympeq"] = 8781,
6228 ["nvap"] = {8781, 8402},
6229 ["Bumpeq"] = 8782,
6230 ["HumpDownHump"] = 8782,
6231 ["NotHumpDownHump"] = {8782, 824},
6232 ["bump"] = 8782,
6233 ["nbump"] = {8782, 824},
6234 ["HumpEqual"] = 8783,
6235 ["NotHumpEqual"] = {8783, 824},
6236 ["bumpe"] = 8783,
6237 ["bumpeq"] = 8783,
6238 ["nbumpe"] = {8783, 824},
6239 ["DotEqual"] = 8784,
6240 ["doteq"] = 8784,
6241 ["esdot"] = 8784,
6242 ["nedot"] = {8784, 824},
6243 ["doteqdot"] = 8785,
6244 ["eDot"] = 8785,
6245 ["efDot"] = 8786,
6246 ["fallingdotseq"] = 8786,
6247 ["erDot"] = 8787,
6248 ["risingdotseq"] = 8787,
6249 ["Assign"] = 8788,
6250 ["colone"] = 8788,
6251 ["coloneq"] = 8788,
6252 ["ecolon"] = 8789,
6253 ["eqcolon"] = 8789,
6254 ["ecir"] = 8790,
6255 ["eqcirc"] = 8790,
6256 ["circeq"] = 8791,
6257 ["cire"] = 8791,
6258 ["wedgeq"] = 8793,
6259 ["veeeq"] = 8794,
6260 ["triangleq"] = 8796,
6261 ["trie"] = 8796,

```

```

6262 ["equest"] = 8799,
6263 ["questeq"] = 8799,
6264 ["NotEqual"] = 8800,
6265 ["ne"] = 8800,
6266 ["Congruent"] = 8801,
6267 ["bnequiv"] = {8801, 8421},
6268 ["equiv"] = 8801,
6269 ["NotCongruent"] = 8802,
6270 ["nequiv"] = 8802,
6271 ["le"] = 8804,
6272 ["leq"] = 8804,
6273 ["nvle"] = {8804, 8402},
6274 ["GreaterEqual"] = 8805,
6275 ["ge"] = 8805,
6276 ["geq"] = 8805,
6277 ["nvge"] = {8805, 8402},
6278 ["LessFullEqual"] = 8806,
6279 ["lE"] = 8806,
6280 ["leqq"] = 8806,
6281 ["nLE"] = {8806, 824},
6282 ["nleqq"] = {8806, 824},
6283 ["GreaterFullEqual"] = 8807,
6284 ["NotGreaterFullEqual"] = {8807, 824},
6285 ["gE"] = 8807,
6286 ["geqq"] = 8807,
6287 ["ngE"] = {8807, 824},
6288 ["ngeqq"] = {8807, 824},
6289 ["lnE"] = 8808,
6290 ["lneqq"] = 8808,
6291 ["lvertneqq"] = {8808, 65024},
6292 ["lvnE"] = {8808, 65024},
6293 ["gnE"] = 8809,
6294 ["gneqq"] = 8809,
6295 ["gvertneqq"] = {8809, 65024},
6296 ["gvnE"] = {8809, 65024},
6297 ["Lt"] = 8810,
6298 ["NestedLessLess"] = 8810,
6299 ["NotLessLess"] = {8810, 824},
6300 ["ll"] = 8810,
6301 ["nLt"] = {8810, 8402},
6302 ["nLtv"] = {8810, 824},
6303 ["Gt"] = 8811,
6304 ["NestedGreaterGreater"] = 8811,
6305 ["NotGreaterGreater"] = {8811, 824},
6306 ["gg"] = 8811,
6307 ["nGt"] = {8811, 8402},
6308 ["nGtv"] = {8811, 824},

```

```

6309 ["between"] = 8812,
6310 ["twixt"] = 8812,
6311 ["NotCupCap"] = 8813,
6312 ["NotLess"] = 8814,
6313 ["nless"] = 8814,
6314 ["nlt"] = 8814,
6315 ["NotGreater"] = 8815,
6316 ["ngt"] = 8815,
6317 ["ngtr"] = 8815,
6318 ["NotLessEqual"] = 8816,
6319 ["nle"] = 8816,
6320 ["nleq"] = 8816,
6321 ["NotGreaterEqual"] = 8817,
6322 ["nge"] = 8817,
6323 ["ngeq"] = 8817,
6324 ["LessTilde"] = 8818,
6325 ["lesssim"] = 8818,
6326 ["lsim"] = 8818,
6327 ["GreaterTilde"] = 8819,
6328 ["gsim"] = 8819,
6329 ["gtrsim"] = 8819,
6330 ["NotLessTilde"] = 8820,
6331 ["nlsim"] = 8820,
6332 ["NotGreaterTilde"] = 8821,
6333 ["ngsim"] = 8821,
6334 ["LessGreater"] = 8822,
6335 ["lessgtr"] = 8822,
6336 ["lg"] = 8822,
6337 ["GreaterLess"] = 8823,
6338 ["gl"] = 8823,
6339 ["gtrless"] = 8823,
6340 ["NotLessGreater"] = 8824,
6341 ["ntlg"] = 8824,
6342 ["NotGreaterLess"] = 8825,
6343 ["ntgl"] = 8825,
6344 ["Precedes"] = 8826,
6345 ["pr"] = 8826,
6346 ["prec"] = 8826,
6347 ["Succeeds"] = 8827,
6348 ["sc"] = 8827,
6349 ["succ"] = 8827,
6350 ["PrecedesSlantEqual"] = 8828,
6351 ["prcue"] = 8828,
6352 ["preccurlyeq"] = 8828,
6353 ["SucceedsSlantEqual"] = 8829,
6354 ["sccue"] = 8829,
6355 ["succcurlyeq"] = 8829,

```

```

6356 ["PrecedesTilde"] = 8830,
6357 ["precsim"] = 8830,
6358 ["prsim"] = 8830,
6359 ["NotSucceedsTilde"] = {8831, 824},
6360 ["SucceedsTilde"] = 8831,
6361 ["scsim"] = 8831,
6362 ["succsim"] = 8831,
6363 ["NotPrecedes"] = 8832,
6364 ["npr"] = 8832,
6365 ["nprec"] = 8832,
6366 ["NotSucceeds"] = 8833,
6367 ["nsc"] = 8833,
6368 ["nsucc"] = 8833,
6369 ["NotSubset"] = {8834, 8402},
6370 ["nsubset"] = {8834, 8402},
6371 ["sub"] = 8834,
6372 ["subset"] = 8834,
6373 ["vnsup"] = {8834, 8402},
6374 ["NotSuperset"] = {8835, 8402},
6375 ["Superset"] = 8835,
6376 ["nsupset"] = {8835, 8402},
6377 ["sup"] = 8835,
6378 ["supset"] = 8835,
6379 ["vnsup"] = {8835, 8402},
6380 ["nsub"] = 8836,
6381 ["nsup"] = 8837,
6382 ["SubsetEqual"] = 8838,
6383 ["sube"] = 8838,
6384 ["subseteq"] = 8838,
6385 ["SupersetEqual"] = 8839,
6386 ["supe"] = 8839,
6387 ["supseteq"] = 8839,
6388 ["NotSubsetEqual"] = 8840,
6389 ["nsube"] = 8840,
6390 ["nsubseteq"] = 8840,
6391 ["NotSupersetEqual"] = 8841,
6392 ["nsupe"] = 8841,
6393 ["nsupseteq"] = 8841,
6394 ["subne"] = 8842,
6395 ["subsetneq"] = 8842,
6396 ["varsubsetneq"] = {8842, 65024},
6397 ["vsubne"] = {8842, 65024},
6398 ["supne"] = 8843,
6399 ["supsetneq"] = 8843,
6400 ["varsupsetneq"] = {8843, 65024},
6401 ["vsupne"] = {8843, 65024},
6402 ["cupdot"] = 8845,

```



```

6403 ["UnionPlus"] = 8846,
6404 ["uplus"] = 8846,
6405 ["NotSquareSubset"] = {8847, 824},
6406 ["SquareSubset"] = 8847,
6407 ["sqsub"] = 8847,
6408 ["sqsubset"] = 8847,
6409 ["NotSquareSuperset"] = {8848, 824},
6410 ["SquareSuperset"] = 8848,
6411 ["sqsup"] = 8848,
6412 ["sqsupset"] = 8848,
6413 ["SquareSubsetEqual"] = 8849,
6414 ["sqsube"] = 8849,
6415 ["sqsubseteq"] = 8849,
6416 ["SquareSupersetEqual"] = 8850,
6417 ["sqsupe"] = 8850,
6418 ["sqsupseteq"] = 8850,
6419 ["SquareIntersection"] = 8851,
6420 ["sqcap"] = 8851,
6421 ["sqcaps"] = {8851, 65024},
6422 ["SquareUnion"] = 8852,
6423 ["sqcup"] = 8852,
6424 ["sqcups"] = {8852, 65024},
6425 ["CirclePlus"] = 8853,
6426 ["oplus"] = 8853,
6427 ["CircleMinus"] = 8854,
6428 ["ominus"] = 8854,
6429 ["CircleTimes"] = 8855,
6430 ["otimes"] = 8855,
6431 ["osol"] = 8856,
6432 ["CircleDot"] = 8857,
6433 ["odot"] = 8857,
6434 ["circledcirc"] = 8858,
6435 ["ocir"] = 8858,
6436 ["circledast"] = 8859,
6437 ["oast"] = 8859,
6438 ["circleddash"] = 8861,
6439 ["odash"] = 8861,
6440 ["boxplus"] = 8862,
6441 ["plusb"] = 8862,
6442 ["boxminus"] = 8863,
6443 ["minusb"] = 8863,
6444 ["boxtimes"] = 8864,
6445 ["timesb"] = 8864,
6446 ["dotssquare"] = 8865,
6447 ["sdotb"] = 8865,
6448 ["RightTee"] = 8866,
6449 ["vdash"] = 8866,

```

```

6450 ["LeftTee"] = 8867,
6451 ["dashv"] = 8867,
6452 ["DownTee"] = 8868,
6453 ["top"] = 8868,
6454 ["UpTee"] = 8869,
6455 ["bot"] = 8869,
6456 ["bottom"] = 8869,
6457 ["perp"] = 8869,
6458 ["models"] = 8871,
6459 ["DoubleRightTee"] = 8872,
6460 ["vDash"] = 8872,
6461 ["Vdash"] = 8873,
6462 ["Vvdash"] = 8874,
6463 ["VDash"] = 8875,
6464 ["nvdash"] = 8876,
6465 ["nvDash"] = 8877,
6466 ["nVdash"] = 8878,
6467 ["nVDash"] = 8879,
6468 ["prurel"] = 8880,
6469 ["LeftTriangle"] = 8882,
6470 ["vartriangleleft"] = 8882,
6471 ["vltri"] = 8882,
6472 ["RightTriangle"] = 8883,
6473 ["vartriangleright"] = 8883,
6474 ["vrtri"] = 8883,
6475 ["LeftTriangleEqual"] = 8884,
6476 ["ltrie"] = 8884,
6477 ["nvltrie"] = {8884, 8402},
6478 ["trianglelefteq"] = 8884,
6479 ["RightTriangleEqual"] = 8885,
6480 ["nvrtrie"] = {8885, 8402},
6481 ["rtrie"] = 8885,
6482 ["trianglerighteq"] = 8885,
6483 ["origof"] = 8886,
6484 ["imof"] = 8887,
6485 ["multimap"] = 8888,
6486 ["mumap"] = 8888,
6487 ["hercon"] = 8889,
6488 ["intcal"] = 8890,
6489 ["intercal"] = 8890,
6490 ["veebar"] = 8891,
6491 ["barvee"] = 8893,
6492 ["angrtvb"] = 8894,
6493 ["lrtri"] = 8895,
6494 ["Wedge"] = 8896,
6495 ["bigwedge"] = 8896,
6496 ["xwedge"] = 8896,

```

```

6497 ["Vee"] = 8897,
6498 ["bigvee"] = 8897,
6499 ["xvee"] = 8897,
6500 ["Intersection"] = 8898,
6501 ["bigcap"] = 8898,
6502 ["xcap"] = 8898,
6503 ["Union"] = 8899,
6504 ["bigcup"] = 8899,
6505 ["xcup"] = 8899,
6506 ["Diamond"] = 8900,
6507 ["diam"] = 8900,
6508 ["diamond"] = 8900,
6509 ["sdot"] = 8901,
6510 ["Star"] = 8902,
6511 ["sstarf"] = 8902,
6512 ["divideontimes"] = 8903,
6513 ["divonx"] = 8903,
6514 ["bowtie"] = 8904,
6515 ["ltimes"] = 8905,
6516 ["rtimes"] = 8906,
6517 ["leftthreetimes"] = 8907,
6518 ["lthree"] = 8907,
6519 ["rightthreetimes"] = 8908,
6520 ["rthree"] = 8908,
6521 ["backsimeq"] = 8909,
6522 ["bsime"] = 8909,
6523 ["curlyvee"] = 8910,
6524 ["cuvee"] = 8910,
6525 ["curlywedge"] = 8911,
6526 ["cuwed"] = 8911,
6527 ["Sub"] = 8912,
6528 ["Subset"] = 8912,
6529 ["Sup"] = 8913,
6530 ["Supset"] = 8913,
6531 ["Cap"] = 8914,
6532 ["Cup"] = 8915,
6533 ["fork"] = 8916,
6534 ["pitchfork"] = 8916,
6535 ["epar"] = 8917,
6536 ["lessdot"] = 8918,
6537 ["ltdot"] = 8918,
6538 ["gtdot"] = 8919,
6539 ["gtrdot"] = 8919,
6540 ["Ll"] = 8920,
6541 ["nLl"] = {8920, 824},
6542 ["Gg"] = 8921,
6543 ["ggg"] = 8921,

```

```

6544 ["nGg"] = {8921, 824},
6545 ["LessEqualGreater"] = 8922,
6546 ["leg"] = 8922,
6547 ["lesg"] = {8922, 65024},
6548 ["lesseqgtr"] = 8922,
6549 ["GreaterEqualLess"] = 8923,
6550 ["gel"] = 8923,
6551 ["gesl"] = {8923, 65024},
6552 ["gtreqless"] = 8923,
6553 ["cuepr"] = 8926,
6554 ["curlyeqprec"] = 8926,
6555 ["cuesc"] = 8927,
6556 ["curlyeqsucc"] = 8927,
6557 ["NotPrecedesSlantEqual"] = 8928,
6558 ["nprcue"] = 8928,
6559 ["NotSucceedsSlantEqual"] = 8929,
6560 ["nsccue"] = 8929,
6561 ["NotSquareSubsetEqual"] = 8930,
6562 ["nsqsube"] = 8930,
6563 ["NotSquareSupersetEqual"] = 8931,
6564 ["nsqsupe"] = 8931,
6565 ["lnsim"] = 8934,
6566 ["gnsim"] = 8935,
6567 ["precnsim"] = 8936,
6568 ["prnsim"] = 8936,
6569 ["scnsim"] = 8937,
6570 ["succnsim"] = 8937,
6571 ["NotLeftTriangle"] = 8938,
6572 ["nltri"] = 8938,
6573 ["ntriangleleft"] = 8938,
6574 ["NotRightTriangle"] = 8939,
6575 ["nrtri"] = 8939,
6576 ["ntriangleright"] = 8939,
6577 ["NotLeftTriangleEqual"] = 8940,
6578 ["nltrie"] = 8940,
6579 ["ntrianglelefteq"] = 8940,
6580 ["NotRightTriangleEqual"] = 8941,
6581 ["nrtrie"] = 8941,
6582 ["ntrianglerighteq"] = 8941,
6583 ["vellip"] = 8942,
6584 ["ctdot"] = 8943,
6585 ["utdot"] = 8944,
6586 ["dtdot"] = 8945,
6587 ["disin"] = 8946,
6588 ["isinsv"] = 8947,
6589 ["isins"] = 8948,
6590 ["isindot"] = 8949,

```

```

6591 ["notindot"] = {8949, 824},
6592 ["notinvc"] = 8950,
6593 ["notinvb"] = 8951,
6594 ["isinE"] = 8953,
6595 ["notinE"] = {8953, 824},
6596 ["nisd"] = 8954,
6597 ["xnis"] = 8955,
6598 ["nis"] = 8956,
6599 ["notnivc"] = 8957,
6600 ["notnivb"] = 8958,
6601 ["barwed"] = 8965,
6602 ["barwedge"] = 8965,
6603 ["Barwed"] = 8966,
6604 ["doublebarwedge"] = 8966,
6605 ["LeftCeiling"] = 8968,
6606 ["lceil"] = 8968,
6607 ["RightCeiling"] = 8969,
6608 ["rceil"] = 8969,
6609 ["LeftFloor"] = 8970,
6610 ["lfloor"] = 8970,
6611 ["RightFloor"] = 8971,
6612 ["rfloor"] = 8971,
6613 ["drcrop"] = 8972,
6614 ["dlcrop"] = 8973,
6615 ["urcrop"] = 8974,
6616 ["ulcrop"] = 8975,
6617 ["bnot"] = 8976,
6618 ["profline"] = 8978,
6619 ["profsurf"] = 8979,
6620 ["telrec"] = 8981,
6621 ["target"] = 8982,
6622 ["ulcorn"] = 8988,
6623 ["ulcorner"] = 8988,
6624 ["urcorn"] = 8989,
6625 ["urcorner"] = 8989,
6626 ["dlcorn"] = 8990,
6627 ["llcorner"] = 8990,
6628 ["drcorn"] = 8991,
6629 ["lrcorner"] = 8991,
6630 ["frown"] = 8994,
6631 ["sfrown"] = 8994,
6632 ["smile"] = 8995,
6633 ["ssmile"] = 8995,
6634 ["cylcty"] = 9005,
6635 ["profalar"] = 9006,
6636 ["topbot"] = 9014,
6637 ["ovbar"] = 9021,

```

```

6638 ["solbar"] = 9023,
6639 ["angzarr"] = 9084,
6640 ["lmoust"] = 9136,
6641 ["lmoustache"] = 9136,
6642 ["rmoust"] = 9137,
6643 ["rmoustache"] = 9137,
6644 ["OverBracket"] = 9140,
6645 ["tbrk"] = 9140,
6646 ["UnderBracket"] = 9141,
6647 ["bbrk"] = 9141,
6648 ["bbrktbrk"] = 9142,
6649 ["OverParenthesis"] = 9180,
6650 ["UnderParenthesis"] = 9181,
6651 ["OverBrace"] = 9182,
6652 ["UnderBrace"] = 9183,
6653 ["trpezium"] = 9186,
6654 ["elinters"] = 9191,
6655 ["blank"] = 9251,
6656 ["circledS"] = 9416,
6657 ["oS"] = 9416,
6658 ["HorizontalLine"] = 9472,
6659 ["boxh"] = 9472,
6660 ["boxv"] = 9474,
6661 ["boxdr"] = 9484,
6662 ["boxdl"] = 9488,
6663 ["boxur"] = 9492,
6664 ["boxul"] = 9496,
6665 ["boxvr"] = 9500,
6666 ["boxvl"] = 9508,
6667 ["boxhd"] = 9516,
6668 ["boxhu"] = 9524,
6669 ["boxvh"] = 9532,
6670 ["boxH"] = 9552,
6671 ["boxV"] = 9553,
6672 ["boxdR"] = 9554,
6673 ["boxDr"] = 9555,
6674 ["boxDR"] = 9556,
6675 ["boxdL"] = 9557,
6676 ["boxDL"] = 9558,
6677 ["boxDL"] = 9559,
6678 ["boxuR"] = 9560,
6679 ["boxUr"] = 9561,
6680 ["boxUR"] = 9562,
6681 ["boxuL"] = 9563,
6682 ["boxUL"] = 9564,
6683 ["boxUL"] = 9565,
6684 ["boxvR"] = 9566,

```

```

6685 ["boxVr"] = 9567,
6686 ["boxVR"] = 9568,
6687 ["boxvL"] = 9569,
6688 ["boxVl"] = 9570,
6689 ["boxVL"] = 9571,
6690 ["boxHd"] = 9572,
6691 ["boxhD"] = 9573,
6692 ["boxHD"] = 9574,
6693 ["boxHu"] = 9575,
6694 ["boxhU"] = 9576,
6695 ["boxHU"] = 9577,
6696 ["boxvH"] = 9578,
6697 ["boxVh"] = 9579,
6698 ["boxVH"] = 9580,
6699 ["uhblk"] = 9600,
6700 ["lhblk"] = 9604,
6701 ["block"] = 9608,
6702 ["blk14"] = 9617,
6703 ["blk12"] = 9618,
6704 ["blk34"] = 9619,
6705 ["Square"] = 9633,
6706 ["squ"] = 9633,
6707 ["square"] = 9633,
6708 ["FilledVerySmallSquare"] = 9642,
6709 ["blacksquare"] = 9642,
6710 ["squarf"] = 9642,
6711 ["squf"] = 9642,
6712 ["EmptyVerySmallSquare"] = 9643,
6713 ["rect"] = 9645,
6714 ["marker"] = 9646,
6715 ["fltns"] = 9649,
6716 ["bigtriangleup"] = 9651,
6717 ["xutri"] = 9651,
6718 ["blacktriangle"] = 9652,
6719 ["utrif"] = 9652,
6720 ["triangle"] = 9653,
6721 ["utri"] = 9653,
6722 ["blacktriangleright"] = 9656,
6723 ["rtrif"] = 9656,
6724 ["rtri"] = 9657,
6725 ["triangleright"] = 9657,
6726 ["bigtriangledown"] = 9661,
6727 ["xdtri"] = 9661,
6728 ["blacktriangledown"] = 9662,
6729 ["dtrif"] = 9662,
6730 ["dtri"] = 9663,
6731 ["triangledown"] = 9663,

```

```

6732 ["blacktriangleleft"] = 9666,
6733 ["ltrif"] = 9666,
6734 ["ltri"] = 9667,
6735 ["triangleleft"] = 9667,
6736 ["loz"] = 9674,
6737 ["lozenge"] = 9674,
6738 ["cir"] = 9675,
6739 ["tridot"] = 9708,
6740 ["bigcirc"] = 9711,
6741 ["xcirc"] = 9711,
6742 ["ultri"] = 9720,
6743 ["urtri"] = 9721,
6744 ["lltri"] = 9722,
6745 ["EmptySmallSquare"] = 9723,
6746 ["FilledSmallSquare"] = 9724,
6747 ["bigstar"] = 9733,
6748 ["starf"] = 9733,
6749 ["star"] = 9734,
6750 ["phone"] = 9742,
6751 ["female"] = 9792,
6752 ["male"] = 9794,
6753 ["spades"] = 9824,
6754 ["spadesuit"] = 9824,
6755 ["clubs"] = 9827,
6756 ["clubsuit"] = 9827,
6757 ["hearts"] = 9829,
6758 ["heartsuit"] = 9829,
6759 ["diamondsuit"] = 9830,
6760 ["diams"] = 9830,
6761 ["sung"] = 9834,
6762 ["flat"] = 9837,
6763 ["natur"] = 9838,
6764 ["natural"] = 9838,
6765 ["sharp"] = 9839,
6766 ["check"] = 10003,
6767 ["checkmark"] = 10003,
6768 ["cross"] = 10007,
6769 ["malt"] = 10016,
6770 ["maltese"] = 10016,
6771 ["sext"] = 10038,
6772 ["VerticalSeparator"] = 10072,
6773 ["lbbbrk"] = 10098,
6774 ["rbbrk"] = 10099,
6775 ["bsolhsub"] = 10184,
6776 ["suphsol"] = 10185,
6777 ["LeftDoubleBracket"] = 10214,
6778 ["lobrk"] = 10214,

```



6779 ["RightDoubleBracket"] = 10215,  
 6780 ["robrk"] = 10215,  
 6781 ["LeftAngleBracket"] = 10216,  
 6782 ["lang"] = 10216,  
 6783 ["langle"] = 10216,  
 6784 ["RightAngleBracket"] = 10217,  
 6785 ["rang"] = 10217,  
 6786 ["rangle"] = 10217,  
 6787 ["Lang"] = 10218,  
 6788 ["Rang"] = 10219,  
 6789 ["loang"] = 10220,  
 6790 ["roang"] = 10221,  
 6791 ["LongLeftArrow"] = 10229,  
 6792 ["longleftarrow"] = 10229,  
 6793 ["xlarr"] = 10229,  
 6794 ["LongRightArrow"] = 10230,  
 6795 ["longrightarrow"] = 10230,  
 6796 ["xrarr"] = 10230,  
 6797 ["LongLeftRightArrow"] = 10231,  
 6798 ["longleftrightarrow"] = 10231,  
 6799 ["xharr"] = 10231,  
 6800 ["DoubleLongLeftArrow"] = 10232,  
 6801 ["Longleftarrow"] = 10232,  
 6802 ["xlArr"] = 10232,  
 6803 ["DoubleLongRightArrow"] = 10233,  
 6804 ["Longrightarrow"] = 10233,  
 6805 ["xrArr"] = 10233,  
 6806 ["DoubleLongLeftRightArrow"] = 10234,  
 6807 ["Longleftrightarrow"] = 10234,  
 6808 ["xhArr"] = 10234,  
 6809 ["longmapsto"] = 10236,  
 6810 ["xmap"] = 10236,  
 6811 ["dzigrarr"] = 10239,  
 6812 ["nvlArr"] = 10498,  
 6813 ["nvrArr"] = 10499,  
 6814 ["nvHarr"] = 10500,  
 6815 ["Map"] = 10501,  
 6816 ["lbarr"] = 10508,  
 6817 ["bkarow"] = 10509,  
 6818 ["rbarr"] = 10509,  
 6819 ["lBarr"] = 10510,  
 6820 ["dbkarow"] = 10511,  
 6821 ["rBarr"] = 10511,  
 6822 ["RBarr"] = 10512,  
 6823 ["drbkarow"] = 10512,  
 6824 ["DDottrahd"] = 10513,  
 6825 ["UpArrowBar"] = 10514,

```

6826 ["DownArrowBar"] = 10515,
6827 ["Rarrtl"] = 10518,
6828 ["latail"] = 10521,
6829 ["ratail"] = 10522,
6830 ["lAtail"] = 10523,
6831 ["rAtail"] = 10524,
6832 ["larrfs"] = 10525,
6833 ["rarrfs"] = 10526,
6834 ["larrbfs"] = 10527,
6835 ["rarrbfs"] = 10528,
6836 ["nwarhk"] = 10531,
6837 ["nearhk"] = 10532,
6838 ["hksearow"] = 10533,
6839 ["searhk"] = 10533,
6840 ["hkswarow"] = 10534,
6841 ["swarhk"] = 10534,
6842 ["nwnear"] = 10535,
6843 ["nesear"] = 10536,
6844 ["toea"] = 10536,
6845 ["seswar"] = 10537,
6846 ["tosa"] = 10537,
6847 ["swnwar"] = 10538,
6848 ["nrarrc"] = {10547, 824},
6849 ["rarrc"] = 10547,
6850 ["cudarrr"] = 10549,
6851 ["ldca"] = 10550,
6852 ["rdca"] = 10551,
6853 ["cudarrrl"] = 10552,
6854 ["larrpl"] = 10553,
6855 ["curarrm"] = 10556,
6856 ["cularrp"] = 10557,
6857 ["rarrpl"] = 10565,
6858 ["harrcir"] = 10568,
6859 ["Uarrocir"] = 10569,
6860 ["lurdshar"] = 10570,
6861 ["ldrushar"] = 10571,
6862 ["LeftRightVector"] = 10574,
6863 ["RightUpDownVector"] = 10575,
6864 ["DownLeftRightVector"] = 10576,
6865 ["LeftUpDownVector"] = 10577,
6866 ["LeftVectorBar"] = 10578,
6867 ["RightVectorBar"] = 10579,
6868 ["RightUpVectorBar"] = 10580,
6869 ["RightDownVectorBar"] = 10581,
6870 ["DownLeftVectorBar"] = 10582,
6871 ["DownRightVectorBar"] = 10583,
6872 ["LeftUpVectorBar"] = 10584,

```

```

6873 ["LeftDownVectorBar"] = 10585,
6874 ["LeftTeeVector"] = 10586,
6875 ["RightTeeVector"] = 10587,
6876 ["RightUpTeeVector"] = 10588,
6877 ["RightDownTeeVector"] = 10589,
6878 ["DownLeftTeeVector"] = 10590,
6879 ["DownRightTeeVector"] = 10591,
6880 ["LeftUpTeeVector"] = 10592,
6881 ["LeftDownTeeVector"] = 10593,
6882 ["lHar"] = 10594,
6883 ["uHar"] = 10595,
6884 ["rHar"] = 10596,
6885 ["dHar"] = 10597,
6886 ["luruhar"] = 10598,
6887 ["ldrdhar"] = 10599,
6888 ["ruluhar"] = 10600,
6889 ["rdldhar"] = 10601,
6890 ["lharul"] = 10602,
6891 ["llhard"] = 10603,
6892 ["rharul"] = 10604,
6893 ["lrhard"] = 10605,
6894 ["UpEquilibrium"] = 10606,
6895 ["udhar"] = 10606,
6896 ["ReverseUpEquilibrium"] = 10607,
6897 ["duhar"] = 10607,
6898 ["RoundImplies"] = 10608,
6899 ["erarr"] = 10609,
6900 ["simrarr"] = 10610,
6901 ["larrsim"] = 10611,
6902 ["rarrsim"] = 10612,
6903 ["rarrap"] = 10613,
6904 ["ltlarr"] = 10614,
6905 ["gtrarr"] = 10616,
6906 ["subrarr"] = 10617,
6907 ["suplarr"] = 10619,
6908 ["lfisht"] = 10620,
6909 ["rfisht"] = 10621,
6910 ["ufisht"] = 10622,
6911 ["dfisht"] = 10623,
6912 ["lopar"] = 10629,
6913 ["ropar"] = 10630,
6914 ["lbrke"] = 10635,
6915 ["rbrke"] = 10636,
6916 ["lbrkslu"] = 10637,
6917 ["rbrksld"] = 10638,
6918 ["lbrksld"] = 10639,
6919 ["rbrkslu"] = 10640,

```

```

6920 ["langd"] = 10641,
6921 ["rangd"] = 10642,
6922 ["lparlt"] = 10643,
6923 ["rpargt"] = 10644,
6924 ["gtlPar"] = 10645,
6925 ["ltrPar"] = 10646,
6926 ["vzigzag"] = 10650,
6927 ["vangrt"] = 10652,
6928 ["angrtvbd"] = 10653,
6929 ["ange"] = 10660,
6930 ["range"] = 10661,
6931 ["dwangle"] = 10662,
6932 ["uwangle"] = 10663,
6933 ["angmsdaa"] = 10664,
6934 ["angmsdab"] = 10665,
6935 ["angmsdac"] = 10666,
6936 ["angmsdad"] = 10667,
6937 ["angmsdae"] = 10668,
6938 ["angmsdaf"] = 10669,
6939 ["angmsdag"] = 10670,
6940 ["angmsdah"] = 10671,
6941 ["bemptyv"] = 10672,
6942 ["demptyv"] = 10673,
6943 ["cemptyv"] = 10674,
6944 ["raemptyv"] = 10675,
6945 ["laemptyv"] = 10676,
6946 ["ohbar"] = 10677,
6947 ["omid"] = 10678,
6948 ["opar"] = 10679,
6949 ["operp"] = 10681,
6950 ["olcross"] = 10683,
6951 ["odsold"] = 10684,
6952 ["olcir"] = 10686,
6953 ["ofcir"] = 10687,
6954 ["olt"] = 10688,
6955 ["ogt"] = 10689,
6956 ["cirscir"] = 10690,
6957 ["cirE"] = 10691,
6958 ["solb"] = 10692,
6959 ["bsolb"] = 10693,
6960 ["boxbox"] = 10697,
6961 ["trish"] = 10701,
6962 ["rtriltri"] = 10702,
6963 ["LeftTriangleBar"] = 10703,
6964 ["NotLeftTriangleBar"] = {10703, 824},
6965 ["NotRightTriangleBar"] = {10704, 824},
6966 ["RightTriangleBar"] = 10704,

```

```

6967 ["i infin"] = 10716,
6968 ["infintie"] = 10717,
6969 ["nvinfin"] = 10718,
6970 ["eparsl"] = 10723,
6971 ["smeparsl"] = 10724,
6972 ["eqvparsl"] = 10725,
6973 ["blacklozenge"] = 10731,
6974 ["lozf"] = 10731,
6975 ["RuleDelayed"] = 10740,
6976 ["dsol"] = 10742,
6977 ["bigodot"] = 10752,
6978 ["xodot"] = 10752,
6979 ["bigoplus"] = 10753,
6980 ["xoplus"] = 10753,
6981 ["bigotimes"] = 10754,
6982 ["xotime"] = 10754,
6983 ["biguplus"] = 10756,
6984 ["xuplus"] = 10756,
6985 ["bigsqcup"] = 10758,
6986 ["xsqcup"] = 10758,
6987 ["iiiint"] = 10764,
6988 ["qint"] = 10764,
6989 ["fpartint"] = 10765,
6990 ["cirfnint"] = 10768,
6991 ["awint"] = 10769,
6992 ["rppolint"] = 10770,
6993 ["scpolint"] = 10771,
6994 ["npolint"] = 10772,
6995 ["pointint"] = 10773,
6996 ["quatint"] = 10774,
6997 ["intlarhk"] = 10775,
6998 ["pluscir"] = 10786,
6999 ["plusacir"] = 10787,
7000 ["simplus"] = 10788,
7001 ["plusdu"] = 10789,
7002 ["plussim"] = 10790,
7003 ["plustwo"] = 10791,
7004 ["mcomma"] = 10793,
7005 ["minusdu"] = 10794,
7006 ["loplus"] = 10797,
7007 ["roplus"] = 10798,
7008 ["Cross"] = 10799,
7009 ["timesd"] = 10800,
7010 ["timesbar"] = 10801,
7011 ["smashp"] = 10803,
7012 ["lotimes"] = 10804,
7013 ["rotimes"] = 10805,

```

```

7014 ["otimesas"] = 10806,
7015 ["Otimes"] = 10807,
7016 ["odiv"] = 10808,
7017 ["tripplus"] = 10809,
7018 ["triminus"] = 10810,
7019 ["tritime"] = 10811,
7020 ["intprod"] = 10812,
7021 ["iprod"] = 10812,
7022 ["amalg"] = 10815,
7023 ["capdot"] = 10816,
7024 ["ncup"] = 10818,
7025 ["ncap"] = 10819,
7026 ["capand"] = 10820,
7027 ["cupor"] = 10821,
7028 ["cupcap"] = 10822,
7029 ["capcup"] = 10823,
7030 ["cupbrcap"] = 10824,
7031 ["capbrcup"] = 10825,
7032 ["cupcup"] = 10826,
7033 ["capcap"] = 10827,
7034 ["ccups"] = 10828,
7035 ["ccaps"] = 10829,
7036 ["ccupssm"] = 10832,
7037 ["And"] = 10835,
7038 ["Or"] = 10836,
7039 ["andand"] = 10837,
7040 ["oror"] = 10838,
7041 ["orslope"] = 10839,
7042 ["andslope"] = 10840,
7043 ["andv"] = 10842,
7044 ["orv"] = 10843,
7045 ["andd"] = 10844,
7046 ["ord"] = 10845,
7047 ["wedbar"] = 10847,
7048 ["sdote"] = 10854,
7049 ["simdot"] = 10858,
7050 ["congdots"] = 10861,
7051 ["ncongdots"] = {10861, 824},
7052 ["easter"] = 10862,
7053 ["apacir"] = 10863,
7054 ["apE"] = 10864,
7055 ["napE"] = {10864, 824},
7056 ["eplus"] = 10865,
7057 ["pluse"] = 10866,
7058 ["Esim"] = 10867,
7059 ["Colone"] = 10868,
7060 ["Equal"] = 10869,

```

```

7061 ["ddotseq"] = 10871,
7062 ["eDDot"] = 10871,
7063 ["equivDD"] = 10872,
7064 ["ltcir"] = 10873,
7065 ["gtcir"] = 10874,
7066 ["ltquest"] = 10875,
7067 ["gtquest"] = 10876,
7068 ["LessSlantEqual"] = 10877,
7069 ["NotLessSlantEqual"] = {10877, 824},
7070 ["leqslant"] = 10877,
7071 ["les"] = 10877,
7072 ["nleqslant"] = {10877, 824},
7073 ["nles"] = {10877, 824},
7074 ["GreaterSlantEqual"] = 10878,
7075 ["NotGreaterSlantEqual"] = {10878, 824},
7076 ["geqslant"] = 10878,
7077 ["ges"] = 10878,
7078 ["ngeqslant"] = {10878, 824},
7079 ["nges"] = {10878, 824},
7080 ["lesdot"] = 10879,
7081 ["gesdot"] = 10880,
7082 ["lesdoto"] = 10881,
7083 ["gesdoto"] = 10882,
7084 ["lesdotor"] = 10883,
7085 ["gesdoto1"] = 10884,
7086 ["lap"] = 10885,
7087 ["lessapprox"] = 10885,
7088 ["gap"] = 10886,
7089 ["gtrapprox"] = 10886,
7090 ["lne"] = 10887,
7091 ["lneq"] = 10887,
7092 ["gne"] = 10888,
7093 ["gneq"] = 10888,
7094 ["lnap"] = 10889,
7095 ["lnapprox"] = 10889,
7096 ["gnap"] = 10890,
7097 ["gnapprox"] = 10890,
7098 ["lEg"] = 10891,
7099 ["lesseqqgtr"] = 10891,
7100 ["gEl"] = 10892,
7101 ["gtreqqless"] = 10892,
7102 ["lsime"] = 10893,
7103 ["gsime"] = 10894,
7104 ["lsimg"] = 10895,
7105 ["gsiml"] = 10896,
7106 ["lgE"] = 10897,
7107 ["glE"] = 10898,

```

```

7108 ["lesges"] = 10899,
7109 ["gesles"] = 10900,
7110 ["els"] = 10901,
7111 ["eqslantless"] = 10901,
7112 ["egs"] = 10902,
7113 ["eqslantgtr"] = 10902,
7114 ["elsdot"] = 10903,
7115 ["egsdot"] = 10904,
7116 ["el"] = 10905,
7117 ["eg"] = 10906,
7118 ["siml"] = 10909,
7119 ["simg"] = 10910,
7120 ["simlE"] = 10911,
7121 ["simgE"] = 10912,
7122 ["LessLess"] = 10913,
7123 ["NotNestedLessLess"] = {10913, 824},
7124 ["GreaterGreater"] = 10914,
7125 ["NotNestedGreaterGreater"] = {10914, 824},
7126 ["glj"] = 10916,
7127 ["gla"] = 10917,
7128 ["ltcc"] = 10918,
7129 ["gtcc"] = 10919,
7130 ["lescc"] = 10920,
7131 ["gescc"] = 10921,
7132 ["smt"] = 10922,
7133 ["lat"] = 10923,
7134 ["smte"] = 10924,
7135 ["smtes"] = {10924, 65024},
7136 ["late"] = 10925,
7137 ["lates"] = {10925, 65024},
7138 ["bumpE"] = 10926,
7139 ["NotPrecedesEqual"] = {10927, 824},
7140 ["PrecedesEqual"] = 10927,
7141 ["npre"] = {10927, 824},
7142 ["npreceq"] = {10927, 824},
7143 ["pre"] = 10927,
7144 ["preceq"] = 10927,
7145 ["NotSucceedsEqual"] = {10928, 824},
7146 ["SucceedsEqual"] = 10928,
7147 ["nsce"] = {10928, 824},
7148 ["nsucceq"] = {10928, 824},
7149 ["sce"] = 10928,
7150 ["succeq"] = 10928,
7151 ["prE"] = 10931,
7152 ["scE"] = 10932,
7153 ["precneqq"] = 10933,
7154 ["prnE"] = 10933,

```



```

7155 ["scnE"] = 10934,
7156 ["succneqq"] = 10934,
7157 ["prap"] = 10935,
7158 ["precapprox"] = 10935,
7159 ["scap"] = 10936,
7160 ["succapprox"] = 10936,
7161 ["precnapprox"] = 10937,
7162 ["prnap"] = 10937,
7163 ["scnap"] = 10938,
7164 ["succnapprox"] = 10938,
7165 ["Pr"] = 10939,
7166 ["Sc"] = 10940,
7167 ["subdot"] = 10941,
7168 ["supdot"] = 10942,
7169 ["subplus"] = 10943,
7170 ["supplus"] = 10944,
7171 ["submult"] = 10945,
7172 ["supmult"] = 10946,
7173 ["subedot"] = 10947,
7174 ["supedot"] = 10948,
7175 ["nsubE"] = {10949, 824},
7176 ["nsubseteqq"] = {10949, 824},
7177 ["subE"] = 10949,
7178 ["subseteqq"] = 10949,
7179 ["nsupE"] = {10950, 824},
7180 ["nsupseteqq"] = {10950, 824},
7181 ["supE"] = 10950,
7182 ["supseteqq"] = 10950,
7183 ["subsim"] = 10951,
7184 ["supsim"] = 10952,
7185 ["subnE"] = 10955,
7186 ["subsetneqq"] = 10955,
7187 ["varsubsetneqq"] = {10955, 65024},
7188 ["vsubnE"] = {10955, 65024},
7189 ["supnE"] = 10956,
7190 ["supsetneqq"] = 10956,
7191 ["varsupsetneqq"] = {10956, 65024},
7192 ["vsupnE"] = {10956, 65024},
7193 ["csub"] = 10959,
7194 ["csup"] = 10960,
7195 ["csube"] = 10961,
7196 ["csupe"] = 10962,
7197 ["subsup"] = 10963,
7198 ["supsub"] = 10964,
7199 ["subsub"] = 10965,
7200 ["supsup"] = 10966,
7201 ["suphsub"] = 10967,

```

```

7202 ["supdsub"] = 10968,
7203 ["forkv"] = 10969,
7204 ["topfork"] = 10970,
7205 ["mlcp"] = 10971,
7206 ["Dashv"] = 10980,
7207 ["DoubleLeftTee"] = 10980,
7208 ["Vdashl"] = 10982,
7209 ["Barv"] = 10983,
7210 ["vBar"] = 10984,
7211 ["vBarv"] = 10985,
7212 ["Vbar"] = 10987,
7213 ["Not"] = 10988,
7214 ["bNot"] = 10989,
7215 ["rnmid"] = 10990,
7216 ["cirmid"] = 10991,
7217 ["midcir"] = 10992,
7218 ["topcir"] = 10993,
7219 ["nhpar"] = 10994,
7220 ["parsim"] = 10995,
7221 ["nparsl"] = {11005, 8421},
7222 ["parsl"] = 11005,
7223 ["fflig"] = 64256,
7224 ["filig"] = 64257,
7225 ["fllig"] = 64258,
7226 ["ffilig"] = 64259,
7227 ["ffllig"] = 64260,
7228 ["Ascr"] = 119964,
7229 ["Cscr"] = 119966,
7230 ["Dscr"] = 119967,
7231 ["Gscr"] = 119970,
7232 ["Jscr"] = 119973,
7233 ["Kscr"] = 119974,
7234 ["Nscr"] = 119977,
7235 ["Oscr"] = 119978,
7236 ["Pscr"] = 119979,
7237 ["Qscr"] = 119980,
7238 ["Sscr"] = 119982,
7239 ["Tscr"] = 119983,
7240 ["Uscr"] = 119984,
7241 ["Vscr"] = 119985,
7242 ["Wscr"] = 119986,
7243 ["Xscr"] = 119987,
7244 ["Yscr"] = 119988,
7245 ["Zscr"] = 119989,
7246 ["ascr"] = 119990,
7247 ["bscr"] = 119991,
7248 ["cscr"] = 119992,

```

```

7249 ["dscr"] = 119993,
7250 ["fscr"] = 119995,
7251 ["hscr"] = 119997,
7252 ["iscr"] = 119998,
7253 ["jscr"] = 119999,
7254 ["kscr"] = 120000,
7255 ["lscr"] = 120001,
7256 ["mscr"] = 120002,
7257 ["nscr"] = 120003,
7258 ["pscr"] = 120005,
7259 ["qscr"] = 120006,
7260 ["rscr"] = 120007,
7261 ["sscr"] = 120008,
7262 ["tscr"] = 120009,
7263 ["uscr"] = 120010,
7264 ["vscr"] = 120011,
7265 ["wscr"] = 120012,
7266 ["xscr"] = 120013,
7267 ["yscr"] = 120014,
7268 ["zscr"] = 120015,
7269 ["Afr"] = 120068,
7270 ["Bfr"] = 120069,
7271 ["Dfr"] = 120071,
7272 ["Efr"] = 120072,
7273 ["Ffr"] = 120073,
7274 ["Gfr"] = 120074,
7275 ["Jfr"] = 120077,
7276 ["Kfr"] = 120078,
7277 ["Lfr"] = 120079,
7278 ["Mfr"] = 120080,
7279 ["Nfr"] = 120081,
7280 ["Ofr"] = 120082,
7281 ["Pfr"] = 120083,
7282 ["Qfr"] = 120084,
7283 ["Sfr"] = 120086,
7284 ["Tfr"] = 120087,
7285 ["Ufr"] = 120088,
7286 ["Vfr"] = 120089,
7287 ["Wfr"] = 120090,
7288 ["Xfr"] = 120091,
7289 ["Yfr"] = 120092,
7290 ["afr"] = 120094,
7291 ["bfr"] = 120095,
7292 ["cfr"] = 120096,
7293 ["dfr"] = 120097,
7294 ["efr"] = 120098,
7295 ["ffr"] = 120099,

```

```

7296 ["gfr"] = 120100,
7297 ["hfr"] = 120101,
7298 ["ifr"] = 120102,
7299 ["jfr"] = 120103,
7300 ["kfr"] = 120104,
7301 ["lfr"] = 120105,
7302 ["mfr"] = 120106,
7303 ["nfr"] = 120107,
7304 ["ofr"] = 120108,
7305 ["pfr"] = 120109,
7306 ["qfr"] = 120110,
7307 ["rfr"] = 120111,
7308 ["sfr"] = 120112,
7309 ["tfr"] = 120113,
7310 ["ufr"] = 120114,
7311 ["vfr"] = 120115,
7312 ["wfr"] = 120116,
7313 ["xfr"] = 120117,
7314 ["yfr"] = 120118,
7315 ["zfr"] = 120119,
7316 ["Aopf"] = 120120,
7317 ["Bopf"] = 120121,
7318 ["Dopf"] = 120123,
7319 ["Eopf"] = 120124,
7320 ["Fopf"] = 120125,
7321 ["Gopf"] = 120126,
7322 ["Iopf"] = 120128,
7323 ["Jopf"] = 120129,
7324 ["Kopf"] = 120130,
7325 ["Lopf"] = 120131,
7326 ["Mopf"] = 120132,
7327 ["Oopf"] = 120134,
7328 ["Sopf"] = 120138,
7329 ["Topf"] = 120139,
7330 ["Uopf"] = 120140,
7331 ["Vopf"] = 120141,
7332 ["Wopf"] = 120142,
7333 ["Xopf"] = 120143,
7334 ["Yopf"] = 120144,
7335 ["aopf"] = 120146,
7336 ["bopf"] = 120147,
7337 ["copf"] = 120148,
7338 ["dopf"] = 120149,
7339 ["eopf"] = 120150,
7340 ["fopf"] = 120151,
7341 ["gopf"] = 120152,
7342 ["hopf"] = 120153,

```

```

7343 ["iopf"] = 120154,
7344 ["jopf"] = 120155,
7345 ["kopf"] = 120156,
7346 ["lopf"] = 120157,
7347 ["mopf"] = 120158,
7348 ["nopf"] = 120159,
7349 ["oopf"] = 120160,
7350 ["popf"] = 120161,
7351 ["qopf"] = 120162,
7352 ["ropf"] = 120163,
7353 ["sopf"] = 120164,
7354 ["topf"] = 120165,
7355 ["uopf"] = 120166,
7356 ["vopf"] = 120167,
7357 ["wopf"] = 120168,
7358 ["xopf"] = 120169,
7359 ["yopf"] = 120170,
7360 ["zopf"] = 120171,
7361 }

```

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

```

7362 function entities.dec_entity(s)
7363   local n = tonumber(s)
7364   if n == nil then
7365     return "&#" .. s .. ";" -- fallback for unknown entities
7366   end
7367   return utf8.char(n)
7368 end

```

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

```

7369 function entities.hex_entity(s)
7370   local n = tonumber("0x"..s)
7371   if n == nil then
7372     return "&#x" .. s .. ";" -- fallback for unknown entities
7373   end
7374   return utf8.char(n)
7375 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.

```

7376 function entities.hex_entity_with_x_char(x, s)
7377   local n = tonumber("0x"..s)
7378   if n == nil then
7379     return "&#" .. x .. s .. ";" -- fallback for unknown entities
7380   end

```

```

7381     return utf8.char(n)
7382 end

```

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

```

7383 function entities.char_entity(s)
7384     local code_points = character_entities[s]
7385     if code_points == nil then
7386         return "&" .. s .. ";"
7387     end
7388     if type(code_points) ~= 'table' then
7389         code_points = {code_points}
7390     end
7391     local char_table = {}
7392     for _, code_point in ipairs(code_points) do
7393         table.insert(char_table, utf8.char(code_point))
7394     end
7395     return table.concat(char_table)
7396 end

```

### 3.1.4 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.

```

7397 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.

```

7398 local parsers
7399 function M.writer.new(options)
7400     local self = {}

```

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

```

7401     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.

```
7402 self.flatten_inlines = false
```

#### 3.1.4.1 Slicing

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

```
7403 local slice_specifiers = {}
7404 for specifier in options.slice:gmatch("[~$s]+") do
7405     table.insert(slice_specifiers, specifier)
7406 end
7407
7408 if #slice_specifiers == 2 then
7409     self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
7410     local slice_begin_type = self.slice_begin:sub(1, 1)
7411     if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
7412         self.slice_begin = "^" .. self.slice_begin
7413     end
7414     local slice_end_type = self.slice_end:sub(1, 1)
7415     if slice_end_type ~= "^" and slice_end_type ~= "$" then
7416         self.slice_end = "$" .. self.slice_end
7417     end
7418 elseif #slice_specifiers == 1 then
7419     self.slice_begin = "^" .. slice_specifiers[1]
7420     self.slice_end = "$" .. slice_specifiers[1]
7421 end
7422
7423 self.slice_begin_type = self.slice_begin:sub(1, 1)
7424 self.slice_begin_identifier = self.slice_begin:sub(2) or ""
7425 self.slice_end_type = self.slice_end:sub(1, 1)
7426 self.slice_end_identifier = self.slice_end:sub(2) or ""
7427
7428 if self.slice_begin == "^" and self.slice_end ~= "^" then
7429     self.is_writing = true
7430 else
7431     self.is_writing = false
7432 end
```

#### 3.1.4.2 Basic Formatter Variables and Functions

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

```
7433 self.space = " "
```

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

```
7434 self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
7435 function self.plain(s)
7436     return s
7437 end
```

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

```
7438 function self.paragraph(s)
7439     if not self.is_writing then return "" end
7440     return s
7441 end
```

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

```
7442 self.interblocksep_text = "\\markdownRendererInterblockSeparator\n{"
7443 function self.interblocksep()
7444     if not self.is_writing then return "" end
7445     return self.interblocksep_text
7446 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.

```
7447 self.paragraphsep_text = "\\markdownRendererParagraphSeparator\n{"
7448 function self.paragraphsep()
7449     if not self.is_writing then return "" end
7450     return self.paragraphsep_text
7451 end
```

Define `writer->undosep` as a function that will remove the output produced by an immediately preceding block element / paragraph separator.

```
7452 self.undosep_text = "\\markdownRendererUndoSeparator\n{"
7453 function self.undosep()
7454     if not self.is_writing then return "" end
7455     return self.undosep_text
7456 end
```

Define `writer->soft_line_break` as the output format of a soft line break.

```
7457 self.soft_line_break = function()
7458     if self.flatten_inlines then return "\n" end
7459     return "\\markdownRendererSoftLineBreak\n{"
7460 end
```

Define `writer->hard_line_break` as the output format of a hard line break.

```
7461 self.hard_line_break = function()
7462     if self.flatten_inlines then return "\n" end
7463     return "\\markdownRendererHardLineBreak\n{"
7464 end
```



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

```
7465 self.ellipsis = "\\markdownRendererEllipsis{}
```

Define `writer->thematic_break` as the output format of a thematic break.

```
7466 function self.thematic_break()
7467   if not self.is_writing then return "" end
7468   return "\\markdownRendererThematicBreak{"
7469 end
```

### 3.1.4.3 Escaping Special Characters

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.

```
7470 self.escaped_uri_chars = {
7471   ["{"] = "\\markdownRendererLeftBrace{",
7472   ["}"] = "\\markdownRendererRightBrace{",
7473   ["\\"] = "\\markdownRendererBackslash{",
7474   ["\r"] = " ",
7475   ["\n"] = " ",
7476 }
7477 self.escaped_minimal_strings = {
7478   ["^"] = "\\markdownRendererCircumflex",
7479   .. "\\markdownRendererCircumflex ",
7480   ["☑"] = "\\markdownRendererTickedBox{",
7481   ["◻"] = "\\markdownRendererHalfTickedBox{",
7482   ["□"] = "\\markdownRendererUntickedBox{",
7483   [entities.hex_entity('FFFD')]
7484     = "\\markdownRendererReplacementCharacter{",
7485 }
```

Define table `writer->escaped_strings` containing the mapping from character strings that need to be escaped in typeset content.

```
7486 self.escaped_strings = util.table_copy(self.escaped_minimal_strings)
7487 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 ConTeXt) that need to be escaped in typeset content.

```
7488 self.escaped_chars = {
7489   ["{"] = "\\markdownRendererLeftBrace{",
7490   ["}"] = "\\markdownRendererRightBrace{",
7491   ["%"] = "\\markdownRendererPercentSign{",
7492   ["\\"] = "\\markdownRendererBackslash{",
7493   ["#"] = "\\markdownRendererHash{",
7494   ["$"] = "\\markdownRendererDollarSign{",
7495   ["&"] = "\\markdownRendererAmpersand{",
7496   ["_"] = "\\markdownRendererUnderscore{",
```

```

7497     ["^"] = "\\markdownRendererCircumflex{}",
7498     ["~"] = "\\markdownRendererTilde{}",
7499     ["|"] = "\\markdownRendererPipe{}",
7500     [entities.hex_entity('0000')]
7501     = "\\markdownRendererReplacementCharacter{}",
7502 }

```

Use the `writer->escaped_chars`, `writer->escaped_uri_chars`, and `writer->escaped_minimal_` tables to create the `escape_typographic_text`, `escape_programmatic_text`, and `escape_minimal` local escaper functions.

```

7503 local function create_escaper(char_escapes, string_escapes)
7504     local escape = util.escaper(char_escapes, string_escapes)
7505     return function(s)
7506         if self.flatten_inlines then return s end
7507         return escape(s)
7508     end
7509 end
7510 local escape_typographic_text = create_escaper(
7511     self.escaped_chars, self.escaped_strings)
7512 local escape_programmatic_text = create_escaper(
7513     self.escaped_uri_chars, self.escaped_minimal_strings)
7514 local escape_minimal = create_escaper(
7515     {}, 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.

```

7516 self.escape = escape_typographic_text
7517 self.math = escape_minimal
7518 if options.hybrid then
7519     self.identifier = escape_minimal
7520     self.string = escape_minimal
7521     self.uri = escape_minimal
7522     self.infostring = escape_minimal
7523 else
7524     self.identifier = escape_programmatic_text
7525     self.string = escape_typographic_text
7526     self.uri = escape_programmatic_text
7527     self.infostring = escape_programmatic_text
7528 end

```

#### 3.1.4.4 Formatters of Warnings and Errors

Define `writer->warning` as a function that will transform an input warning `t` with optional more warning text `m` to the output format.

```
7529 function self.warning(t, m)
7530   return {"\\markdownRendererWarning{" , self.escape(t), "}{" ,
7531         escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
7532         escape_minimal(m or ""), "}"}
7533 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.

```
7534 function self.error(t, m)
7535   return {"\\markdownRendererError{" , self.escape(t), "}{" ,
7536         escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
7537         escape_minimal(m or ""), "}"}
7538 end
```

#### 3.1.4.5 Formatter of Code Spans

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

```
7539 function self.code(s, attributes)
7540   if self.flatten_inlines then return s end
7541   local buf = {}
7542   if attributes ~= nil then
7543     table.insert(buf,
7544       "\\markdownRendererCodeSpanAttributeContextBegin\n")
7545     table.insert(buf, self.attributes(attributes))
7546   end
7547   table.insert(buf,
7548     {"\\markdownRendererCodeSpan{" , self.escape(s), "}"}
7549   if attributes ~= nil then
7550     table.insert(buf,
7551       "\\markdownRendererCodeSpanAttributeContextEnd{")
7552   end
7553   return buf
7554 end
```

#### 3.1.4.6 Formatter of Hyperlinks

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.

```
7555 function self.link(lab, src, tit, attributes)
7556   if self.flatten_inlines then return lab end
7557   local buf = {}
7558   if attributes ~= nil then
```

```

7559     table.insert(buf,
7560         "\\markdownRendererLinkAttributeContextBegin\n")
7561     table.insert(buf, self.attributes(attributes))
7562 end
7563 table.insert(buf, {"\\markdownRendererLink{",lab,"}",
7564     "{",self.escape(src),"}",
7565     "{",self.uri(src),"}",
7566     "{",self.string(tit or ""),"}}"})
7567 if attributes ~= nil then
7568     table.insert(buf,
7569         "\\markdownRendererLinkAttributeContextEnd{ }")
7570 end
7571 return buf
7572 end

```

### 3.1.4.7 Formatter of Images

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.

```

7573 function self.image(lab, src, tit, attributes)
7574     if self.flatten_inlines then return lab end
7575     local buf = {}
7576     if attributes ~= nil then
7577         table.insert(buf,
7578             "\\markdownRendererImageAttributeContextBegin\n")
7579         table.insert(buf, self.attributes(attributes))
7580     end
7581     table.insert(buf, {"\\markdownRendererImage{",lab,"}",
7582         "{",self.string(src),"}",
7583         "{",self.uri(src),"}",
7584         "{",self.string(tit or ""),"}}"})
7585     if attributes ~= nil then
7586         table.insert(buf,
7587             "\\markdownRendererImageAttributeContextEnd{ }")
7588     end
7589     return buf
7590 end

```

### 3.1.4.8 Formatters of Lists

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.

```

7591 function self.bulletlist(items,tight)
7592     if not self.is_writing then return "" end
7593     local buffer = {}

```

```

7594     for _,item in ipairs(items) do
7595         if item ~= "" then
7596             buffer[#buffer + 1] = self.bulletitem(item)
7597         end
7598     end
7599     local contents = util.intersperse(buffer,"\n")
7600     if tight and options.tightLists then
7601         return {"\\markdownRenderU1BeginTight\n",contents,
7602             "\n\\markdownRenderU1EndTight "}
7603     else
7604         return {"\\markdownRenderU1Begin\n",contents,
7605             "\n\\markdownRenderU1End "}
7606     end
7607 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.

```

7608 function self.bulletitem(s)
7609     return {"\\markdownRenderU1Item ",s,
7610         "\\markdownRenderU1ItemEnd "}
7611 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.

```

7612 function self.orderedlist(items,tight,startnum)
7613     if not self.is_writing then return "" end
7614     local buffer = {}
7615     local num = startnum
7616     for _,item in ipairs(items) do
7617         if item ~= "" then
7618             buffer[#buffer + 1] = self.ordereditem(item,num)
7619         end
7620         if num ~= nil and item ~= "" then
7621             num = num + 1
7622         end
7623     end
7624     local contents = util.intersperse(buffer,"\n")
7625     if tight and options.tightLists then
7626         return {"\\markdownRenderO1BeginTight\n",contents,
7627             "\n\\markdownRenderO1EndTight "}
7628     else
7629         return {"\\markdownRenderO1Begin\n",contents,
7630             "\n\\markdownRenderO1End "}
7631     end
7632 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.

```

7633 function self.ordereditem(s,num)
7634   if num ~= nil then
7635     return {"\\markdownRendererOlItemWithNumber{" ,num,"} ",s,
7636           "\\markdownRendererOlItemEnd "}
7637   else
7638     return {"\\markdownRendererOlItem ",s,
7639           "\\markdownRendererOlItemEnd "}
7640   end
7641 end

```

#### 3.1.4.9 Formatters of HTML Tags, Elements, and Comments

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.

```

7642 function self.inline_html_comment(contents)
7643   if self.flatten_inlines then return contents end
7644   return {"\\markdownRendererInlineHtmlComment{" ,contents,"}"}
7645 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.

```

7646 function self.inline_html_tag(contents)
7647   if self.flatten_inlines then return contents end
7648   return {"\\markdownRendererInlineHtmlTag{" ,
7649         self.string(contents),"}"}
7650 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.

```

7651 function self.block_html_element(s)
7652   if not self.is_writing then return "" end
7653   local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
7654   return {"\\markdownRendererInputBlockHtmlElement{" ,name,"}"}
7655 end

```

#### 3.1.4.10 Formatter of Emphasis

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

```

7656 function self.emphasis(s)
7657   if self.flatten_inlines then return s end

```

```

7658     return {"\\markdownRendererEmphasis{" ,s,""} }
7659 end

```

#### 3.1.4.11 Formatter of Strong Emphasis

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

```

7660 function self.strong(s)
7661     if self.flatten_inlines then return s end
7662     return {"\\markdownRendererStrongEmphasis{" ,s,""} }
7663 end

```

#### 3.1.4.12 Formatter of Tickboxes

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

```

7664 function self.tickbox(f)
7665     if f == 1.0 then
7666         return "☒ "
7667     elseif f == 0.0 then
7668         return "☐ "
7669     else
7670         return "◻ "
7671     end
7672 end

```

#### 3.1.4.13 Formatter of Blockquotes

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

```

7673 function self.blockquote(s)
7674     if not self.is_writing then return "" end
7675     return {"\\markdownRendererBlockQuoteBegin\n",s,
7676         "\\markdownRendererBlockQuoteEnd "}
7677 end

```

#### 3.1.4.14 Formatter of Code Blocks

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

```

7678 function self.verbatim(s)
7679     if not self.is_writing then return "" end
7680     s = s:gsub("\n$", "")
7681     local name = util.cache_verbatim(options.cacheDir, s)
7682     return {"\\markdownRendererInputVerbatim{" ,name,""} }
7683 end

```

### 3.1.4.15 Formatter of Documents

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

```
7684 function self.document(d)
7685   local buf = {"\\markdownRendererDocumentBegin\n"}
7686
7687   -- warn against the `hybrid` option
7688   if options.hybrid then
7689     local text = "The `hybrid` option has been soft-deprecated."
7690     local more = "Consider using one of the following better options "
7691       .. "for mixing TeX and markdown: `contentBlocks`, "
7692       .. "`rawAttribute`, `texComments`, `texMathDollars`, "
7693       .. "`texMathSingleBackslash`, and "
7694       .. "`texMathDoubleBackslash`. "
7695     .. "For more information, see the user manual at "
7696     .. "<https://witiko.github.io/markdown/>."
7697     table.insert(buf, self.warning(text, more))
7698   end
7699
7700   -- insert the text of the document
7701   table.insert(buf, d)
7702
7703   -- pop all attributes
7704   table.insert(buf, self.pop_attributes())
7705
7706   table.insert(buf, "\\markdownRendererDocumentEnd")
7707
7708   return buf
7709 end
```

### 3.1.4.16 Formatter of Attributes

Define `writer->attributes` as a function that will transform input attributes `attrs` to the output format.

```
7710 local seen_identifiers = {}
7711 local key_value_regex = "([^\s= ]+)%s*=%s*(.*)"
7712 local function normalize_attributes(attributes, auto_identifiers)
7713   -- normalize attributes
7714   local normalized_attributes = {}
7715   local has_explicit_identifiers = false
7716   local key, value
7717   for _, attribute in ipairs(attributes or {}) do
7718     if attribute:sub(1, 1) == "#" then
7719       table.insert(normalized_attributes, attribute)
7720       has_explicit_identifiers = true
7721       seen_identifiers[attribute:sub(2)] = true
7722     elseif attribute:sub(1, 1) == "." then
```



```

7723         table.insert(normalized_attributes, attribute)
7724     else
7725         key, value = attribute:match(key_value_regex)
7726         if key:lower() == "id" then
7727             table.insert(normalized_attributes, "#" .. value)
7728         elseif key:lower() == "class" then
7729             local classes = {}
7730             for class in value:gmatch("%S+") do
7731                 table.insert(classes, class)
7732             end
7733             table.sort(classes)
7734             for _, class in ipairs(classes) do
7735                 table.insert(normalized_attributes, "." .. class)
7736             end
7737         else
7738             table.insert(normalized_attributes, attribute)
7739         end
7740     end
7741 end
7742
7743 -- if no explicit identifiers exist, add auto identifiers
7744 if not has_explicit_identifiers and auto_identifiers ~= nil then
7745     local seen_auto_identifiers = {}
7746     for _, auto_identifier in ipairs(auto_identifiers) do
7747         if seen_auto_identifiers[auto_identifier] == nil then
7748             seen_auto_identifiers[auto_identifier] = true
7749             if seen_identifiers[auto_identifier] == nil then
7750                 seen_identifiers[auto_identifier] = true
7751                 table.insert(normalized_attributes,
7752                     "#" .. auto_identifier)
7753             else
7754                 local auto_identifier_number = 1
7755                 while true do
7756                     local numbered_auto_identifier = auto_identifier .. "-"
7757                                             .. auto_identifier_number
7758                     if seen_identifiers[numbered_auto_identifier] == nil then
7759                         seen_identifiers[numbered_auto_identifier] = true
7760                         table.insert(normalized_attributes,
7761                             "#" .. numbered_auto_identifier)
7762                     break
7763                 end
7764                 auto_identifier_number = auto_identifier_number + 1
7765             end
7766         end
7767     end
7768 end
7769 end

```

```

7770
7771     -- sort and deduplicate normalized attributes
7772     table.sort(normalized_attributes)
7773     local seen_normalized_attributes = {}
7774     local deduplicated_normalized_attributes = {}
7775     for _, attribute in ipairs(normalized_attributes) do
7776         if seen_normalized_attributes[attribute] == nil then
7777             seen_normalized_attributes[attribute] = true
7778             table.insert(deduplicated_normalized_attributes, attribute)
7779         end
7780     end
7781
7782     return deduplicated_normalized_attributes
7783 end
7784
7785 function self.attributes(attributes, should_normalize_attributes)
7786     local normalized_attributes
7787     if should_normalize_attributes == false then
7788         normalized_attributes = attributes
7789     else
7790         normalized_attributes = normalize_attributes(attributes)
7791     end
7792
7793     local buf = {}
7794     local key, value
7795     for _, attribute in ipairs(normalized_attributes) do
7796         if attribute:sub(1, 1) == "#" then
7797             table.insert(buf, {"\\markdownRendererAttributeIdentifier{",
7798                               attribute:sub(2), "}"}
7799         elseif attribute:sub(1, 1) == "." then
7800             table.insert(buf, {"\\markdownRendererAttributeName{",
7801                               attribute:sub(2), "}"}
7802         else
7803             key, value = attribute:match(key_value_regex)
7804             table.insert(buf, {"\\markdownRendererAttributeKeyValue{",
7805                               key, "}{" , value, "}"}
7806         end
7807     end
7808
7809     return buf
7810 end

```

#### 3.1.4.17 Tracking Active Attributes

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

```

7811     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`.

```
7812 self.attribute_type_levels = {}
7813 setmetatable(self.attribute_type_levels,
7814               { __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`.

```
7815 local function apply_attributes()
7816     local buf = {}
7817     for i = 1, #self.active_attributes do
7818         local start_output = self.active_attributes[i][3]
7819         if start_output ~= nil then
7820             table.insert(buf, start_output)
7821         end
7822     end
7823     return buf
7824 end
7825
7826 local function tear_down_attributes()
7827     local buf = {}
7828     for i = #self.active_attributes, 1, -1 do
7829         local end_output = self.active_attributes[i][4]
7830         if end_output ~= nil then
7831             table.insert(buf, end_output)
7832         end
7833     end
7834     return buf
7835 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.

```
7836 function self.push_attributes(attribute_type, attributes,
7837                               start_output, end_output)
7838     local attribute_type_level
7839     = self.attribute_type_levels[attribute_type]
7840     self.attribute_type_levels[attribute_type]
7841     = attribute_type_level + 1
7842
7843     -- index attributes in a hash table for easy lookup
7844     attributes = attributes or {}
7845     for i = 1, #attributes do
7846         attributes[attributes[i]] = true
```

```

7847     end
7848
7849     local buf = {}
7850     -- handle slicing
7851     if attributes["#" .. self.slice_end_identifier] ~= nil and
7852        self.slice_end_type == "^" then
7853         if self.is_writing then
7854             table.insert(buf, self.undosep())
7855             table.insert(buf, tear_down_attributes())
7856         end
7857         self.is_writing = false
7858     end
7859     if attributes["#" .. self.slice_begin_identifier] ~= nil and
7860        self.slice_begin_type == "^" then
7861         table.insert(buf, apply_attributes())
7862         self.is_writing = true
7863     end
7864     if self.is_writing and start_output ~= nil then
7865         table.insert(buf, start_output)
7866     end
7867     table.insert(self.active_attributes,
7868                 {attribute_type, attributes,
7869                  start_output, end_output})
7870     return buf
7871 end
7872

```

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`).

```

7873 function self.pop_attributes(attribute_type)
7874     local buf = {}
7875     -- pop attributes until we find attributes of correct type
7876     -- or until no attributes remain
7877     local current_attribute_type = false
7878     while current_attribute_type ~= attribute_type and
7879        #self.active_attributes > 0 do
7880         local attributes, _, end_output
7881         current_attribute_type, attributes, _, end_output = table.unpack(
7882             self.active_attributes[#self.active_attributes])
7883         local attribute_type_level
7884         = self.attribute_type_levels[current_attribute_type]
7885         self.attribute_type_levels[current_attribute_type]
7886         = attribute_type_level - 1

```

```

7887     if self.is_writing and end_output ~= nil then
7888         table.insert(buf, end_output)
7889     end
7890     table.remove(self.active_attributes, #self.active_attributes)
7891     -- handle slicing
7892     if attributes["#" .. self.slice_end_identifier] ~= nil
7893         and self.slice_end_type == "$" then
7894         if self.is_writing then
7895             table.insert(buf, self.undosep())
7896             table.insert(buf, tear_down_attributes())
7897         end
7898         self.is_writing = false
7899     end
7900     if attributes["#" .. self.slice_begin_identifier] ~= nil and
7901         self.slice_begin_type == "$" then
7902         self.is_writing = true
7903         table.insert(buf, apply_attributes())
7904     end
7905 end
7906 return buf
7907 end

```

#### 3.1.4.18 Automatically Generated Identifiers for Headings

Create an auto identifier string by stripping and converting characters from string `s`.

```

7908 local function create_auto_identifier(s)
7909     local buffer = {}
7910     local prev_space = false
7911     local letter_found = false
7912     local normalized_s = s
7913     if not options.unicodeNormalization
7914         or options.unicodeNormalizationForm ~= "nfc" then
7915         normalized_s = util.normalize(normalized_s, "nfc")
7916     end
7917
7918     for _, code in utf8.codes(normalized_s) do
7919         local char = utf8.char(code)
7920
7921         -- Remove everything up to the first letter.
7922         if not letter_found then
7923             local is_letter = lpeg.match(
7924                 parsers.unicode.following_alpha,
7925                 char
7926             )
7927             if is_letter then
7928                 letter_found = true

```

```

7929         else
7930             goto continue
7931         end
7932     end
7933
7934     -- Remove all non-alphanumeric characters, except underscores,
7935     -- hyphens, and periods.
7936     if not lpeg.match(
7937         ( parsers.underscore
7938         + parsers.dash
7939         + parsers.period
7940         + parsers.unicode.following_word
7941         + parsers.unicode.following_whitespace ),
7942         char
7943     ) then
7944         goto continue
7945     end
7946
7947     -- Replace all spaces and newlines with hyphens.
7948     if lpeg.match(
7949         ( parsers.newline
7950         + parsers.unicode.following_whitespace ),
7951         char
7952     ) then
7953         char = "-"
7954         if prev_space then
7955             goto continue
7956         else
7957             prev_space = true
7958         end
7959     else
7960         -- Case-fold all alphabetic characters.
7961         local form = nil
7962         if options.unicodeNormalization then
7963             form = options.unicodeNormalizationForm
7964         end
7965         char = util.casefold(char, form)
7966         prev_space = false
7967     end
7968
7969     table.insert(buffer, char)
7970
7971     ::continue::
7972 end
7973
7974 if prev_space then
7975     table.remove(buffer)

```

```

7976     end
7977
7978     local identifier = #buffer == 0 and "section"
7979                     or table.concat(buffer, "")
7980     return identifier
7981 end

```

Create an GitHub-flavored auto identifier string by stripping and converting characters from string `s`.

```

7982 local function create_gfm_auto_identifier(s)
7983     local buffer = {}
7984     local prev_space = false
7985     local letter_found = false
7986     local normalized_s = s
7987     if not options.unicodeNormalization
7988         or options.unicodeNormalizationForm ~= "nfc" then
7989         normalized_s = util.normalize(normalized_s, "nfc")
7990     end
7991
7992     for _, code in utf8.codes(normalized_s) do
7993         local char = utf8.char(code)
7994
7995         -- Remove everything up to the first non-space.
7996         if not letter_found then
7997             local is_letter = not lpeg.match(
7998                 parsers.unicode.following_whitespace,
7999                 char
8000             )
8001             if is_letter then
8002                 letter_found = true
8003             else
8004                 goto continue
8005             end
8006         end
8007
8008         -- Remove all non-alphanumeric characters, except underscores
8009         -- and hyphens.
8010         if not lpeg.match(
8011             ( parsers.underscore
8012               + parsers.dash
8013               + parsers.unicode.following_word
8014               + parsers.unicode.following_whitespace ),
8015             char
8016         ) then
8017             prev_space = false
8018             goto continue
8019         end
8019     end

```

```

8020
8021      -- Replace all spaces and newlines with hyphens.
8022      if lpeg.match(
8023          ( parsers.newline
8024            + parsers.unicode.following_whitespace ),
8025          char
8026      ) then
8027          char = "-"
8028          if prev_space then
8029              goto continue
8030          else
8031              prev_space = true
8032          end
8033      else
8034          -- Case-fold all alphabetic characters.
8035          local form = nil
8036          if options.unicodeNormalization then
8037              form = options.unicodeNormalizationForm
8038          end
8039          char = util.casefold(char, form)
8040          prev_space = false
8041      end
8042
8043      table.insert(buffer, char)
8044
8045      ::continue::
8046  end
8047
8048  if prev_space then
8049      table.remove(buffer)
8050  end
8051
8052  local identifier = #buffer == 0 and "section"
8053                  or table.concat(buffer, "")
8054  return identifier
8055  end

```

### 3.1.4.19 Formatter of Headings

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

```

8056  self.secbegin_text = "\\markdownRendererSectionBegin\n"
8057  self.secend_text = "\\n\\markdownRendererSectionEnd "
8058  function self.heading(s, level, attributes)
8059      local buf = {}
8060      local flat_text, inlines = table.unpack(s)
8061

```



```

8062 -- push empty attributes for implied sections
8063 while self.attribute_type_levels["heading"] < level - 1 do
8064     table.insert(buf,
8065                 self.push_attributes("heading",
8066                                     nil,
8067                                     self.secbegin_text,
8068                                     self.secend_text))
8069 end
8070
8071 -- pop attributes for sections that have ended
8072 while self.attribute_type_levels["heading"] >= level do
8073     table.insert(buf, self.pop_attributes("heading"))
8074 end
8075
8076 -- construct attributes for the new section
8077 local auto_identifiers = {}
8078 if self.options.autoIdentifiers then
8079     table.insert(auto_identifiers, create_auto_identifier(flat_text))
8080 end
8081 if self.options.gfmAutoIdentifiers then
8082     table.insert(auto_identifiers,
8083                 create_gfm_auto_identifier(flat_text))
8084 end
8085 local normalized_attributes = normalize_attributes(attributes,
8086                                                    auto_identifiers)
8087
8088 -- push attributes for the new section
8089 local start_output = {}
8090 local end_output = {}
8091 table.insert(start_output, self.secbegin_text)
8092 table.insert(end_output, self.secend_text)
8093
8094 table.insert(buf, self.push_attributes("heading",
8095                                       normalized_attributes,
8096                                       start_output,
8097                                       end_output))
8098 assert(self.attribute_type_levels["heading"] == level)
8099
8100 -- render the heading and its attributes
8101 if self.is_writing and #normalized_attributes > 0 then
8102     table.insert(buf,
8103                 "\\markdownRendererHeaderAttributeContextBegin\n")
8104     table.insert(buf, self.attributes(normalized_attributes, false))
8105 end
8106
8107 local cmd
8108 level = level + options.shiftHeadings

```

```

8109     if level <= 1 then
8110         cmd = "\\markdownRendererHeadingOne"
8111     elseif level == 2 then
8112         cmd = "\\markdownRendererHeadingTwo"
8113     elseif level == 3 then
8114         cmd = "\\markdownRendererHeadingThree"
8115     elseif level == 4 then
8116         cmd = "\\markdownRendererHeadingFour"
8117     elseif level == 5 then
8118         cmd = "\\markdownRendererHeadingFive"
8119     elseif level >= 6 then
8120         cmd = "\\markdownRendererHeadingSix"
8121     else
8122         cmd = ""
8123     end
8124     if self.is_writing then
8125         table.insert(buf, {cmd, "{", inlines, "}"})
8126     end
8127
8128     if self.is_writing and #normalized_attributes > 0 then
8129         table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd{")
8130     end
8131
8132     return buf
8133 end

```

#### 3.1.4.20 Managing State and Deferred Writer Calls

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.

```

8134 function self.get_state()
8135     return {
8136         is_writing=self.is_writing,
8137         flatten_inlines=self.flatten_inlines,
8138         active_attributes={table.unpack(self.active_attributes)},
8139     }
8140 end

```

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

```

8141 function self.set_state(s)
8142     local previous_state = self.get_state()
8143     for key, value in pairs(s) do
8144         self[key] = value
8145     end
8146     return previous_state
8147 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`.

```

8148 function self.defer_call(f)
8149     local previous_state = self.get_state()
8150     return function(...)
8151         local state = self.set_state(previous_state)
8152         local return_value = f(...)
8153         self.set_state(state)
8154         return return_value
8155     end
8156 end
8157
8158 return self
8159 end

```

### 3.1.5 Parsers

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

```

8160 parsers = {}

```

#### 3.1.5.1 Basic Parsers

```

8161 parsers.percent = P("%")
8162 parsers.at = P("@")
8163 parsers.comma = P(",")
8164 parsers.asterisk = P("*")
8165 parsers.dash = P("-")
8166 parsers.plus = P("+")
8167 parsers.underscore = P("_")
8168 parsers.period = P(".")
8169 parsers.hash = P("#")
8170 parsers.dollar = P("$")
8171 parsers.ampersand = P("&")
8172 parsers.backtick = P("`")
8173 parsers.less = P("<")
8174 parsers.more = P(">")
8175 parsers.space = P(" ")
8176 parsers.squote = P("'")
8177 parsers.dquote = P('"')
8178 parsers.lparent = P("(")
8179 parsers.rparent = P(")")
8180 parsers.lbracket = P("[")
8181 parsers.rbracket = P("]")
8182 parsers.lbrace = P("{")

```

```

8183 parsers.rbrace          = P("}")
8184 parsers.circumflex       = P("^")
8185 parsers.slash            = P("/")
8186 parsers.equal            = P("=")
8187 parsers.colon            = P(":")
8188 parsers.semicolon        = P(";")
8189 parsers.exclamation      = P("!")
8190 parsers.pipe             = P("|")
8191 parsers.tilde            = P("~")
8192 parsers.backslash        = P("\\")
8193 parsers.tab              = P("\t")
8194 parsers.newline          = P("\n")
8195
8196 parsers.digit            = R("09")
8197 parsers.hexdigit         = R("09","af","AF")
8198 parsers.letter          = R("AZ","az")
8199 parsers.alphanumeric     = R("AZ","az","09")
8200 parsers.keyword          = parsers.letter
8201                          * (parsers.alphanumeric + parsers.dash)^0
8202
8203 parsers.doubleasterisks  = P("**")
8204 parsers.doubleunderscores = P("__")
8205 parsers.doubletildes     = P("~")
8206 parsers.fourspace       = P("    ")
8207
8208 parsers.any              = P(1)
8209 parsers.succeed          = P(true)
8210 parsers.fail             = P(false)
8211
8212 parsers.internal_punctuation = S(";,.,?")
8213 parsers.ascii_punctuation  = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")
8214
8215 parsers.escapable        = parsers.ascii_punctuation
8216 parsers.anyescaped       = parsers.backslash / ""
8217                          * parsers.escapable
8218                          + parsers.any
8219
8220 parsers.spacechar        = S("\t ")
8221 parsers.spacing          = S(" \n\r\t")
8222 parsers.nonspacechar     = parsers.any - parsers.spacing
8223 parsers.optionalspace    = parsers.spacechar^0
8224
8225 parsers.normalchar       = parsers.any - (V("SpecialChar")
8226                                     + parsers.spacing)
8227 parsers.eof              = -parsers.any
8228 parsers.nonindentSPACE   = parsers.space^3 * - parsers.spacechar
8229 parsers.indent           = parsers.space^3 * parsers.tab

```

```

8230                                     + parsers.fourspace / ""
8231 parsers.linechar                     = P(1 - parsers.newline)
8232
8233 parsers.blankline                    = parsers.optionalspace
8234                                     * parsers.newline / "\n"
8235 parsers.blanklines                    = parsers.blankline^0
8236 parsers.skipblanklines                = ( parsers.optionalspace
8237                                     * parsers.newline)^0
8238 parsers.indentedline                  = parsers.indent / ""
8239                                     * C( parsers.linechar^1
8240                                     * parsers.newline^~1)
8241 parsers.optionallyindentedline        = parsers.indent^~1 / ""
8242                                     * C( parsers.linechar^1
8243                                     * parsers.newline^~1)
8244 parsers.sp                            = parsers.spacing^0
8245 parsers.spnl                          = parsers.optionalspace
8246                                     * ( parsers.newline
8247                                     * parsers.optionalspace)^~1
8248 parsers.line                          = parsers.linechar^0 * parsers.newline
8249 parsers.nonemptyline                  = parsers.line - parsers.blankline
8250

```

### 3.1.5.2 Parsers for Unicode Character Classes and Categories

We define high-level parsers in table `parsers.unicode` based on the low-level parsers in `unicode_data.categories`, defined in Section 3.1.1.5. Unlike the low-level parsers, the high-level parsers are invariant to the number of bytes the Unicode characters occupy after conversion to UTF-8.

```

8251 parsers.unicode = {}
8252 parsers.unicode.preceding_punctuation = parsers.fail
8253 parsers.unicode.following_punctuation = parsers.fail
8254 parsers.unicode.following_alpha = parsers.fail
8255 parsers.unicode.following_word = parsers.fail
8256 parsers.unicode.preceding_whitespace = parsers.fail
8257 parsers.unicode.following_whitespace = parsers.fail
8258 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>.

```

8259     local punctuation_of_length_n
8260     = unicode_data.categories.P[n]
8261     + unicode_data.categories.S[n]
8262     parsers.unicode.preceding_punctuation
8263     = parsers.unicode.preceding_punctuation
8264     + B(punctuation_of_length_n)
8265     parsers.unicode.following_punctuation

```

<sup>35</sup>See <https://spec.commonmark.org/0.31.2/#unicode-punctuation-character>.

```

8266     = parsers.unicode.following_punctuation
8267     + #punctuation_of_length_n

```

For alphabetical characters, accept any characters from Unicode category L (letter), similar to the character class ‘Unicode.

```

8268     local alpha_of_length_n = unicode_data.categories.L[n]
8269     parsers.unicode.following_alpha
8270     = parsers.unicode.following_alpha
8271     + 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 ‘

```

8272     local word_of_length_n
8273     = unicode_data.categories.L[n]
8274     + unicode_data.categories.N[n]
8275     + unicode_data.categories.Pc[n]
8276     parsers.unicode.following_word
8277     = parsers.unicode.following_word
8278     + 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.

```

8279     local whitespace_of_length_n = unicode_data.categories.Z[n]
8280     if n == 1 then
8281         whitespace_of_length_n
8282         = whitespace_of_length_n
8283         + R("\t\r")
8284     end
8285     parsers.unicode.preceding_whitespace
8286     = parsers.unicode.preceding_whitespace
8287     + B(whitespace_of_length_n)
8288     parsers.unicode.following_whitespace
8289     = parsers.unicode.following_whitespace
8290     + #whitespace_of_length_n
8291 end

```

### 3.1.5.3 Parsers Used for Indentation

```

8292
8293 parsers.leader      = parsers.space^-3
8294

```

Check if a trail exists and is non-empty in the indent table `indent_table`.

```

8295 local function has_trail(indent_table)
8296     return indent_table ~= nil and
8297         indent_table.trail ~= nil and
8298         next(indent_table.trail) ~= nil
8299 end

```

8300

Check if indent table `indent_table` has any indents.

```
8301 local function has_indents(indent_table)
8302   return indent_table ~= nil and
8303     indent_table.indents ~= nil and
8304     next(indent_table.indents) ~= nil
8305 end
8306
```

Add a trail `trail_info` to the indent table `indent_table`.

```
8307 local function add_trail(indent_table, trail_info)
8308   indent_table.trail = trail_info
8309   return indent_table
8310 end
8311
```

Remove a trail `trail_info` from the indent table `indent_table`.

```
8312 local function remove_trail(indent_table)
8313   indent_table.trail = nil
8314   return indent_table
8315 end
8316
```

Update the indent table `indent_table` by adding or removing a new indent `add`.

```
8317 local function update_indent_table(indent_table, new_indent, add)
8318   indent_table = remove_trail(indent_table)
8319
8320   if not has_indents(indent_table) then
8321     indent_table.indents = {}
8322   end
8323
8324
8325   if add then
8326     indent_table.indents[#indent_table.indents + 1] = new_indent
8327   else
8328     if indent_table.indents[#indent_table.indents].name
8329       == new_indent.name then
8330       indent_table.indents[#indent_table.indents] = nil
8331     end
8332   end
8333
8334   return indent_table
8335 end
8336
```

Remove an indent by its name `name`.

```
8337 local function remove_indent(name)
8338   local remove_indent_level =
```

```

8339     function(s, i, indent_table) -- luacheck: ignore s i
8340         indent_table = update_indent_table(indent_table, {name=name},
8341                                           false)
8342         return true, indent_table
8343     end
8344
8345     return Cg(Cmt(Cb("indent_info"), remove_indent_level), "indent_info")
8346 end
8347

```

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`.

```

8348 local function process_starter_spacing(indent, spacing,
8349                                     minimum, left_strip_length)
8350     left_strip_length = left_strip_length or 0
8351
8352     local count = 0
8353     local tab_value = 4 - (indent) % 4
8354
8355     local code_started, minimum_found = false, false
8356     local code_start, minimum_remainder = "", ""
8357
8358     local left_total_stripped = 0
8359     local full_remainder = ""
8360
8361     if spacing ~= nil then
8362         for i = 1, #spacing do
8363             local character = spacing:sub(i, i)
8364
8365             if character == "\t" then
8366                 count = count + tab_value
8367                 tab_value = 4
8368             elseif character == " " then
8369                 count = count + 1
8370                 tab_value = 4 - (1 - tab_value) % 4
8371             end
8372
8373             if (left_strip_length ~= 0) then
8374                 local possible_to_strip = math.min(count, left_strip_length)
8375                 count = count - possible_to_strip
8376                 left_strip_length = left_strip_length - possible_to_strip
8377                 left_total_stripped = left_total_stripped + possible_to_strip
8378             else
8379                 full_remainder = full_remainder .. character

```



```

8380     end
8381
8382     if (minimum_found) then
8383         minimum_remainder = minimum_remainder .. character
8384     elseif (count >= minimum) then
8385         minimum_found = true
8386         minimum_remainder = minimum_remainder
8387             .. string.rep(" ", count - minimum)
8388     end
8389
8390     if (code_started) then
8391         code_start = code_start .. character
8392     elseif (count >= minimum + 4) then
8393         code_started = true
8394         code_start = code_start
8395             .. string.rep(" ", count - (minimum + 4))
8396     end
8397 end
8398 end
8399
8400 local remainder
8401 if (code_started) then
8402     remainder = code_start
8403 else
8404     remainder = string.rep(" ", count - minimum)
8405 end
8406
8407 local is_minimum = count >= minimum
8408 return {
8409     is_code = code_started,
8410     remainder = remainder,
8411     left_total_stripped = left_total_stripped,
8412     is_minimum = is_minimum,
8413     minimum_remainder = minimum_remainder,
8414     total_length = count,
8415     full_remainder = full_remainder
8416 }
8417 end
8418

```

Count the total width of all indents in the indent table [indent\\_table](#).

```

8419 local function count_indent_tab_level(indent_table)
8420     local count = 0
8421     if not has_indents(indent_table) then
8422         return count
8423     end
8424
8425     for i=1, #indent_table.indents do

```

```

8426     count = count + indent_table.indents[i].length
8427 end
8428 return count
8429 end
8430

```

Count the total width of a delimiter `delimiter`.

```

8431 local function total_delimiter_length(delimiter)
8432     local count = 0
8433     if type(delimiter) == "string" then return #delimiter end
8434     for _, value in pairs(delimiter) do
8435         count = count + total_delimiter_length(value)
8436     end
8437     return count
8438 end
8439

```

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`.

```

8440 local function process_starter_indent(_, _, indent_table, starter,
8441                                     is_blank, indent_type, breakable)
8442     local last_trail = starter[1]
8443     local delimiter = starter[2]
8444     local raw_new_trail = starter[3]
8445
8446     if indent_type == "bq" and not breakable then
8447         indent_table.ignore_blockquote_blank = true
8448     end
8449
8450     if has_trail(indent_table) then
8451         local trail = indent_table.trail
8452         if trail.is_code then
8453             return false
8454         end
8455         last_trail = trail.remainder
8456     else
8457         local sp = process_starter_spacing(0, last_trail, 0, 0)
8458
8459         if sp.is_code then
8460             return false
8461         end
8462         last_trail = sp.remainder
8463     end
8464
8465     local preceding_indentation = count_indent_tab_level(indent_table) % 4
8466     local last_trail_length = #last_trail
8467     local delimiter_length = total_delimiter_length(delimiter)
8468

```

```

8469 local total_indent_level = preceding_indentation + last_trail_length
8470                                + delimiter_length
8471
8472 local sp = {}
8473 if not is_blank then
8474     sp = process_starter_spacing(total_indent_level, raw_new_trail,
8475                                0, 1)
8476 end
8477
8478 local del_trail_length = sp.left_total_stripped
8479 if is_blank then
8480     del_trail_length = 1
8481 elseif not sp.is_code then
8482     del_trail_length = del_trail_length + #sp.remainder
8483 end
8484
8485 local indent_length = last_trail_length + delimiter_length
8486                                + del_trail_length
8487 local new_indent_info = {name=indent_type, length=indent_length}
8488
8489 indent_table = update_indent_table(indent_table, new_indent_info,
8490                                   true)
8491 indent_table = add_trail(indent_table,
8492                          {is_code=sp.is_code,
8493                           remainder=sp.remainder,
8494                           total_length=sp.total_length,
8495                           full_remainder=sp.full_remainder})
8496
8497 return true, indent_table
8498 end
8499

```

Return the pattern corresponding with the indent name `name`.

```

8500 local function decode_pattern(name)
8501     local delimiter = parsers.succeed
8502     if name == "bq" then
8503         delimiter = parsers.more
8504     end
8505
8506     return C(parsers.optionalspace) * C(delimiter)
8507           * C(parsers.optionalspace) * Cp()
8508 end
8509

```

Find the first blank-only indent of the indent table `indent_table` followed by blank-only indents.

```

8510 local function left_blank_starter(indent_table)
8511     local blank_starter_index

```

```

8512
8513   if not has_indents(indent_table) then
8514       return
8515   end
8516
8517   for i = #indent_table.indents,1,-1 do
8518       local value = indent_table.indents[i]
8519       if value.name == "li" then
8520           blank_starter_index = i
8521       else
8522           break
8523       end
8524   end
8525
8526   return blank_starter_index
8527 end
8528

```

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.

```

8529 local function traverse_indent(s, i, indent_table, is_optional,
8530                               is_blank, current_line_indents)
8531     local new_index = i
8532
8533     local preceding_indentation = 0
8534     local current_trail = {}
8535
8536     local blank_starter = left_blank_starter(indent_table)
8537
8538     if current_line_indents == nil then
8539         current_line_indents = {}
8540     end
8541
8542     for index = 1,#indent_table.indents do
8543         local value = indent_table.indents[index]
8544         local pattern = decode_pattern(value.name)
8545
8546         -- match decoded pattern
8547         local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
8548         if new_indent_info == nil then
8549             local blankline_end = lpeg.match(
8550                 Ct(parsers.blankline * Cg(Cp(), "pos")), s, new_index)
8551             if is_optional or not indent_table.ignore_blockquote_blank
8552                 or not blankline_end then

```

```

8553         return is_optional, new_index, current_trail,
8554                current_line_indents
8555     end
8556
8557     return traverse_indent(s, tonumber(blankline_end.pos),
8558                           indent_table, is_optional, is_blank,
8559                           current_line_indents)
8560 end
8561
8562 local raw_last_trail = new_indent_info[1]
8563 local delimiter = new_indent_info[2]
8564 local raw_new_trail = new_indent_info[3]
8565 local next_index = new_indent_info[4]
8566
8567 local space_only = delimiter == ""
8568
8569 -- check previous trail
8570 if not space_only and next(current_trail) == nil then
8571     local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
8572     current_trail = {is_code=sp.is_code, remainder=sp.remainder,
8573                     total_length=sp.total_length,
8574                     full_remainder=sp.full_remainder}
8575 end
8576
8577 if next(current_trail) ~= nil then
8578     if not space_only and current_trail.is_code then
8579         return is_optional, new_index, current_trail,
8580                current_line_indents
8581     end
8582     if current_trail.internal_remainder ~= nil then
8583         raw_last_trail = current_trail.internal_remainder
8584     end
8585 end
8586
8587 local raw_last_trail_length = 0
8588 local delimiter_length = 0
8589
8590 if not space_only then
8591     delimiter_length = #delimiter
8592     raw_last_trail_length = #raw_last_trail
8593 end
8594
8595 local total_indent_level = preceding_indentation
8596                          + raw_last_trail_length + delimiter_length
8597
8598 local spacing_to_process
8599 local minimum = 0

```

```

8600     local left_strip_length = 0
8601
8602     if not space_only then
8603         spacing_to_process = raw_new_trail
8604         left_strip_length = 1
8605     else
8606         spacing_to_process = raw_last_trail
8607         minimum = value.length
8608     end
8609
8610     local sp = process_starter_spacing(total_indent_level,
8611                                       spacing_to_process, minimum,
8612                                       left_strip_length)
8613
8614     if space_only and not sp.is_minimum then
8615         return is_optional or (is_blank and blank_starter <= index),
8616             new_index, current_trail, current_line_indents
8617     end
8618
8619     local indent_length = raw_last_trail_length + delimiter_length
8620                       + sp.left_total_stripped
8621
8622     -- update info for the next pattern
8623     if not space_only then
8624         preceding_indentation = preceding_indentation + indent_length
8625     else
8626         preceding_indentation = preceding_indentation + value.length
8627     end
8628
8629     current_trail = {is_code=sp.is_code, remainder=sp.remainder,
8630                    internal_remainder=sp.minimum_remainder,
8631                    total_length=sp.total_length,
8632                    full_remainder=sp.full_remainder}
8633
8634     current_line_indents[#current_line_indents + 1] = new_indent_info
8635     new_index = next_index
8636 end
8637
8638 return true, new_index, current_trail, current_line_indents
8639 end
8640

```

Check if a code trail is expected.

```

8641 local function check_trail(expect_code, is_code)
8642     return (expect_code and is_code) or (not expect_code and not is_code)
8643 end
8644

```

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`.

```

8645 local check_trail_joined =
8646   function(s, i, indent_table, -- luacheck: ignore s i
8647           spacing, expect_code, omit_remainder)
8648     local is_code
8649     local remainder
8650
8651     if has_trail(indent_table) then
8652       local trail = indent_table.trail
8653       is_code = trail.is_code
8654       if is_code then
8655         remainder = trail.remainder
8656       else
8657         remainder = trail.full_remainder
8658       end
8659     else
8660       local sp = process_starter_spacing(0, spacing, 0, 0)
8661       is_code = sp.is_code
8662       if is_code then
8663         remainder = sp.remainder
8664       else
8665         remainder = sp.full_remainder
8666       end
8667     end
8668
8669     local result = check_trail(expect_code, is_code)
8670     if omit_remainder then
8671       return result
8672     end
8673     return result, remainder
8674   end
8675
```

Check if the current trail of the `indent_table` is of length between `min` and `max`.

```

8676 local check_trail_length =
8677   function(s, i, indent_table, -- luacheck: ignore s i
8678           spacing, min, max)
8679     local trail
8680
8681     if has_trail(indent_table) then
8682       trail = indent_table.trail
8683     else
8684       trail = process_starter_spacing(0, spacing, 0, 0)
8685     end
8686
```

```

8687     local total_length = trail.total_length
8688     if total_length == nil then
8689         return false
8690     end
8691
8692     return min <= total_length and total_length <= max
8693 end
8694

```

Check the indentation of the continuation line, optionally with the mode `is_optional` selected. Check blank line exclusively with `is_blank`.

```

8695 local function check_continuation_indentation(s, i, indent_table,
8696                                             is_optional, is_blank)
8697     if not has_indents(indent_table) then
8698         return true
8699     end
8700
8701     local passes, new_index, current_trail, current_line_indents =
8702         traverse_indent(s, i, indent_table, is_optional, is_blank)
8703
8704     if passes then
8705         indent_table.current_line_indents = current_line_indents
8706         indent_table = add_trail(indent_table, current_trail)
8707         return new_index, indent_table
8708     end
8709     return false
8710 end
8711

```

Get name of the last indent from the `indent_table`.

```

8712 local function get_last_indent_name(indent_table)
8713     if has_indents(indent_table) then
8714         return indent_table.indents[#indent_table.indents].name
8715     end
8716 end
8717

```

Remove the remainder altogether if the last indent from the `indent_table` is blank-only.

```

8718 local function remove_remainder_if_blank(indent_table, remainder)
8719     if get_last_indent_name(indent_table) == "li" then
8720         return ""
8721     end
8722     return remainder
8723 end
8724

```



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.

```
8725 local check_trail_type =
8726   function(s, i, -- luacheck: ignore s i
8727             trail, spacing, trail_type)
8728     if trail == nil then
8729         trail = process_starter_spacing(0, spacing, 0, 0)
8730     end
8731
8732     if trail_type == "non-code" then
8733         return check_trail(false, trail.is_code)
8734     end
8735     if trail_type == "code" then
8736         return check_trail(true, trail.is_code)
8737     end
8738     if trail_type == "full-code" then
8739         if (trail.is_code) then
8740             return i, trail.remainder
8741         end
8742         return i, ""
8743     end
8744     if trail_type == "full-any" then
8745         return i, trail.internal_remainder
8746     end
8747 end
8748
```

Stores or restores an `is_freezing` trail from indent table `indent_table`.

```
8749 local trail_freezing =
8750   function(s, i, -- luacheck: ignore s i
8751             indent_table, is_freezing)
8752     if is_freezing then
8753         if indent_table.is_trail_frozen then
8754             indent_table.trail = indent_table.frozen_trail
8755         else
8756             indent_table.frozen_trail = indent_table.trail
8757             indent_table.is_trail_frozen = true
8758         end
8759     else
8760         indent_table.frozen_trail = nil
8761         indent_table.is_trail_frozen = false
8762     end
8763     return true, indent_table
8764 end
8765
```

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`.

```
8766 local check_continuation_indentation_and_trail =
8767   function (s, i, indent_table, is_optional, is_blank, trail_type,
8768             reset_rem, omit_remainder)
8769     if not has_indents(indent_table) then
8770       local spacing, new_index = lpeg.match( C(parsers.spacechar^0)
8771                                             * Cp(), s, i)
8772       local result, remainder = check_trail_type(s, i,
8773         indent_table.trail, spacing, trail_type)
8774       if remainder == nil then
8775         if result then
8776           return new_index
8777         end
8778         return false
8779       end
8780       if result then
8781         return new_index, remainder
8782       end
8783       return false
8784     end
8785
8786     local passes, new_index, current_trail = traverse_indent(s, i,
8787       indent_table, is_optional, is_blank)
8788
8789     if passes then
8790       local spacing
8791       if current_trail == nil then
8792         local newer_spacing, newer_index = lpeg.match(
8793           C(parsers.spacechar^0) * Cp(), s, i)
8794         current_trail = process_starter_spacing(0, newer_spacing, 0, 0)
8795         new_index = newer_index
8796         spacing = newer_spacing
8797       else
8798         spacing = current_trail.remainder
8799       end
8800       local result, remainder = check_trail_type(s, new_index,
8801         current_trail, spacing, trail_type)
8802       if remainder == nil or omit_remainder then
8803         if result then
8804           return new_index
8805         end
8806         return false
8807       end
8808
8809       if is_blank and reset_rem then
8810         remainder = remove_remainder_if_blank(indent_table, remainder)
```

```

8811     end
8812     if result then
8813         return new_index, remainder
8814     end
8815     return false
8816 end
8817 return false
8818 end
8819

```

The following patterns check whitespace indentation at the start of a block.

```

8820 parsers.check_trail = Cmt( Cb("indent_info") * C(parsers.spacechar^0)
8821                        * Cc(false), check_trail_joined)
8822
8823 parsers.check_trail_no_rem = Cmt( Cb("indent_info")
8824                                * C(parsers.spacechar^0) * Cc(false)
8825                                * Cc(true), check_trail_joined)
8826
8827 parsers.check_code_trail  = Cmt( Cb("indent_info")
8828                                * C(parsers.spacechar^0)
8829                                * Cc(true), check_trail_joined)
8830
8831 parsers.check_trail_length_range = function(min, max)
8832     return Cmt( Cb("indent_info") * C(parsers.spacechar^0) * Cc(min)
8833               * Cc(max), check_trail_length)
8834 end
8835
8836 parsers.check_trail_length = function(n)
8837     return parsers.check_trail_length_range(n, n)
8838 end
8839

```

The following patterns handle trail backup, to prevent a failing pattern to modify it before passing it to the next.

```

8840 parsers.freeze_trail = Cg( Cmt(Cb("indent_info")
8841                        * Cc(true), trail_freezing), "indent_info")
8842
8843 parsers.unfreeze_trail = Cg(Cmt(Cb("indent_info") * Cc(false),
8844                        trail_freezing), "indent_info")
8845

```

The following patterns check indentation in continuation lines as defined by the container start.

```

8846 parsers.check_minimal_indent = Cmt(Cb("indent_info") * Cc(false),
8847                                check_continuation_indentation)
8848
8849 parsers.check_optional_indent = Cmt(Cb("indent_info") * Cc(true),
8850                                check_continuation_indentation)

```

```

8851
8852 parsers.check_minimal_blank_indent
8853     = Cmt( Cb("indent_info") * Cc(false)
8854           * Cc(true)
8855           , check_continuation_indentation)
8856

```

The following patterns check indentation in continuation lines as defined by the container start. Additionally the subsequent trail is also directly checked.

```

8857
8858 parsers.check_minimal_indent_and_trail =
8859     Cmt( Cb("indent_info")
8860         * Cc(false) * Cc(false) * Cc("non-code") * Cc(true)
8861         , check_continuation_indentation_and_trail)
8862
8863 parsers.check_minimal_indent_and_code_trail =
8864     Cmt( Cb("indent_info")
8865         * Cc(false) * Cc(false) * Cc("code") * Cc(false)
8866         , check_continuation_indentation_and_trail)
8867
8868 parsers.check_minimal_blank_indent_and_full_code_trail =
8869     Cmt( Cb("indent_info")
8870         * Cc(false) * Cc(true) * Cc("full-code") * Cc(true)
8871         , check_continuation_indentation_and_trail)
8872
8873 parsers.check_minimal_indent_and_any_trail =
8874     Cmt( Cb("indent_info")
8875         * Cc(false) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
8876         , check_continuation_indentation_and_trail)
8877
8878 parsers.check_minimal_blank_indent_and_any_trail =
8879     Cmt( Cb("indent_info")
8880         * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
8881         , check_continuation_indentation_and_trail)
8882
8883 parsers.check_minimal_blank_indent_and_any_trail_no_rem =
8884     Cmt( Cb("indent_info")
8885         * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(true)
8886         , check_continuation_indentation_and_trail)
8887
8888 parsers.check_optional_indent_and_any_trail =
8889     Cmt( Cb("indent_info")
8890         * Cc(true) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
8891         , check_continuation_indentation_and_trail)
8892
8893 parsers.check_optional_blank_indent_and_any_trail =
8894     Cmt( Cb("indent_info")

```

```

8895      * Cc(true) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
8896      , check_continuation_indentation_and_trail)
8897

```

The following patterns specify behaviour around newlines.

```

8898
8899 parsers.spnlc_noexc = parsers.optionalspace
8900                      * ( parsers.newline
8901                        * parsers.check_minimal_indent_and_any_trail)^-1
8902
8903 parsers.spnlc = parsers.optionalspace
8904               * (V("EndlineNoSub"))^-1
8905
8906 parsers.spnlc_sep = parsers.optionalspace * V("EndlineNoSub")
8907                  + parsers.spacechar^1
8908
8909 parsers.only_blank = parsers.spacechar^0
8910                   * (parsers.newline + parsers.eof)
8911

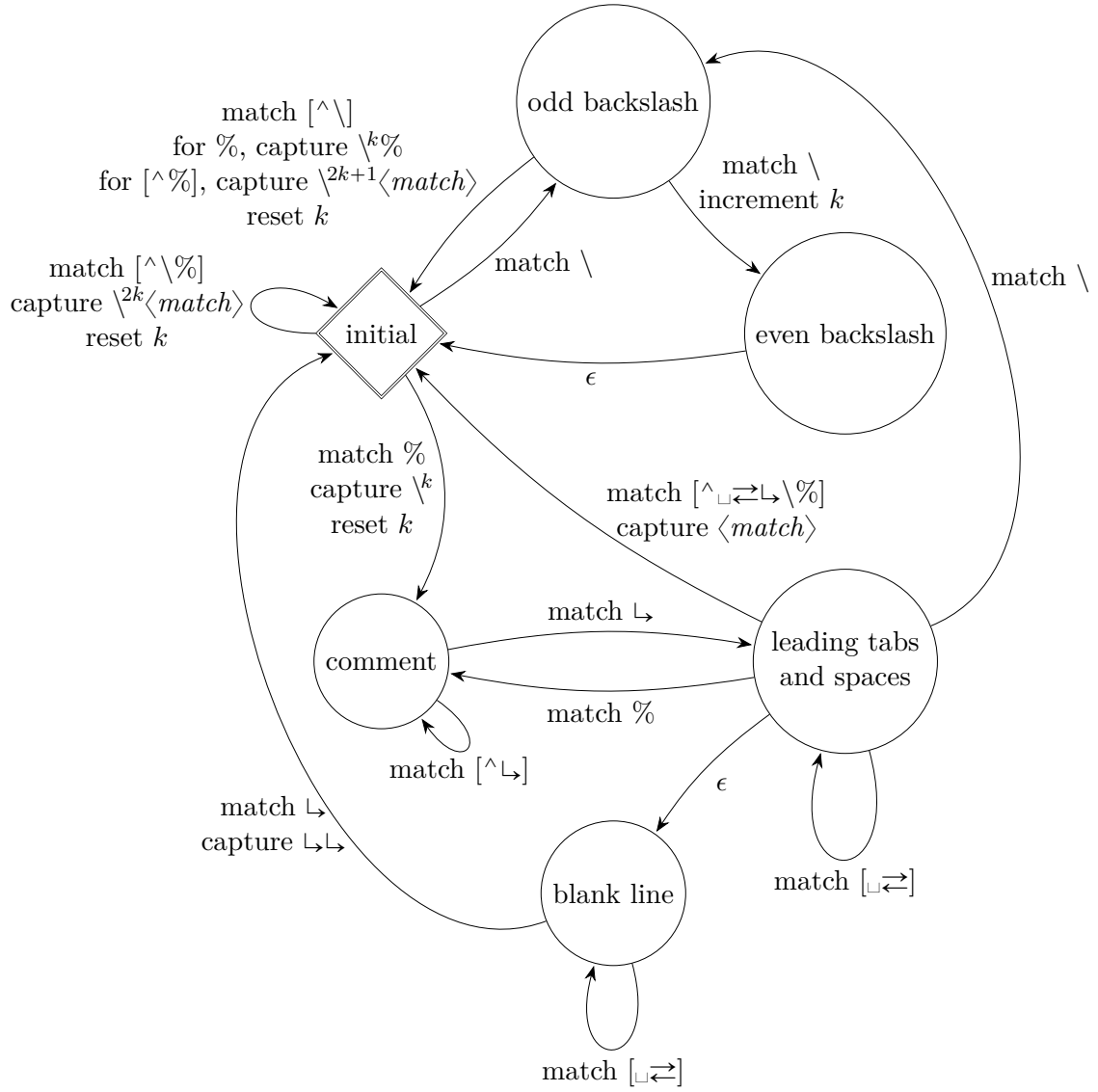
```

The `parserscommented\_line^1` parser recognizes the regular language of  $\text{T}_{\text{E}}\text{X}$  comments, see an equivalent finite automaton in Figure 8.

```

8912 parserscommented_line_letter = parsers.linechar
8913                               + parsers.newline
8914                               - parsers.backslash
8915                               - parsers.percent
8916 parserscommented_line = Cg(Cc(""), "backslashes")
8917                       * ((#(parserscommented_line_letter
8918                          - parsers.newline)
8919                          * Cb("backslashes")
8920                          * Cs(parserscommented_line_letter
8921                             - parsers.newline)^1 -- initial
8922                          * Cg(Cc(""), "backslashes"))
8923                       + #( parsers.backslash
8924                          * (parsers.backslash + parsers.newline))
8925                       * Cg((parsers.backslash -- even backslash
8926                          * ( parsers.backslash
8927                            + #parsers.newline))^1, "backslashes")
8928                       + (parsers.backslash
8929                          * (#parsers.percent
8930                            * Cb("backslashes")
8931                            / function(backslashes)
8932                              return string.rep("\\", #backslashes / 2)
8933                            end
8934                          * C(parsers.percent)
8935                          + #parserscommented_line_letter
8936                          * Cb("backslashes")
8937                          * Cc("\\"))

```



**Figure 8: A pushdown automaton that recognizes TeX comments**

```

8938         * C(parsers.commented_line_letter))
8939         * Cg(Cc(""), "backslashes"))^0
8940     * (#parsers.percent
8941     * Cb("backslashes")
8942     / function(backslashes)
8943         return string.rep("\\", #backslashes / 2)
8944     end
8945     * ((parsers.percent -- comment
8946     * parsers.line
8947     * #parsers.blankline) -- blank line
8948     / "\n"
8949     + parsers.percent -- comment
8950     * parsers.line
8951     * parsers.optionalspace) -- leading spaces
8952     + #(parsers.newline)
8953     * Cb("backslashes")
8954     * C(parsers.newline))
8955
8956 parsers.chunk = parsers.line * (parsers.optionallyindentedline
8957                               - parsers.blankline)^0
8958
8959 parsers.attribute_key_char = parsers.alphanumeric + S("-_:.")
8960 parsers.attribute_raw_char = parsers.alphanumeric + S("-_")
8961 parsers.attribute_key = (parsers.attribute_key_char
8962                        - parsers.dash - parsers.digit)
8963                        * parsers.attribute_key_char^0
8964 parsers.attribute_value = ( (parsers.dquote / "\"")
8965                          * (parsers.anyescaped - parsers.dquote)^0
8966                          * (parsers.dquote / "\""))
8967                      + ( (parsers.squote / "\"")
8968                        * (parsers.anyescaped - parsers.squote)^0
8969                        * (parsers.squote / "\""))
8970                      + ( parsers.anyescaped
8971                        - parsers.dquote
8972                        - parsers.rbrace
8973                        - parsers.space)^0
8974 parsers.attribute_identifier = parsers.attribute_key_char^1
8975 parsers.attribute_classname = parsers.letter
8976                             * parsers.attribute_key_char^0
8977 parsers.attribute_raw = parsers.attribute_raw_char^1
8978
8979 parsers.attribute = (parsers.dash * Cc(".unnumbered"))
8980                  + C( parsers.hash
8981                    * parsers.attribute_identifier)
8982                  + C( parsers.period
8983                    * parsers.attribute_classname)
8984                  + Cs( parsers.attribute_key

```

```

8985             * parsers.optionalspace
8986             * parsers.equal
8987             * parsers.optionalspace
8988             * parsers.attribute_value)
8989 parsers.attributes = parsers.lbrace
8990             * parsers.optionalspace
8991             * parsers.attribute
8992             * (parsers.spacechar^1
8993             * parsers.attribute)^0
8994             * parsers.optionalspace
8995             * parsers.rbrace
8996
8997 parsers.raw_attribute = parsers.lbrace
8998             * parsers.optionalspace
8999             * parsers.equal
9000             * C(parsers.attribute_raw)
9001             * parsers.optionalspace
9002             * parsers.rbrace
9003
9004 -- block followed by 0 or more optionally
9005 -- indented blocks with first line indented.
9006 parsers.indented_blocks = function(bl)
9007   return Cs( bl
9008             * ( parsers.blankline^1
9009             * parsers.indent
9010             * -parsers.blankline
9011             * bl)^0
9012             * (parsers.blankline^1 + parsers.eof) )
9013 end

```

### 3.1.5.4 Parsers Used for HTML Entities

```

9014 local function repeat_between(pattern, min, max)
9015   return -pattern^(max + 1) * pattern^min
9016 end
9017
9018 parsers.hexentity = parsers.ampersand * parsers.hash * C(S("Xx"))
9019             * C(repeat_between(parsers.hexdigit, 1, 6))
9020             * parsers.semicolon
9021 parsers.decentity = parsers.ampersand * parsers.hash
9022             * C(repeat_between(parsers.digit, 1, 7))
9023             * parsers.semicolon
9024 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
9025             * parsers.semicolon
9026
9027 parsers.html_entities
9028   = parsers.hexentity / entities.hex_entity_with_x_char

```



```

9029 + parsers.decentity / entities.dec_entity
9030 + parsers.tagentity / entities.char_entity

```

### 3.1.5.5 Parsers Used for Markdown Lists

```

9031 parsers.bullet = function(bullet_char, interrupting)
9032   local allowed_end
9033   if interrupting then
9034     allowed_end = C(parsers.spacechar^1) * #parsers.linechar
9035   else
9036     allowed_end = C(parsers.spacechar^1)
9037                   + #(parsers.newline + parsers.eof)
9038   end
9039   return parsers.check_trail
9040         * Ct(C(bullet_char) * Cc(""))
9041         * allowed_end
9042 end
9043
9044 local function tickbox(interior)
9045   return parsers.optionalspace * parsers.lbracket
9046         * interior * parsers.rbracket * parsers.spacechar^1
9047 end
9048
9049 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
9050 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
9051 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
9052

```

### 3.1.5.6 Parsers Used for Markdown Code Spans

```

9053 parsers.openticks = Cg(parsers.backtick^1, "ticks")
9054
9055 local function captures_equal_length(_,i,a,b)
9056   return #a == #b and i
9057 end
9058
9059 parsers.closeticks = Cmt(C(parsers.backtick^1)
9060                          * Cb("ticks"), captures_equal_length)
9061
9062 parsers.intickschar = (parsers.any - S("\n\r`"))
9063                      + V("NoSoftLineBreakEndline")
9064                      + (parsers.backtick^1 - parsers.closeticks)
9065
9066 local function process_inticks(s)
9067   s = s:gsub("\n", " ")
9068   s = s:gsub("^ (.*) $", "%1")
9069   return s
9070 end

```

```

9071
9072 parsers.inticks = parsers.openticks
9073         * C(parsers.space^0)
9074         * parsers.closeticks
9075         + parsers.openticks
9076         * Cs(Cs(parsers.intickschar^0) / process_inticks)
9077         * parsers.closeticks
9078

```

### 3.1.5.7 Parsers Used for HTML

```

9079 -- case-insensitive match (we assume s is lowercase)
9080 -- must be single byte encoding
9081 parsers.keyword_exact = function(s)
9082     local parser = P(0)
9083     for i=1,#s do
9084         local c = s:sub(i,i)
9085         local m = c .. upper(c)
9086         parser = parser * S(m)
9087     end
9088     return parser
9089 end
9090
9091 parsers.special_block_keyword =
9092     parsers.keyword_exact("pre") +
9093     parsers.keyword_exact("script") +
9094     parsers.keyword_exact("style") +
9095     parsers.keyword_exact("textarea")
9096
9097 parsers.block_keyword =
9098     parsers.keyword_exact("address") +
9099     parsers.keyword_exact("article") +
9100     parsers.keyword_exact("aside") +
9101     parsers.keyword_exact("base") +
9102     parsers.keyword_exact("basefont") +
9103     parsers.keyword_exact("blockquote") +
9104     parsers.keyword_exact("body") +
9105     parsers.keyword_exact("caption") +
9106     parsers.keyword_exact("center") +
9107     parsers.keyword_exact("col") +
9108     parsers.keyword_exact("colgroup") +
9109     parsers.keyword_exact("dd") +
9110     parsers.keyword_exact("details") +
9111     parsers.keyword_exact("dialog") +
9112     parsers.keyword_exact("dir") +
9113     parsers.keyword_exact("div") +
9114     parsers.keyword_exact("dl") +

```

```

9115     parsers.keyword_exact("dt") +
9116     parsers.keyword_exact("fieldset") +
9117     parsers.keyword_exact("figcaption") +
9118     parsers.keyword_exact("figure") +
9119     parsers.keyword_exact("footer") +
9120     parsers.keyword_exact("form") +
9121     parsers.keyword_exact("frame") +
9122     parsers.keyword_exact("frameset") +
9123     parsers.keyword_exact("h1") +
9124     parsers.keyword_exact("h2") +
9125     parsers.keyword_exact("h3") +
9126     parsers.keyword_exact("h4") +
9127     parsers.keyword_exact("h5") +
9128     parsers.keyword_exact("h6") +
9129     parsers.keyword_exact("head") +
9130     parsers.keyword_exact("header") +
9131     parsers.keyword_exact("hr") +
9132     parsers.keyword_exact("html") +
9133     parsers.keyword_exact("iframe") +
9134     parsers.keyword_exact("legend") +
9135     parsers.keyword_exact("li") +
9136     parsers.keyword_exact("link") +
9137     parsers.keyword_exact("main") +
9138     parsers.keyword_exact("menu") +
9139     parsers.keyword_exact("menuitem") +
9140     parsers.keyword_exact("nav") +
9141     parsers.keyword_exact("noframes") +
9142     parsers.keyword_exact("ol") +
9143     parsers.keyword_exact("optgroup") +
9144     parsers.keyword_exact("option") +
9145     parsers.keyword_exact("p") +
9146     parsers.keyword_exact("param") +
9147     parsers.keyword_exact("section") +
9148     parsers.keyword_exact("source") +
9149     parsers.keyword_exact("summary") +
9150     parsers.keyword_exact("table") +
9151     parsers.keyword_exact("tbody") +
9152     parsers.keyword_exact("td") +
9153     parsers.keyword_exact("tfoot") +
9154     parsers.keyword_exact("th") +
9155     parsers.keyword_exact("thead") +
9156     parsers.keyword_exact("title") +
9157     parsers.keyword_exact("tr") +
9158     parsers.keyword_exact("track") +
9159     parsers.keyword_exact("ul")
9160
9161 -- end conditions

```

```

9162 parsers.html_blankline_end_condition
9163   = parsers.linechar^0
9164   * ( parsers.newline
9165       * (parsers.check_minimal_blank_indent_and_any_trail
9166           * #parsers.blankline
9167           + parsers.check_minimal_indent_and_any_trail)
9168       * parsers.linechar^1)^0
9169   * (parsers.newline^-1 / "")
9170
9171 local function remove_trailing_blank_lines(s)
9172   return s:gsub("[\n\r]+%s*$", "")
9173 end
9174
9175 parsers.html_until_end = function(end_marker)
9176   return Cs(Cs((parsers.newline
9177                 * (parsers.check_minimal_blank_indent_and_any_trail
9178                     * #parsers.blankline
9179                     + parsers.check_minimal_indent_and_any_trail)
9180                 + parsers.linechar - end_marker)^0
9181               * parsers.linechar^0 * parsers.newline^-1)
9182             / remove_trailing_blank_lines)
9183 end
9184
9185 -- attributes
9186 parsers.html_attribute_spacing = parsers.optionalspace
9187                                * V("NoSoftLineBreakEndline")
9188                                * parsers.optionalspace
9189                                + parsers.spacechar^1
9190
9191 parsers.html_attribute_name = ( parsers.letter
9192                                + parsers.colon
9193                                + parsers.underscore)
9194                                * ( parsers.alphanumeric
9195                                + parsers.colon
9196                                + parsers.underscore
9197                                + parsers.period
9198                                + parsers.dash)^0
9199
9200 parsers.html_attribute_value = parsers.squote
9201                                * (parsers.linechar - parsers.squote)^0
9202                                * parsers.squote
9203                                + parsers.dquote
9204                                * (parsers.linechar - parsers.dquote)^0
9205                                * parsers.dquote
9206                                + ( parsers.any
9207                                    - parsers.spacechar
9208                                    - parsers.newline

```

```

9209         - parsers.dquote
9210         - parsers.squote
9211         - parsers.backtick
9212         - parsers.equal
9213         - parsers.less
9214         - parsers.more)^1
9215
9216 parsers.html_inline_attribute_value = parsers.squote
9217         * (V("NoSoftLineBreakEndline")
9218         + parsers.any
9219         - parsers.blankline^2
9220         - parsers.squote)^0
9221         * parsers.squote
9222         + parsers.dquote
9223         * (V("NoSoftLineBreakEndline")
9224         + parsers.any
9225         - parsers.blankline^2
9226         - parsers.dquote)^0
9227         * parsers.dquote
9228         + (parsers.any
9229         - parsers.spacechar
9230         - parsers.newline
9231         - parsers.dquote
9232         - parsers.squote
9233         - parsers.backtick
9234         - parsers.equal
9235         - parsers.less
9236         - parsers.more)^1
9237
9238 parsers.html_attribute_value_specification
9239     = parsers.optionalspace
9240     * parsers.equal
9241     * parsers.optionalspace
9242     * parsers.html_attribute_value
9243
9244 parsers.html_spnl = parsers.optionalspace
9245         * (V("NoSoftLineBreakEndline")
9246         * parsers.optionalspace)^-1
9247
9248 parsers.html_inline_attribute_value_specification
9249     = parsers.html_spnl
9250     * parsers.equal
9251     * parsers.html_spnl
9252     * parsers.html_inline_attribute_value
9253
9254 parsers.html_attribute
9255     = parsers.html_attribute_spacing

```

```

9256 * parsers.html_attribute_name
9257 * parsers.html_inline_attribute_value_specification^-1
9258
9259 parsers.html_non_newline_attribute
9260 = parsers.spacechar^1
9261 * parsers.html_attribute_name
9262 * parsers.html_attribute_value_specification^-1
9263
9264 parsers.nested_breaking_blank = parsers.newline
9265                               * parsers.check_minimal_blank_indent
9266                               * parsers.blankline
9267
9268 parsers.html_comment_start = P("<!--")
9269
9270 parsers.html_comment_end = P("-->")
9271
9272 parsers.html_comment
9273 = Cs( parsers.html_comment_start
9274       * parsers.html_until_end(parsers.html_comment_end))
9275
9276 parsers.html_inline_comment = (parsers.html_comment_start / "")
9277                             * -P(">") * -P("-->")
9278                             * Cs(( V("NoSoftLineBreakEndline")
9279                                   + parsers.any
9280                                   - parsers.nested_breaking_blank
9281                                   - parsers.html_comment_end)^0)
9282                             * (parsers.html_comment_end / "")
9283
9284 parsers.html_cdatasection_start = P("<![CDATA[")
9285
9286 parsers.html_cdatasection_end = P("]]>")
9287
9288 parsers.html_cdatasection
9289 = Cs( parsers.html_cdatasection_start
9290       * parsers.html_until_end(parsers.html_cdatasection_end))
9291
9292 parsers.html_inline_cdatasection
9293 = parsers.html_cdatasection_start
9294 * Cs(V("NoSoftLineBreakEndline") + parsers.any
9295     - parsers.nested_breaking_blank - parsers.html_cdatasection_end)^0
9296 * parsers.html_cdatasection_end
9297
9298 parsers.html_declaration_start = P("<!") * parsers.letter
9299
9300 parsers.html_declaration_end = P(">")
9301
9302 parsers.html_declaration

```

```

9303     = Cs( parsers.html_declaration_start
9304         * parsers.html_until_end(parsers.html_declaration_end))
9305
9306 parsers.html_inline_declaration
9307     = parsers.html_declaration_start
9308     * Cs(V("NoSoftLineBreakEndline") + parsers.any
9309         - parsers.nested_breaking_blank - parsers.html_declaration_end)^0
9310     * parsers.html_declaration_end
9311
9312 parsers.html_instruction_start = P("<?")
9313
9314 parsers.html_instruction_end = P("?>")
9315
9316 parsers.html_instruction
9317     = Cs( parsers.html_instruction_start
9318         * parsers.html_until_end(parsers.html_instruction_end))
9319
9320 parsers.html_inline_instruction = parsers.html_instruction_start
9321                                 * Cs( V("NoSoftLineBreakEndline")
9322                                     + parsers.any
9323                                     - parsers.nested_breaking_blank
9324                                     - parsers.html_instruction_end)^0
9325                                 * parsers.html_instruction_end
9326
9327 parsers.html_blankline = parsers.newline
9328                       * parsers.optionalspace
9329                       * parsers.newline
9330
9331 parsers.html_tag_start = parsers.less
9332
9333 parsers.html_tag_closing_start = parsers.less
9334                               * parsers.slash
9335
9336 parsers.html_tag_end = parsers.html_spnl
9337                     * parsers.more
9338
9339 parsers.html_empty_tag_end = parsers.html_spnl
9340                           * parsers.slash
9341                           * parsers.more
9342
9343 -- opening tags
9344 parsers.html_any_open_inline_tag = parsers.html_tag_start
9345                                 * parsers.keyword
9346                                 * parsers.html_attribute^0
9347                                 * parsers.html_tag_end
9348
9349 parsers.html_any_open_tag = parsers.html_tag_start

```

```

9350             * parsers.keyword
9351             * parsers.html_non_newline_attribute^0
9352             * parsers.html_tag_end
9353
9354 parsers.html_open_tag = parsers.html_tag_start
9355             * parsers.block_keyword
9356             * parsers.html_attribute^0
9357             * parsers.html_tag_end
9358
9359 parsers.html_open_special_tag = parsers.html_tag_start
9360             * parsers.special_block_keyword
9361             * parsers.html_attribute^0
9362             * parsers.html_tag_end
9363
9364 -- incomplete tags
9365 parsers.incomplete_tag_following = parsers.spacechar
9366             + parsers.more
9367             + parsers.slash * parsers.more
9368             + #(parsers.newline + parsers.eof)
9369
9370 parsers.incomplete_special_tag_following = parsers.spacechar
9371             + parsers.more
9372             + #( parsers.newline
9373                 + parsers.eof)
9374
9375 parsers.html_incomplete_open_tag = parsers.html_tag_start
9376             * parsers.block_keyword
9377             * parsers.incomplete_tag_following
9378
9379 parsers.html_incomplete_open_special_tag
9380 = parsers.html_tag_start
9381     * parsers.special_block_keyword
9382     * parsers.incomplete_special_tag_following
9383
9384 parsers.html_incomplete_close_tag = parsers.html_tag_closing_start
9385             * parsers.block_keyword
9386             * parsers.incomplete_tag_following
9387
9388 parsers.html_incomplete_close_special_tag
9389 = parsers.html_tag_closing_start
9390     * parsers.special_block_keyword
9391     * parsers.incomplete_tag_following
9392
9393 -- closing tags
9394 parsers.html_close_tag = parsers.html_tag_closing_start
9395             * parsers.block_keyword
9396             * parsers.html_tag_end

```



```

9397
9398 parsers.html_any_close_tag = parsers.html_tag_closing_start
9399                             * parsers.keyword
9400                             * parsers.html_tag_end
9401
9402 parsers.html_close_special_tag = parsers.html_tag_closing_start
9403                             * parsers.special_block_keyword
9404                             * parsers.html_tag_end
9405
9406 -- empty tags
9407 parsers.html_any_empty_inline_tag = parsers.html_tag_start
9408                                 * parsers.keyword
9409                                 * parsers.html_attribute^0
9410                                 * parsers.html_empty_tag_end
9411
9412 parsers.html_any_empty_tag = parsers.html_tag_start
9413                             * parsers.keyword
9414                             * parsers.html_non_newline_attribute^0
9415                             * parsers.optionalspace
9416                             * parsers.slash
9417                             * parsers.more
9418
9419 parsers.html_empty_tag = parsers.html_tag_start
9420                       * parsers.block_keyword
9421                       * parsers.html_attribute^0
9422                       * parsers.html_empty_tag_end
9423
9424 parsers.html_empty_special_tag = parsers.html_tag_start
9425                               * parsers.special_block_keyword
9426                               * parsers.html_attribute^0
9427                               * parsers.html_empty_tag_end
9428
9429 parsers.html_incomplete_blocks
9430   = parsers.html_incomplete_open_tag
9431   + parsers.html_incomplete_open_special_tag
9432   + parsers.html_incomplete_close_tag
9433
9434 -- parse special html blocks
9435 parsers.html_blankline_ending_special_block_opening
9436   = ( parsers.html_close_special_tag
9437       + parsers.html_empty_special_tag)
9438   * #( parsers.optionalspace
9439       * (parsers.newline + parsers.eof))
9440
9441 parsers.html_blankline_ending_special_block
9442   = parsers.html_blankline_ending_special_block_opening
9443   * parsers.html_blankline_end_condition

```

```

9444
9445 parsers.html_special_block_opening
9446     = parsers.html_incomplete_open_special_tag
9447     - parsers.html_empty_special_tag
9448
9449 parsers.html_closing_special_block
9450     = parsers.html_special_block_opening
9451     * parsers.html_until_end(parsers.html_close_special_tag)
9452
9453 parsers.html_special_block
9454     = parsers.html_blankline_ending_special_block
9455     + parsers.html_closing_special_block
9456
9457 -- parse html blocks
9458 parsers.html_block_opening = parsers.html_incomplete_open_tag
9459                             + parsers.html_incomplete_close_tag
9460
9461 parsers.html_block = parsers.html_block_opening
9462                   * parsers.html_blankline_end_condition
9463
9464 -- parse any html blocks
9465 parsers.html_any_block_opening
9466     = ( parsers.html_any_open_tag
9467         + parsers.html_any_close_tag
9468         + parsers.html_any_empty_tag)
9469     * #(parsers.optionalspace * (parsers.newline + parsers.eof))
9470
9471 parsers.html_any_block = parsers.html_any_block_opening
9472                       * parsers.html_blankline_end_condition
9473
9474 parsers.html_inline_comment_full = parsers.html_comment_start
9475                                 * -P(">") * -P("->")
9476                                 * Cs(( V("NoSoftLineBreakEndline")
9477                                     + parsers.any - P("--")
9478                                     - parsers.nested_breaking_blank
9479                                     - parsers.html_comment_end)^0)
9480                                 * parsers.html_comment_end
9481
9482 parsers.html_inline_tags = parsers.html_inline_comment_full
9483                          + parsers.html_any_empty_inline_tag
9484                          + parsers.html_inline_instruction
9485                          + parsers.html_inline_cdatasection
9486                          + parsers.html_inline_declaration
9487                          + parsers.html_any_open_inline_tag
9488                          + parsers.html_any_close_tag
9489

```

### 3.1.5.8 Parsers Used for Markdown Tags and Links

```
9490 parsers.urlchar = parsers.anyescaped
9491             - parsers.newline
9492             - parsers.more
9493
9494 parsers.auto_link_scheme_part = parsers.alphanumeric
9495             + parsers.plus
9496             + parsers.period
9497             + parsers.dash
9498
9499 parsers.auto_link_scheme = parsers.letter
9500             * parsers.auto_link_scheme_part
9501             * parsers.auto_link_scheme_part^-30
9502
9503 parsers.absolute_uri = parsers.auto_link_scheme * parsers.colon
9504             * ( parsers.any - parsers.spacing
9505             - parsers.less - parsers.more)^0
9506
9507 parsers.printable_characters = S("!.#$%&'*/+/?^_`{|}~-")
9508
9509 parsers.email_address_local_part_char = parsers.alphanumeric
9510             + parsers.printable_characters
9511
9512 parsers.email_address_local_part
9513     = parsers.email_address_local_part_char^1
9514
9515 parsers.email_address_dns_label = parsers.alphanumeric
9516             * ( parsers.alphanumeric
9517             + parsers.dash)^-62
9518             * B(parsers.alphanumeric)
9519
9520 parsers.email_address_domain = parsers.email_address_dns_label
9521             * ( parsers.period
9522             * parsers.email_address_dns_label)^0
9523
9524 parsers.email_address = parsers.email_address_local_part
9525             * parsers.at
9526             * parsers.email_address_domain
9527
9528 parsers.auto_link_url = parsers.less
9529             * C(parsers.absolute_uri)
9530             * parsers.more
9531
9532 parsers.auto_link_email = parsers.less
9533             * C(parsers.email_address)
9534             * parsers.more
9535
```

```

9536 parsers.auto_link_relative_reference = parsers.less
9537                                     * C(parsers.urlchar^1)
9538                                     * parsers.more
9539
9540 parsers.autolink = parsers.auto_link_url
9541                 + parsers.auto_link_email
9542
9543 -- content in balanced brackets, parentheses, or quotes:
9544 parsers.bracketed = P{ parsers.lbracket
9545                       * (( parsers.backslash / "\"" * parsers.rbracket
9546                           + parsers.any - (parsers.lbracket
9547                                           + parsers.rbracket
9548                                           + parsers.blankline^2)
9549                           ) + V(1))^0
9550                       * parsers.rbracket }
9551
9552 parsers.inparens = P{ parsers.lparent
9553                       * ((parsers.anyescaped - (parsers.lparent
9554                                                  + parsers.rparent
9555                                                  + parsers.blankline^2)
9556                       ) + V(1))^0
9557                       * parsers.rparent }
9558
9559 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
9560                      * ((parsers.anyescaped - (parsers.squote
9561                                                  + parsers.blankline^2)
9562                      ) + V(1))^0
9563                      * parsers.squote }
9564
9565 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
9566                     * ((parsers.anyescaped - (parsers.dquote
9567                                                  + parsers.blankline^2)
9568                     ) + V(1))^0
9569                     * parsers.dquote }
9570
9571 parsers.link_text = parsers.lbracket
9572                  * Cs((parsers.alphanumeric^1
9573                      + parsers.bracketed
9574                      + parsers.inticks
9575                      + parsers.autolink
9576                      + V("InlineHtml")
9577                      + ( parsers.backslash * parsers.backslash)
9578                      + ( parsers.backslash
9579                          * ( parsers.lbracket
9580                              + parsers.rbracket)
9581                      + V("NoSoftLineBreakSpace")
9582                      + V("NoSoftLineBreakEndline")

```

```

9583         + (parsers.any
9584         - ( parsers.newline
9585         + parsers.lbracket
9586         + parsers.rbracket
9587         + parsers.blankline^2))))^0)
9588     * parsers.rbracket
9589
9590 parsers.link_label_body = -(parsers.sp * parsers.rbracket)
9591     * #( ( parsers.any
9592     - parsers.rbracket)^-999
9593     * parsers.rbracket)
9594     * Cs((parsers.alphanumeric^1
9595     + parsers.inticks
9596     + parsers.autolink
9597     + V("InlineHtml")
9598     + ( parsers.backslash * parsers.backslash)
9599     + ( parsers.backslash
9600     * ( parsers.lbracket
9601     + parsers.rbracket)
9602     + V("NoSoftLineBreakSpace")
9603     + V("NoSoftLineBreakEndline")
9604     + (parsers.any
9605     - ( parsers.newline
9606     + parsers.lbracket
9607     + parsers.rbracket
9608     + parsers.blankline^2))))^1)
9609
9610 parsers.link_label = parsers.lbracket
9611     * parsers.link_label_body
9612     * parsers.rbracket
9613
9614 parsers.inparens_url = P{ parsers.lparent
9615     * ((parsers.anyescaped - (parsers.lparent
9616     + parsers.rparent
9617     + parsers.spacing)
9618     ) + V(1))^0
9619     * parsers.rparent }
9620
9621 -- url for markdown links, allowing nested brackets:
9622 parsers.url = parsers.less * Cs((parsers.anyescaped
9623     - parsers.newline
9624     - parsers.less
9625     - parsers.more)^0)
9626     * parsers.more
9627     + -parsers.less
9628     * Cs((parsers.inparens_url + (parsers.anyescaped
9629     - parsers.spacing

```

```

9630                                     - parsers.lparent
9631                                     - parsers.rparent))^1)
9632
9633 -- quoted text:
9634 parsers.title_s      = parsers.squote
9635                       * Cs((parsers.html_entities
9636                           + V("NoSoftLineBreakSpace")
9637                           + V("NoSoftLineBreakEndline")
9638                           + ( parsers.anyescaped
9639                             - parsers.newline
9640                             - parsers.squote
9641                             - parsers.blankline^2))^0)
9642                       * parsers.squote
9643
9644 parsers.title_d      = parsers.dquote
9645                       * Cs((parsers.html_entities
9646                           + V("NoSoftLineBreakSpace")
9647                           + V("NoSoftLineBreakEndline")
9648                           + ( parsers.anyescaped
9649                             - parsers.newline
9650                             - parsers.dquote
9651                             - parsers.blankline^2))^0)
9652                       * parsers.dquote
9653
9654 parsers.title_p      = parsers.lparent
9655                       * Cs((parsers.html_entities
9656                           + V("NoSoftLineBreakSpace")
9657                           + V("NoSoftLineBreakEndline")
9658                           + ( parsers.anyescaped
9659                             - parsers.newline
9660                             - parsers.lparent
9661                             - parsers.rparent
9662                             - parsers.blankline^2))^0)
9663                       * parsers.rparent
9664
9665 parsers.title
9666     = parsers.title_d + parsers.title_s + parsers.title_p
9667
9668 parsers.optionaltitle
9669     = parsers.spnlc * parsers.title * parsers.spacechar^0 + Cc("")
9670

```

### 3.1.5.9 Helpers for Links and Link Reference Definitions

```

9671 -- parse a reference definition: [foo]: /bar "title"
9672 parsers.define_reference_parser = (parsers.check_trail / "")
9673                                * parsers.link_label * parsers.colon

```

```

9674                                     * parsers.spnlc * parsers.url
9675                                     * ( parsers.spnlc_sep * parsers.title
9676                                     * parsers.only_blank
9677                                     + Cc("") * parsers.only_blank)

```

### 3.1.5.10 Inline Elements

```

9678 parsers.Inline          = V("Inline")
9679
9680 -- parse many p between starter and ender
9681 parsers.between = function(p, starter, ender)
9682   local ender2 = B(parsers.nonspacechar) * ender
9683   return ( starter
9684           * #parsers.nonspacechar
9685           * Ct(p * (p - ender2)^0)
9686           * ender2)
9687 end
9688

```

### 3.1.5.11 Block Elements

```

9689 parsers.lineof = function(c)
9690   return ( parsers.check_trail_no_rem
9691           * (P(c) * parsers.optionalspace)^3
9692           * (parsers.newline + parsers.eof))
9693 end
9694
9695 parsers.thematic_break_lines = parsers.lineof(parsers.asterisk)
9696                               + parsers.lineof(parsers.dash)
9697                               + parsers.lineof(parsers.underscore)

```

### 3.1.5.12 Headings

```

9698 -- parse Atx heading start and return level
9699 parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
9700                       * -parsers.hash / length
9701
9702 -- parse setext header ending and return level
9703 parsers.heading_level
9704   = parsers.nonindentSPACE * parsers.equal^1
9705   * parsers.optionalspace * #parsers.newline * Cc(1)
9706   + parsers.nonindentSPACE * parsers.dash^1
9707   * parsers.optionalspace * #parsers.newline * Cc(2)
9708
9709 local function strip_atx_end(s)
9710   return s:gsub("%s+#+%s*\n$", "")
9711 end
9712

```

```

9713 parsers.atx_heading = parsers.check_trail_no_rem
9714         * Cg(parsers.heading_start, "level")
9715         * (C( parsers.optionalspace
9716             * parsers.hash^0
9717             * parsers.optionalspace
9718             * parsers.newline)
9719         + parsers.spacechar^1
9720         * 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`.

```

9721 M.reader = {}
9722 function M.reader.new(writer, options)
9723     local self = {}

```

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

```

9724     self.writer = writer
9725     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.

```

9726     self.parsers = {}
9727     (function(parsers)
9728         setmetatable(self.parsers, {
9729             __index = function (_, key)
9730                 return parsers[key]
9731             end
9732         })
9733     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.

```

9734     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.

```
9735 function self.normalize_tag(tag)
9736     tag = util.rope_to_string(tag)
9737     tag = tag:gsub("[\n\r\t]+", " ")
9738     tag = tag:gsub("^ ", ""):gsub(" $", "")
9739     local form = nil
9740     if options.unicodeNormalization then
9741         form = options.unicodeNormalizationForm
9742     end
9743     tag = util.casefold(tag, form)
9744     return tag
9745 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.

```
9746 local function iterlines(s, f)
9747     local rope = lpeg.match(Ct((parsers.line / f)^1), s)
9748     return util.rope_to_string(rope)
9749 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.

```
9750 if options.preserveTabs then
9751     self.expandtabs = function(s) return s end
9752 else
9753     self.expandtabs = function(s)
9754         if s:find("\t") then
9755             return iterlines(s, util.expand_tabs_in_line)
9756         else
9757             return s
9758         end
9759     end
9760 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.

```
9761 self.parser_functions = {}
9762 self.create_parser = function(name, grammar, topLevel)
9763     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.

```

9764     if toplevel and options.stripIndent then
9765         local min_prefix_length, min_prefix = nil, ''
9766         str = iterlines(str, function(line)
9767             if lpeg.match(parsers.nonemptyline, line) == nil then
9768                 return line
9769             end
9770             line = util.expand_tabs_in_line(line)
9771             local prefix = lpeg.match(C(parsers.optionalspace), line)
9772             local prefix_length = #prefix
9773             local is_shorter = min_prefix_length == nil
9774             if not is_shorter then
9775                 is_shorter = prefix_length < min_prefix_length
9776             end
9777             if is_shorter then
9778                 min_prefix_length, min_prefix = prefix_length, prefix
9779             end
9780             return line
9781         end)
9782         str = str:gsub('^' .. min_prefix, '')
9783     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.

```

9784     if toplevel and (options.texComments or options.hybrid) then
9785         str = lpeg.match(Ct(parsers.commented_line^1), str)
9786         str = util.rope_to_string(str)
9787     end
9788     local res = lpeg.match(grammar(), str)
9789     if res == nil then
9790         return writer.error(
9791             format("Parser `%s` failed to process the input text.", name),
9792             format("Here are the first 20 characters of the remaining "
9793                 .. "unprocessed text: `%s`.", str:sub(1,20))
9794         )
9795     else
9796         return res
9797     end
9798 end
9799 end
9800
9801 self.create_parser("parse_blocks",
9802     function()

```

```

9803         return parsers.blocks
9804     end, true)
9805
9806     self.create_parser("parse_blocks_nested",
9807         function()
9808             return parsers.blocks_nested
9809         end, false)
9810
9811     self.create_parser("parse_inlines",
9812         function()
9813             return parsers.inlines
9814         end, false)
9815
9816     self.create_parser("parse_inlines_no_inline_note",
9817         function()
9818             return parsers.inlines_no_inline_note
9819         end, false)
9820
9821     self.create_parser("parse_inlines_no_html",
9822         function()
9823             return parsers.inlines_no_html
9824         end, false)
9825
9826     self.create_parser("parse_inlines_nbsp",
9827         function()
9828             return parsers.inlines_nbsp
9829         end, false)
9830     self.create_parser("parse_inlines_no_link_or_emphasis",
9831         function()
9832             return parsers.inlines_no_link_or_emphasis
9833         end, false)

```

### 3.1.6.3 Parsers Used for Indentation (local)

The following patterns represent basic building blocks of indented content.

```

9834     parsers.minimally_indented_blankline
9835         = parsers.check_minimal_indent * (parsers.blankline / "")
9836
9837     parsers.minimally_indented_block
9838         = parsers.check_minimal_indent * V("Block")
9839
9840     parsers.minimally_indented_block_or_paragraph
9841         = parsers.check_minimal_indent * V("BlockOrParagraph")
9842
9843     parsers.minimally_indented_paragraph
9844         = parsers.check_minimal_indent * V("Paragraph")
9845

```

```

9846 parsers.minimally_indented_plain
9847     = parsers.check_minimal_indent * V("Plain")
9848
9849 parsers.minimally_indented_par_or_plain
9850     = parsers.minimally_indented_paragraph
9851     + parsers.minimally_indented_plain
9852
9853 parsers.minimally_indented_par_or_plain_no_blank
9854     = parsers.minimally_indented_par_or_plain
9855     - parsers.minimally_indented_blankline
9856
9857 parsers.minimally_indented_ref
9858     = parsers.check_minimal_indent * V("Reference")
9859
9860 parsers.minimally_indented_blank
9861     = parsers.check_minimal_indent * V("Blank")
9862
9863 parsers.conditionally_indented_blankline
9864     = parsers.check_minimal_blank_indent * (parsers.blankline / "")
9865
9866 parsers.minimally_indented_ref_or_block
9867     = parsers.minimally_indented_ref
9868     + parsers.minimally_indented_block
9869     - parsers.minimally_indented_blankline
9870
9871 parsers.minimally_indented_ref_or_block_or_par
9872     = parsers.minimally_indented_ref
9873     + parsers.minimally_indented_block_or_paragraph
9874     - parsers.minimally_indented_blankline
9875

```

The following pattern parses the properly indented content that follows the initial container start.

```

9876
9877 function parsers.separator_loop(separated_block, paragraph,
9878                                block_separator, paragraph_separator)
9879     return  separated_block
9880           + block_separator
9881           * paragraph
9882           * separated_block
9883           + paragraph_separator
9884           * paragraph
9885 end
9886
9887 function parsers.create_loop_body_pair(separated_block, paragraph,
9888                                       block_separator,
9889                                       paragraph_separator)

```

```

9890     return {
9891         block = parsers.separator_loop(separated_block, paragraph,
9892                                     block_separator, block_separator),
9893         par = parsers.separator_loop(separated_block, paragraph,
9894                                    block_separator, paragraph_separator)
9895     }
9896 end
9897
9898 parsers.block_sep_group = function(blank)
9899     return blank^0 * parsers.eof
9900         + ( blank^2 / writer.paragraphsep
9901           + blank^0 / writer.interblocksep
9902         )
9903 end
9904
9905 parsers.par_sep_group = function(blank)
9906     return blank^0 * parsers.eof
9907         + blank^0 / writer.paragraphsep
9908 end
9909
9910 parsers.sep_group_no_output = function(blank)
9911     return blank^0 * parsers.eof
9912         + blank^0
9913 end
9914
9915 parsers.content_blank = parsers.minimally_indented_blankline
9916
9917 parsers.ref_or_block_separated
9918     = parsers.sep_group_no_output(parsers.content_blank)
9919     * ( parsers.minimally_indented_ref
9920       - parsers.content_blank)
9921     + parsers.block_sep_group(parsers.content_blank)
9922     * ( parsers.minimally_indented_block
9923       - parsers.content_blank)
9924
9925 parsers.loop_body_pair =
9926     parsers.create_loop_body_pair(
9927         parsers.ref_or_block_separated,
9928         parsers.minimally_indented_par_or_plain_no_blank,
9929         parsers.block_sep_group(parsers.content_blank),
9930         parsers.par_sep_group(parsers.content_blank))
9931
9932 parsers.content_loop = ( V("Block")
9933                        * parsers.loop_body_pair.block^0
9934                        + (V("Paragraph") + V("Plain")))
9935                        * parsers.ref_or_block_separated
9936                        * parsers.loop_body_pair.block^0

```

```

9937         + (V("Paragraph") + V("Plain"))
9938         * parsers.loop_body_pair.par^0)
9939         * parsers.content_blank^0
9940
9941     parsers.indented_content = function()
9942         return Ct( (V("Reference") + (parsers.blankline / ""))
9943             * parsers.content_blank^0
9944             * parsers.check_minimal_indent
9945             * parsers.content_loop
9946             + (V("Reference") + (parsers.blankline / ""))
9947             * parsers.content_blank^0
9948             + parsers.content_loop)
9949     end
9950
9951     parsers.add_indent = function(pattern, name, breakable)
9952         return Cg(Cmt( Cb("indent_info")
9953             * Ct(pattern)
9954             * ( #parsers.linechar -- check if starter is blank
9955                 * Cc(false) + Cc(true))
9956             * Cc(name)
9957             * Cc(breakable),
9958                 process_starter_indent), "indent_info")
9959     end
9960

```

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

```

9961     if options.hashEnumerators then
9962         parsers.dig = parsers.digit + parsers.hash
9963     else
9964         parsers.dig = parsers.digit
9965     end
9966
9967     parsers.enumerator = function(delimiter_type, interrupting)
9968         local delimiter_range
9969         local allowed_end
9970         if interrupting then
9971             delimiter_range = P("1")
9972             allowed_end = C(parsers.spacechar^1) * #parsers.linechar
9973         else
9974             delimiter_range = parsers.dig * parsers.dig^-8
9975             allowed_end = C(parsers.spacechar^1)
9976                 + #(parsers.newline + parsers.eof)
9977         end
9978
9979         return parsers.check_trail
9980             * Ct(C(delimiter_range) * C(delimiter_type))

```

```

9981             * allowed_end
9982     end
9983
9984     parsers.starter = parsers.bullet(parsers.dash)
9985                     + parsers.bullet(parsers.asterisk)
9986                     + parsers.bullet(parsers.plus)
9987                     + parsers.enumerator(parsers.period)
9988                     + parsers.enumerator(parsers.rparent)
9989

```

### 3.1.6.5 Parsers Used for Blockquotes (local)

```

9990     parsers.blockquote_start
9991     = parsers.check_trail
9992     * C(parsers.more)
9993     * C(parsers.spacechar^0)
9994
9995     parsers.blockquote_body
9996     = parsers.add_indent(parsers.blockquote_start, "bq", true)
9997     * parsers.indented_content()
9998     * remove_indent("bq")
9999
10000     if not options.breakableBlockquotes then
10001         parsers.blockquote_body
10002         = parsers.add_indent(parsers.blockquote_start, "bq", false)
10003         * parsers.indented_content()
10004         * remove_indent("bq")
10005     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.

```

10006     local function parse_content_part(content_part)
10007         local rope = util.rope_to_string(content_part)
10008         local parsed
10009         = self.parser_functions.parse_inlines_no_link_or_emphasis(rope)
10010         parsed.indent_info = nil
10011         return parsed
10012     end
10013

```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

10014     local collect_emphasis_content =
10015         function(t, opening_index, closing_index)
10016             local content = {}
10017

```

```

10018     local content_part = {}
10019     for i = opening_index, closing_index do
10020         local value = t[i]
10021
10022         if value.rendered ~= nil then
10023             content[#content + 1] = parse_content_part(content_part)
10024             content_part = {}
10025             content[#content + 1] = value.rendered
10026             value.rendered = nil
10027         else
10028             if value.warning ~= nil then
10029                 if next(content_part) ~= nil then
10030                     content[#content + 1] = parse_content_part(content_part)
10031                 end
10032                 content_part = {}
10033
10034                 content[#content + 1] = value.warning
10035                 value.warning = nil
10036             end
10037             if value.type == "delimiter"
10038                 and value.element == "emphasis" then
10039                 if value.is_active then
10040                     content_part[#content_part + 1]
10041                         = string.rep(value.character, value.current_count)
10042                 end
10043             else
10044                 content_part[#content_part + 1] = value.content
10045             end
10046             value.content = ''
10047             value.is_active = false
10048         end
10049     end
10050
10051     if next(content_part) ~= nil then
10052         content[#content + 1] = parse_content_part(content_part)
10053     end
10054
10055     return content
10056 end
10057

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as emphasis.

```

10058 local function fill_emph(t, opening_index, closing_index)
10059     local content
10060     = collect_emphasis_content(t, opening_index + 1,
10061                               closing_index - 1)

```



```

10062     t[opening_index + 1].is_active = true
10063     t[opening_index + 1].rendered = writer.emphasis(content)
10064 end
10065

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as strong emphasis.

```

10066 local function fill_strong(t, opening_index, closing_index)
10067     local content
10068     = collect_emphasis_content(t, opening_index + 1,
10069                               closing_index - 1)
10070     t[opening_index + 1].is_active = true
10071     t[opening_index + 1].rendered = writer.strong(content)
10072 end
10073

```

Check whether the opening delimiter `opening_delimiter` and closing delimiter `closing_delimiter` break rule three together.

```

10074 local function breaks_three_rule(opening_delimiter, closing_delimiter)
10075     return ( opening_delimiter.is_closing
10076             or closing_delimiter.is_opening)
10077             and (( opening_delimiter.original_count
10078                   + closing_delimiter.original_count) % 3 == 0)
10079             and ( opening_delimiter.original_count % 3 ~= 0
10080                 or closing_delimiter.original_count % 3 ~= 0)
10081 end
10082

```

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`.

```

10083 local find_emphasis_opener = function(t, bottom_index, latest_index,
10084                                     character, closing_delimiter)
10085     for i = latest_index, bottom_index, -1 do
10086         local value = t[i]
10087         if value.is_active and
10088            value.is_opening and
10089            value.type == "delimiter" and
10090            value.element == "emphasis" and
10091            (value.character == character) and
10092            (value.current_count > 0) then
10093             if not breaks_three_rule(value, closing_delimiter) then
10094                 return i
10095             end
10096         end
10097     end
10098 end
10099

```

Iterate over the delimiters in the delimiter table `t`, producing emphasis or strong emphasis macros.

```
10100 local function process_emphasis(t, opening_index, closing_index)
10101   for i = opening_index, closing_index do
10102     local value = t[i]
10103     if value.type == "delimiter" and value.element == "emphasis" then
10104       local delimiter_length = string.len(value.content)
10105       value.character = string.sub(value.content, 1, 1)
10106       value.current_count = delimiter_length
10107       value.original_count = delimiter_length
10108     end
10109   end
10110
10111   local openers_bottom = {
10112     ['*'] = {
10113       [true] = {opening_index, opening_index, opening_index},
10114       [false] = {opening_index, opening_index, opening_index}
10115     },
10116     ['_'] = {
10117       [true] = {opening_index, opening_index, opening_index},
10118       [false] = {opening_index, opening_index, opening_index}
10119     }
10120   }
10121
10122   local current_position = opening_index
10123   local max_position = closing_index
10124
10125   while current_position <= max_position do
10126     local value = t[current_position]
10127
10128     if value.type ~= "delimiter" or
10129        value.element ~= "emphasis" or
10130        not value.is_active or
10131        not value.is_closing or
10132        (value.current_count <= 0) then
10133       current_position = current_position + 1
10134       goto continue
10135     end
10136
10137     local character = value.character
10138     local is_opening = value.is_opening
10139     local closing_length_modulo_three = value.original_count % 3
10140
10141     local current_openers_bottom
10142     = openers_bottom[character][is_opening]
10143       [closing_length_modulo_three + 1]
10144
```

```

10145     local opener_position
10146         = find_emphasis_opener(t, current_openers_bottom,
10147                                 current_position - 1, character, value)
10148
10149     if (opener_position == nil) then
10150         openers_bottom[character][is_opening]
10151             [closing_length_modulo_three + 1]
10152             = current_position
10153         current_position = current_position + 1
10154         goto continue
10155     end
10156
10157     local opening_delimiter = t[opener_position]
10158
10159     local current_opening_count = opening_delimiter.current_count
10160     local current_closing_count = t[current_position].current_count
10161
10162     if (current_opening_count >= 2)
10163         and (current_closing_count >= 2) then
10164         opening_delimiter.current_count = current_opening_count - 2
10165         t[current_position].current_count = current_closing_count - 2
10166         fill_strong(t, opener_position, current_position)
10167     else
10168         opening_delimiter.current_count = current_opening_count - 1
10169         t[current_position].current_count = current_closing_count - 1
10170         fill_emph(t, opener_position, current_position)
10171     end
10172
10173     ::continue::
10174 end
10175 end
10176
10177 parsers.delimiter_run = function(character)
10178     return (B(parsers.backslash * character) + -B(character))
10179         * character~1
10180         * -#character
10181 end
10182
10183 parsers.left_flanking_delimiter_run = function(character)
10184     return (B( parsers.any)
10185         * ( parsers.unicode.preceding_punctuation
10186           + parsers.unicode.preceding_whitespace)
10187         + -B(parsers.any))
10188         * parsers.delimiter_run(character)
10189         * parsers.unicode.following_punctuation
10190         + parsers.delimiter_run(character)
10191         * -#( parsers.unicode.following_punctuation

```

```

10192             + parsers.unicode.following_whitespace
10193             + parsers.eof)
10194     end
10195
10196     parsers.right_flanking_delimiter_run = function(character)
10197         return parsers.unicode.preceding_punctuation
10198             * parsers.delimiter_run(character)
10199             * ( parsers.unicode.following_punctuation
10200               + parsers.unicode.following_whitespace
10201               + parsers.eof)
10202             + (B(parsers.any)
10203               * -( parsers.unicode.preceding_punctuation
10204                  + parsers.unicode.preceding_whitespace))
10205             * parsers.delimiter_run(character)
10206     end
10207
10208     if options.underscores then
10209         parsers.emph_start
10210             = parsers.left_flanking_delimiter_run(parsers.asterisk)
10211             + ( -#parsers.right_flanking_delimiter_run(parsers.underscore)
10212               + ( parsers.unicode.preceding_punctuation
10213                 * #parsers.right_flanking_delimiter_run(parsers.underscore)))
10214             * parsers.left_flanking_delimiter_run(parsers.underscore)
10215
10216         parsers.emph_end
10217             = parsers.right_flanking_delimiter_run(parsers.asterisk)
10218             + ( -#parsers.left_flanking_delimiter_run(parsers.underscore)
10219               + #( parsers.left_flanking_delimiter_run(parsers.underscore)
10220                  * parsers.unicode.following_punctuation))
10221             * parsers.right_flanking_delimiter_run(parsers.underscore)
10222     else
10223         parsers.emph_start
10224             = parsers.left_flanking_delimiter_run(parsers.asterisk)
10225
10226         parsers.emph_end
10227             = parsers.right_flanking_delimiter_run(parsers.asterisk)
10228     end
10229
10230     parsers.emph_capturing_open_and_close
10231         = #parsers.emph_start * #parsers.emph_end
10232         * Ct( Cg(Cc("delimiter"), "type")
10233             * Cg(Cc("emphasis"), "element")
10234             * Cg(C(parsers.emph_start), "content")
10235             * Cg(Cc(true), "is_opening")
10236             * Cg(Cc(true), "is_closing"))
10237
10238     parsers.emph_capturing_open = Ct( Cg(Cc("delimiter"), "type")

```

```

10239             * Cg(Cc("emphasis"), "element")
10240             * Cg(C(parsers.emph_start), "content")
10241             * Cg(Cc(true), "is_opening")
10242             * Cg(Cc(false), "is_closing"))
10243
10244     parsers.emph_capturing_close = Ct( Cg(Cc("delimiter"), "type")
10245             * Cg(Cc("emphasis"), "element")
10246             * Cg(C(parsers.emph_end), "content")
10247             * Cg(Cc(false), "is_opening")
10248             * Cg(Cc(true), "is_closing"))
10249
10250     parsers.emph_open_or_close = parsers.emph_capturing_open_and_close
10251             + parsers.emph_capturing_open
10252             + parsers.emph_capturing_close
10253
10254     parsers.emph_open = parsers.emph_capturing_open_and_close
10255             + parsers.emph_capturing_open
10256
10257     parsers.emph_close = parsers.emph_capturing_open_and_close
10258             + parsers.emph_capturing_close
10259

```

### 3.1.6.7 Helpers for Links and Link Reference Definitions (local)

```

10260     -- List of references defined in the document
10261     local references
10262
10263     -- List of note references defined in the document
10264     parsers.rawnotes = {}
10265

```

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.

```

10266     function self.register_link(_, tag, url, title,
10267             attributes)
10268         local normalized_tag = self.normalize_tag(tag)
10269         if references[normalized_tag] == nil then
10270             references[normalized_tag] = {
10271                 url = url,
10272                 title = title,
10273                 attributes = attributes
10274             }
10275             return ""
10276         else
10277             local text
10278                 = string.format('Multiply defined link reference "%s"', tag)
10279             local more = string.format("Look for the text `[%s]: ...`.", tag)

```

```

10280         return writer.warning(text, more)
10281     end
10282 end
10283

```

The `reader->lookup_reference` method looks up a reference with link label `tag`.

```

10284 function self.lookup_reference(tag)
10285     return references[self.normalize_tag(tag)]
10286 end
10287

```

The `reader->lookup_note_reference` method looks up a note reference with label `tag`.

```

10288 function self.lookup_note_reference(tag)
10289     return parsers.rawnotes[self.normalize_tag(tag)]
10290 end
10291
10292 parsers.title_s_direct_ref = parsers.squote
10293                             * Cs((parsers.html_entities
10294                                 + ( parsers.anyescaped
10295                                   - parsers.squote
10296                                   - parsers.blankline^2))^0)
10297                             * parsers.squote
10298
10299 parsers.title_d_direct_ref = parsers.dquote
10300                             * Cs((parsers.html_entities
10301                                 + ( parsers.anyescaped
10302                                   - parsers.dquote
10303                                   - parsers.blankline^2))^0)
10304                             * parsers.dquote
10305
10306 parsers.title_p_direct_ref = parsers.lparent
10307                             * Cs((parsers.html_entities
10308                                 + ( parsers.anyescaped
10309                                   - parsers.lparent
10310                                   - parsers.rparent
10311                                   - parsers.blankline^2))^0)
10312                             * parsers.rparent
10313
10314 parsers.title_direct_ref = parsers.title_s_direct_ref
10315                           + parsers.title_d_direct_ref
10316                           + parsers.title_p_direct_ref
10317
10318 parsers.inline_direct_ref_inside = parsers.lparent * parsers.spnl
10319                                   * Cg(parsers.url + Cc(""), "url")
10320                                   * parsers.spnl
10321                                   * Cg( parsers.title_direct_ref
10322                                       + Cc(""), "title")

```

```

10323                                     * parsers.spnl * parsers.rparent
10324
10325 parsers.inline_direct_ref = parsers.lparent * parsers.spnlc
10326                             * Cg(parsers.url + Cc(""), "url")
10327                             * parsers.spnlc
10328                             * Cg(parsers.title + Cc(""), "title")
10329                             * parsers.spnlc * parsers.rparent
10330
10331 parsers.empty_link = parsers.lbracket
10332                     * parsers.rbracket
10333
10334 parsers.inline_link = parsers.link_text
10335                     * parsers.inline_direct_ref
10336
10337 parsers.full_link = parsers.link_text
10338                   * parsers.link_label
10339
10340 parsers.shortcut_link = parsers.link_label
10341                       * -(parsers.empty_link + parsers.link_label)
10342
10343 parsers.collapsed_link = parsers.link_label
10344                       * parsers.empty_link
10345
10346 parsers.image_opening = #(parsers.exclamation * parsers.inline_link)
10347                       * Cg(Cc("inline"), "link_type")
10348                       + #(parsers.exclamation * parsers.full_link)
10349                       * Cg(Cc("full"), "link_type")
10350                       + #( parsers.exclamation
10351                          * parsers.collapsed_link)
10352                       * Cg(Cc("collapsed"), "link_type")
10353                       + #(parsers.exclamation * parsers.shortcut_link)
10354                       * Cg(Cc("shortcut"), "link_type")
10355                       + #(parsers.exclamation * parsers.empty_link)
10356                       * Cg(Cc("empty"), "link_type")
10357
10358 parsers.link_opening = #parsers.inline_link
10359                       * Cg(Cc("inline"), "link_type")
10360                       + #parsers.full_link
10361                       * Cg(Cc("full"), "link_type")
10362                       + #parsers.collapsed_link
10363                       * Cg(Cc("collapsed"), "link_type")
10364                       + #parsers.shortcut_link
10365                       * Cg(Cc("shortcut"), "link_type")
10366                       + #parsers.empty_link
10367                       * Cg(Cc("empty_link"), "link_type")
10368                       + #parsers.link_text
10369                       * Cg(Cc("link_text"), "link_type")

```

```

10370
10371 parsers.note_opening = #(parsers.circumflex * parsers.link_text)
10372                       * Cg(Cc("note_inline"), "link_type")
10373
10374 parsers.raw_note_opening = #( parsers.lbracket
10375                               * parsers.circumflex
10376                               * parsers.link_label_body
10377                               * parsers.rbracket)
10378                               * Cg(Cc("raw_note"), "link_type")
10379
10380 local inline_note_element = Cg(Cc("note"), "element")
10381                             * parsers.note_opening
10382                             * Cg( parsers.circumflex
10383                                 * parsers.lbracket, "content")
10384
10385 local image_element = Cg(Cc("image"), "element")
10386                       * parsers.image_opening
10387                       * Cg( parsers.exclamation
10388                           * parsers.lbracket, "content")
10389
10390 local note_element = Cg(Cc("note"), "element")
10391                     * parsers.raw_note_opening
10392                     * Cg( parsers.lbracket
10393                         * parsers.circumflex, "content")
10394
10395 local link_element = Cg(Cc("link"), "element")
10396                     * parsers.link_opening
10397                     * Cg(parsers.lbracket, "content")
10398
10399 local opening_elements = parsers.fail
10400
10401 if options.inlineNotes then
10402   opening_elements = opening_elements + inline_note_element
10403 end
10404
10405 opening_elements = opening_elements + image_element
10406
10407 if options.notes then
10408   opening_elements = opening_elements + note_element
10409 end
10410
10411 opening_elements = opening_elements + link_element
10412
10413 parsers.link_image_opening = Ct( Cg(Cc("delimiter"), "type")
10414                                 * Cg(Cc(true), "is_opening")
10415                                 * Cg(Cc(false), "is_closing")
10416                                 * opening_elements)

```



```

10417
10418 parsers.link_image_closing = Ct( Cg(Cc("delimiter"), "type")
10419                                * Cg(Cc("link"), "element")
10420                                * Cg(Cc(false), "is_opening")
10421                                * Cg(Cc(true), "is_closing")
10422                                * ( Cg(Cc(true), "is_direct")
10423                                * Cg( parsers.rbracket
10424                                    * #parsers.inline_direct_ref,
10425                                    "content")
10426                                + Cg(Cc(false), "is_direct")
10427                                * Cg(parsers.rbracket, "content")))
10428
10429 parsers.link_image_open_or_close = parsers.link_image_opening
10430                                + parsers.link_image_closing
10431
10432 if options.html then
10433     parsers.link_emph_precedence = parsers.inticks
10434                                + parsers.autolink
10435                                + parsers.html_inline_tags
10436 else
10437     parsers.link_emph_precedence = parsers.inticks
10438                                + parsers.autolink
10439 end
10440
10441 parsers.link_and_emph_endline = parsers.newline
10442                                * ((parsers.check_minimal_indent
10443                                * -V("EndlineExceptions")
10444                                + parsers.check_optional_indent
10445                                * -V("EndlineExceptions")
10446                                * -V("ListStarter"))) / "")
10447                                * parsers.spacechar^0 / "\n"
10448
10449 parsers.link_and_emph_content
10450     = Ct( Cg(Cc("content"), "type")
10451          * Cg(Cs(( parsers.link_emph_precedence
10452                    + parsers.backslash * parsers.linechar
10453                    + parsers.link_and_emph_endline
10454                    + (parsers.linechar
10455                      - parsers.blankline^2
10456                      - parsers.link_image_open_or_close
10457                      - parsers.emph_open_or_close))^0), "content"))
10458
10459 parsers.link_and_emph_table
10460     = (parsers.link_image_opening + parsers.emph_open)
10461     * parsers.link_and_emph_content
10462     * ((parsers.link_image_open_or_close + parsers.emph_open_or_close)
10463        * parsers.link_and_emph_content)^1

```

10464

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
10465 local function collect_link_content(t, opening_index, closing_index)
10466     local content = {}
10467     for i = opening_index, closing_index do
10468         content[#content + 1] = t[i].content
10469     end
10470     return util.rope_to_string(content)
10471 end
10472
```

Look for the closest potential link opener in the delimiter table `t` in the range from `bottom_index` to `latest_index`.

```
10473 local function find_link_opener(t, bottom_index, latest_index)
10474     for i = latest_index, bottom_index, -1 do
10475         local value = t[i]
10476         if value.type == "delimiter" and
10477             value.is_opening and
10478             ( value.element == "link"
10479             or value.element == "image"
10480             or value.element == "note")
10481             and not value.removed then
10482             if value.is_active then
10483                 return i
10484             end
10485             value.removed = true
10486             return nil
10487         end
10488     end
10489 end
10490
```

Find the position of a delimiter that closes a full link after an an index `latest_index` in the delimiter table `t`.

```
10491 local function find_next_link_closing_index(t, latest_index)
10492     for i = latest_index, #t do
10493         local value = t[i]
10494         if value.is_closing and
10495             value.element == "link" and
10496             not value.removed then
10497             return i
10498         end
10499     end
10500 end
10501
```

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.

```
10502 local function disable_previous_link_openers(t, opening_index)
10503   if t[opening_index].element == "image" then
10504     return
10505   end
10506
10507   for i = opening_index, 1, -1 do
10508     local value = t[i]
10509     if value.is_active and
10510        value.type == "delimiter" and
10511        value.is_opening and
10512        value.element == "link" then
10513       value.is_active = false
10514     end
10515   end
10516 end
10517
```

Disable the delimiters between the `opening_index` and `closing_index` in the delimiter table `t` by marking them inactive with the `is_active` property.

```
10518 local function disable_range(t, opening_index, closing_index)
10519   for i = opening_index, closing_index do
10520     local value = t[i]
10521     if value.is_active then
10522       value.is_active = false
10523       if value.type == "delimiter" then
10524         value.removed = true
10525       end
10526     end
10527   end
10528 end
10529
```

Clear the parsed content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
10530 local delete_parsed_content_in_range =
10531   function(t, opening_index, closing_index)
10532     for i = opening_index, closing_index do
10533       t[i].rendered = nil
10534     end
10535   end
10536
```

Clear the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
10537 local function empty_content_in_range(t, opening_index, closing_index)
10538   for i = opening_index, closing_index do
```

```

10539     t[i].content = ''
10540   end
10541 end
10542

```

Join the attributes from the link reference definition `reference_attributes` with the link's own attributes `own_attributes`.

```

10543   local function join_attributes(reference_attributes, own_attributes)
10544     local merged_attributes = {}
10545     for _, attribute in ipairs(reference_attributes or {}) do
10546       table.insert(merged_attributes, attribute)
10547     end
10548     for _, attribute in ipairs(own_attributes or {}) do
10549       table.insert(merged_attributes, attribute)
10550     end
10551     if next(merged_attributes) == nil then
10552       merged_attributes = nil
10553     end
10554     return merged_attributes
10555   end
10556

```

Parse content between two delimiters in the delimiter table `t`. Produce the respective link and image macros.

```

10557   local render_link_or_image =
10558     function(t, opening_index, closing_index, content_end_index,
10559             reference)
10560       process_emphasis(t, opening_index, content_end_index)
10561       local mapped = collect_emphasis_content(t, opening_index + 1,
10562                                             content_end_index - 1)
10563
10564       local rendered = {}
10565       if (t[opening_index].element == "link") then
10566         rendered = writer.link(mapped, reference.url,
10567                               reference.title, reference.attributes)
10568       end
10569
10570       if (t[opening_index].element == "image") then
10571         rendered = writer.image(mapped, reference.url, reference.title,
10572                                reference.attributes)
10573       end
10574
10575       if (t[opening_index].element == "note") then
10576         if (t[opening_index].link_type == "note_inline") then
10577           rendered = writer.note(mapped)
10578         end
10579         if (t[opening_index].link_type == "raw_note") then
10580           rendered = writer.note(reference)

```

```

10581         end
10582     end
10583
10584     t[opening_index].rendered = rendered
10585     delete_parsed_content_in_range(t, opening_index + 1,
10586                                   closing_index)
10587     empty_content_in_range(t, opening_index, closing_index)
10588     disable_previous_link_openers(t, opening_index)
10589     disable_range(t, opening_index, closing_index)
10590 end
10591

```

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.

```

10592 local resolve_inline_following_content =
10593     function(t, closing_index, match_reference, match_link_attributes)
10594         local content = ""
10595         for i = closing_index + 1, #t do
10596             content = content .. t[i].content
10597         end
10598
10599         local matching_content = parsers.succeed
10600
10601         if match_reference then
10602             matching_content = matching_content
10603                 * parsers.inline_direct_ref_inside
10604         end
10605
10606         if match_link_attributes then
10607             matching_content = matching_content
10608                 * Cg(Ct(parsers.attributes^-1), "attributes")
10609         end
10610
10611         local matched = lpeg.match(Ct( matching_content
10612                                       * Cg(Cp(), "end_position")), content)
10613
10614         local matched_count = matched.end_position - 1
10615         for i = closing_index + 1, #t do
10616             local value = t[i]
10617
10618             local chars_left = matched_count
10619             matched_count = matched_count - #value.content
10620
10621             if matched_count <= 0 then
10622                 value.content = value.content:sub(chars_left + 1)
10623                 break

```

```

10624         end
10625
10626         value.content = ''
10627         value.is_active = false
10628     end
10629
10630     local attributes = matched.attributes
10631     if attributes == nil or next(attributes) == nil then
10632         attributes = nil
10633     end
10634
10635     return {
10636         url = matched.url or "",
10637         title = matched.title or "",
10638         attributes = attributes
10639     }
10640 end
10641

```

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.

```

10642     local function resolve_inline_link(t, opening_index, closing_index)
10643         local inline_content
10644         = resolve_inline_following_content(t, closing_index, true,
10645                                           t.match_link_attributes)
10646         render_link_or_image(t, opening_index, closing_index,
10647                             closing_index, inline_content)
10648     end
10649

```

Resolve an inline note `^[a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`.

```

10650     local resolve_note_inline_link =
10651     function(t, opening_index, closing_index)
10652         local inline_content
10653         = resolve_inline_following_content(t, closing_index,
10654                                           false, false)
10655         render_link_or_image(t, opening_index, closing_index,
10656                             closing_index, inline_content)
10657     end
10658

```

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.

```

10659     local function resolve_shortcut_link(t, opening_index, closing_index)
10660         local content

```

```

10661     = collect_link_content(t, opening_index + 1, closing_index - 1)
10662 local r = self.lookup_reference(content)
10663
10664 if r then
10665     local inline_content
10666     = resolve_inline_following_content(t, closing_index, false,
10667                                       t.match_link_attributes)
10668     r.attributes
10669     = join_attributes(r.attributes, inline_content.attributes)
10670     render_link_or_image(t, opening_index, closing_index,
10671                         closing_index, r)
10672 end
10673 end
10674

```

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.

```

10675 local function resolve_raw_note_link(t, opening_index, closing_index)
10676     local content
10677     = collect_link_content(t, opening_index + 1, closing_index - 1)
10678     local r = self.lookup_note_reference(content)
10679
10680     if r then
10681         local parsed_ref = self.parser_functions.parse_blocks_nested(r)
10682         render_link_or_image(t, opening_index, closing_index,
10683                             closing_index, parsed_ref)
10684     end
10685 end
10686

```

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.

```

10687 local function resolve_full_link(t, opening_index, closing_index)
10688     local next_link_closing_index
10689     = find_next_link_closing_index(t, closing_index + 4)
10690     local next_link_content
10691     = collect_link_content(t, closing_index + 3,
10692                           next_link_closing_index - 1)
10693     local r = self.lookup_reference(next_link_content)
10694
10695     if r then
10696         local inline_content
10697         = resolve_inline_following_content(t, next_link_closing_index,
10698                                           false,
10699                                           t.match_link_attributes)
10700         r.attributes
10701         = join_attributes(r.attributes, inline_content.attributes)
10702         render_link_or_image(t, opening_index, next_link_closing_index,

```

```

10703             closing_index, r)
10704     else
10705         local text = string.format('Undefined link reference "%s"',
10706                                   next_link_content)
10707         local more = string.format("Look for the text `[...] [%s]`.",
10708                                   next_link_content)
10709         t[opening_index].warning = writer.warning(text, more)
10710     end
10711 end
10712

```

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.

```

10713 local function resolve_collapsed_link(t, opening_index, closing_index)
10714     local next_link_closing_index
10715     = find_next_link_closing_index(t, closing_index + 4)
10716     local content
10717     = collect_link_content(t, opening_index + 1, closing_index - 1)
10718     local r = self.lookup_reference(content)
10719
10720     if r then
10721         local inline_content
10722         = resolve_inline_following_content(t, closing_index, false,
10723                                           t.match_link_attributes)
10724         r.attributes
10725         = join_attributes(r.attributes, inline_content.attributes)
10726         render_link_or_image(t, opening_index, next_link_closing_index,
10727                             closing_index, r)
10728     else
10729         local text = string.format('Undefined link reference "%s"',
10730                                   content)
10731         local more = string.format("Look for the text `[...] [%s]`.",
10732                                   content)
10733         t[opening_index].warning = writer.warning(text, more)
10734     end
10735 end
10736

```

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.

```

10737 local function process_links_and_emphasis(t)
10738     for _,value in ipairs(t) do
10739         value.is_active = true
10740     end

```



```

10741
10742     for i,value in ipairs(t) do
10743         if not value.is_closing
10744             or value.type ~= "delimiter"
10745             or not ( value.element == "link"
10746                     or value.element == "image"
10747                     or value.element == "note")
10748             or value.removed then
10749             goto continue
10750         end
10751
10752         local opener_position = find_link_opener(t, 1, i - 1)
10753         if (opener_position == nil) then
10754             goto continue
10755         end
10756
10757         local opening_delimiter = t[opener_position]
10758         opening_delimiter.removed = true
10759
10760         local link_type = opening_delimiter.link_type
10761
10762         if (link_type == "inline") then
10763             resolve_inline_link(t, opener_position, i)
10764         end
10765         if (link_type == "shortcut") then
10766             resolve_shortcut_link(t, opener_position, i)
10767         end
10768         if (link_type == "full") then
10769             resolve_full_link(t, opener_position, i)
10770         end
10771         if (link_type == "collapsed") then
10772             resolve_collapsed_link(t, opener_position, i)
10773         end
10774         if (link_type == "note_inline") then
10775             resolve_note_inline_link(t, opener_position, i)
10776         end
10777         if (link_type == "raw_note") then
10778             resolve_raw_note_link(t, opener_position, i)
10779         end
10780
10781         ::continue::
10782     end
10783
10784     t[#t].content = t[#t].content:gsub("%s*$","")
10785
10786     process_emphasis(t, 1, #t)
10787     local final_result = collect_emphasis_content(t, 1, #t)

```

```

10788     return final_result
10789 end
10790
10791 function self.defer_link_and_emphasis_processing(delimiter_table)
10792     return writer.defer_call(function()
10793         return process_links_and_emphasis(delimiter_table)
10794     end)
10795 end
10796

```

### 3.1.6.8 Inline Elements (local)

```

10797 parsers.Str      = ( parsers.normalchar
10798                     * (parsers.normalchar + parsers.at)^0)
10799                     / writer.string
10800
10801 parsers.Symbol   = (parsers.backtick^1 + V("SpecialChar"))
10802                     / writer.string
10803
10804 parsers.Ellipsis = P("...") / writer.ellipsis
10805
10806 parsers.Smart    = parsers.Ellipsis
10807
10808 parsers.Code     = parsers.inticks / writer.code
10809
10810 if options.blankBeforeBlockquote then
10811     parsers.bqstart = parsers.fail
10812 else
10813     parsers.bqstart = parsers.blockquote_start
10814 end
10815
10816 if options.blankBeforeHeading then
10817     parsers.headerstart = parsers.fail
10818 else
10819     parsers.headerstart = parsers.atx_heading
10820 end
10821
10822 if options.blankBeforeList then
10823     parsers.interrupting_bullets = parsers.fail
10824     parsers.interrupting_enumerators = parsers.fail
10825 else
10826     parsers.interrupting_bullets
10827         = parsers.bullet(parsers.dash, true)
10828         + parsers.bullet(parsers.asterisk, true)
10829         + parsers.bullet(parsers.plus, true)
10830
10831     parsers.interrupting_enumerators

```

```

10832         = parsers.enumerator(parsers.period, true)
10833         + parsers.enumerator(parsers.rparent, true)
10834     end
10835
10836     if options.html then
10837         parsers.html_interrupting
10838         = parsers.check_trail
10839         * ( parsers.html_incomplete_open_tag
10840           + parsers.html_incomplete_close_tag
10841           + parsers.html_incomplete_open_special_tag
10842           + parsers.html_comment_start
10843           + parsers.html_cdatasection_start
10844           + parsers.html_declaration_start
10845           + parsers.html_instruction_start
10846           - parsers.html_close_special_tag
10847           - parsers.html_empty_special_tag)
10848     else
10849         parsers.html_interrupting = parsers.fail
10850     end
10851
10852     parsers.ListStarter = parsers.starter
10853
10854     parsers.EndlineExceptions
10855         = parsers.blankline -- paragraph break
10856         + parsers.eof       -- end of document
10857         + parsers.bqstart
10858         + parsers.thematic_break_lines
10859         + parsers.interrupting_bullets
10860         + parsers.interrupting_enumerators
10861         + parsers.headerstart
10862         + parsers.html_interrupting
10863
10864     parsers.NoSoftLineBreakEndlineExceptions = parsers.EndlineExceptions
10865
10866     parsers.endline = parsers.newline
10867         * (parsers.check_minimal_indent
10868           * -V("EndlineExceptions")
10869           + parsers.check_optional_indent
10870           * -V("EndlineExceptions")
10871           * -V("ListStarter")) / function(_) return end
10872         * parsers.spacechar~0
10873
10874     parsers.Endline = parsers.endline
10875         / writer.soft_line_break
10876
10877     parsers.EndlineNoSub = parsers.endline
10878

```

```

10879 parsers.NoSoftLineBreakEndline
10880         = parsers.newline
10881         * (parsers.check_minimal_indent
10882         * -V("NoSoftLineBreakEndlineExceptions")
10883         + parsers.check_optional_indent
10884         * -V("NoSoftLineBreakEndlineExceptions")
10885         * -V("ListStarter"))
10886         * parsers.spacechar^0
10887         / writer.space
10888
10889 parsers.EndlineBreak = parsers.backslash * parsers.endline
10890                       / writer.hard_line_break
10891
10892 parsers.OptionalIndent
10893         = parsers.spacechar^1 / writer.space
10894
10895 parsers.Space       = parsers.spacechar^2 * parsers.endline
10896                       / writer.hard_line_break
10897         + parsers.spacechar^1
10898         * parsers.endline^-1
10899         * parsers.eof / self.expandtabs
10900         + parsers.spacechar^1 * parsers.endline
10901                       / writer.soft_line_break
10902         + parsers.spacechar^1
10903         * -parsers.newline / self.expandtabs
10904         + parsers.spacechar^1
10905
10906 parsers.NoSoftLineBreakSpace
10907         = parsers.spacechar^2 * parsers.endline
10908                       / writer.hard_line_break
10909         + parsers.spacechar^1
10910         * parsers.endline^-1
10911         * parsers.eof / self.expandtabs
10912         + parsers.spacechar^1 * parsers.endline
10913                       / writer.soft_line_break
10914         + parsers.spacechar^1
10915         * -parsers.newline / self.expandtabs
10916         + parsers.spacechar^1
10917
10918 parsers.NonbreakingEndline
10919         = parsers.endline
10920         / writer.nbsp
10921
10922 parsers.NonbreakingSpace
10923         = parsers.spacechar^2 * parsers.endline
10924                       / writer.nbsp
10925         + parsers.spacechar^1

```

```

10926             * parsers.endline~-1 * parsers.eof / ""
10927             + parsers.spacechar~1 * parsers.endline
10928                     * parsers.optionalspace
10929                     / writer.nbsp
10930             + parsers.spacechar~1 * parsers.optionalspace
10931                     / writer.nbsp
10932

```

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.

```

10933 function self.auto_link_url(url, attributes)
10934   return writer.link(writer.escape(url),
10935                     url, nil, attributes)
10936 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.

```

10937 function self.auto_link_email(email, attributes)
10938   return writer.link(writer.escape(email),
10939                     "mailto:".email,
10940                     nil, attributes)
10941 end
10942
10943 parsers.AutoLinkUrl = parsers.auto_link_url
10944                     / self.auto_link_url
10945
10946 parsers.AutoLinkEmail
10947                     = parsers.auto_link_email
10948                     / self.auto_link_email
10949
10950 parsers.AutoLinkRelativeReference
10951                     = parsers.auto_link_relative_reference
10952                     / self.auto_link_url
10953
10954 parsers.LinkAndEmph = Ct(parsers.link_and_emph_table)
10955                     / self.defer_link_and_emphasis_processing
10956
10957 parsers.EscapedChar = parsers.backslash
10958                     * C(parsers escapable) / writer.string
10959
10960 parsers.InlineHtml = Cs(parsers.html_inline_comment)
10961                     / writer.inline_html_comment
10962                     + Cs(parsers.html_any_empty_inline_tag
10963                       + parsers.html_inline_instruction
10964                       + parsers.html_inline_cdatasection)

```

```

10965         + parsers.html_inline_declaration
10966         + parsers.html_any_open_inline_tag
10967         + parsers.html_any_close_tag)
10968     / writer.inline_html_tag
10969
10970     parsers.HtmlEntity = parsers.html_entities / writer.string

```

### 3.1.6.9 Block Elements (local)

```

10971     parsers.DisplayHtml = Cs(parsers.check_trail
10972         * ( parsers.html_comment
10973         + parsers.html_special_block
10974         + parsers.html_block
10975         + parsers.html_any_block
10976         + parsers.html_instruction
10977         + parsers.html_cdatasection
10978         + parsers.html_declaration))
10979     / writer.block_html_element
10980
10981     parsers.indented_non_blank_line = parsers.indentedline
10982         - parsers.blankline
10983
10984     parsers.Verbatim
10985     = Cs( parsers.check_code_trail
10986         * (parsers.line - parsers.blankline)
10987         * (( parsers.check_minimal_blank_indent_and_full_code_trail
10988         * parsers.blankline)^0
10989         * ( (parsers.check_minimal_indent / "")
10990         * parsers.check_code_trail
10991         * (parsers.line - parsers.blankline))^1)^0)
10992     / self.expandtabs / writer.verbatim
10993
10994     parsers.Blockquote    = parsers.blockquote_body
10995         / writer.blockquote
10996
10997     parsers.ThematicBreak = parsers.thematic_break_lines
10998         / writer.thematic_break
10999
11000     parsers.Reference     = parsers.define_reference_parser
11001         / self.register_link
11002
11003     parsers.Paragraph     = parsers.freeze_trail
11004         * (Ct((parsers.Inline)^1)
11005         * (parsers.newline + parsers.eof)
11006         * parsers.unfreeze_trail
11007         / writer.paragraph)
11008

```

```

11009 parsers.Plain          = parsers.nonindentspace * Ct(parsers.Inline~1)
11010                        / writer.plain

```

### 3.1.6.10 Lists (local)

```

11011
11012 if options.taskLists then
11013   parsers.tickbox = ( parsers.ticked_box
11014                       + parsers.halfticked_box
11015                       + parsers.unticked_box
11016                     ) / writer.tickbox
11017 else
11018   parsers.tickbox = parsers.fail
11019 end
11020
11021 parsers.list_blank = parsers.conditionally_indented_blankline
11022
11023 parsers.ref_or_block_list_separated
11024   = parsers.sep_group_no_output(parsers.list_blank)
11025   * parsers.minimally_indented_ref
11026   + parsers.block_sep_group(parsers.list_blank)
11027   * parsers.minimally_indented_block
11028
11029 parsers.ref_or_block_non_separated
11030   = parsers.minimally_indented_ref
11031   + (parsers.succeed / writer.interblocksep)
11032   * parsers.minimally_indented_block
11033   - parsers.minimally_indented_blankline
11034
11035 parsers.tight_list_loop_body_pair =
11036   parsers.create_loop_body_pair(
11037     parsers.ref_or_block_non_separated,
11038     parsers.minimally_indented_par_or_plain_no_blank,
11039     (parsers.succeed / writer.interblocksep),
11040     (parsers.succeed / writer.paragraphsep))
11041
11042 parsers.loose_list_loop_body_pair =
11043   parsers.create_loop_body_pair(
11044     parsers.ref_or_block_list_separated,
11045     parsers.minimally_indented_par_or_plain,
11046     parsers.block_sep_group(parsers.list_blank),
11047     parsers.par_sep_group(parsers.list_blank))
11048
11049 parsers.tight_list_content_loop
11050   = V("Block")
11051   * parsers.tight_list_loop_body_pair.block~0
11052   + (V("Paragraph") + V("Plain"))

```

```

11053     * parsers.ref_or_block_non_separated
11054     * parsers.tight_list_loop_body_pair.block^0
11055     + (V("Paragraph") + V("Plain"))
11056     * parsers.tight_list_loop_body_pair.par^0
11057
11058     parsers.loose_list_content_loop
11059     = V("Block")
11060     * parsers.loose_list_loop_body_pair.block^0
11061     + (V("Paragraph") + V("Plain"))
11062     * parsers.ref_or_block_list_separated
11063     * parsers.loose_list_loop_body_pair.block^0
11064     + (V("Paragraph") + V("Plain"))
11065     * parsers.loose_list_loop_body_pair.par^0
11066
11067     parsers.list_item_tightness_condition
11068     = -#( parsers.list_blank^0
11069         * parsers.minimally_indented_ref_or_block_or_par)
11070     * remove_indent("li")
11071     + remove_indent("li")
11072     * parsers.fail
11073
11074     parsers.indented_content_tight
11075     = Ct( (parsers.blankline / "")
11076         * #parsers.list_blank
11077         * remove_indent("li")
11078         + ( (V("Reference") + (parsers.blankline / ""))
11079             * parsers.check_minimal_indent
11080             * parsers.tight_list_content_loop
11081             + (V("Reference") + (parsers.blankline / ""))
11082             + (parsers.tickbox^-1 / writer.escape)
11083             * parsers.tight_list_content_loop
11084             )
11085         * parsers.list_item_tightness_condition)
11086
11087     parsers.indented_content_loose
11088     = Ct( (parsers.blankline / "")
11089         * #parsers.list_blank
11090         + ( (V("Reference") + (parsers.blankline / ""))
11091             * parsers.check_minimal_indent
11092             * parsers.loose_list_content_loop
11093             + (V("Reference") + (parsers.blankline / ""))
11094             + (parsers.tickbox^-1 / writer.escape)
11095             * parsers.loose_list_content_loop))
11096
11097     parsers.TightListItem = function(starter)
11098         return -parsers.ThematicBreak
11099             * parsers.add_indent(starter, "li")

```



```

11100             * parsers.indented_content_tight
11101     end
11102
11103     parsers.LooseListItem = function(starter)
11104         return -parsers.ThematicBreak
11105             * parsers.add_indent(starter, "li")
11106             * parsers.indented_content_loose
11107             * remove_indent("li")
11108     end
11109
11110     parsers.BulletListOfType = function(bullet_type)
11111         local bullet = parsers.bullet(bullet_type)
11112         return ( Ct( parsers.TightListItem(bullet)
11113             * ( (parsers.check_minimal_indent / "")
11114             * parsers.TightListItem(bullet)
11115             )^0
11116         )
11117             * Cc(true)
11118             * -#( (parsers.list_blank^0 / "")
11119             * parsers.check_minimal_indent
11120             * (bullet - parsers.ThematicBreak)
11121         )
11122         + Ct( parsers.LooseListItem(bullet)
11123             * ( (parsers.list_blank^0 / "")
11124             * (parsers.check_minimal_indent / "")
11125             * parsers.LooseListItem(bullet)
11126             )^0
11127         )
11128             * Cc(false)
11129         ) / writer.bulletlist
11130     end
11131
11132     parsers.BulletList = parsers.BulletListOfType(parsers.dash)
11133         + parsers.BulletListOfType(parsers.asterisk)
11134         + parsers.BulletListOfType(parsers.plus)
11135
11136     local function ordered_list(items,tight,starter)
11137         local startnum = starter[2][1]
11138         if options.startNumber then
11139             startnum = tonumber(startnum) or 1 -- fallback for '#'
11140             if startnum ~= nil then
11141                 startnum = math.floor(startnum)
11142             end
11143         else
11144             startnum = nil
11145         end
11146         return writer.orderedlist(items,tight,startnum)

```

```

11147 end
11148
11149 parsers.OrderedListOfType = function(delimiter_type)
11150     local enumerator = parsers.enumerator(delimiter_type)
11151     return Cg(enumerator, "listtype")
11152         * (Ct( parsers.TightListItem(Cb("listtype"))
11153             * ( (parsers.check_minimal_indent / "")
11154                 * parsers.TightListItem(enumerator))^0)
11155         * Cc(true)
11156         * -#((parsers.list_blank^0 / "")
11157             * parsers.check_minimal_indent * enumerator)
11158     + Ct( parsers.LooseListItem(Cb("listtype"))
11159         * ((parsers.list_blank^0 / "")
11160             * (parsers.check_minimal_indent / "")
11161             * parsers.LooseListItem(enumerator))^0)
11162         * Cc(false)
11163     ) * Ct(Cb("listtype")) / ordered_list
11164 end
11165
11166 parsers.OrderedList = parsers.OrderedListOfType(parsers.period)
11167     + parsers.OrderedListOfType(parsers.rparent)

```

### 3.1.6.11 Blank (local)

```

11168 parsers.Blank          = parsers.blankline / ""
11169                        + V("Reference")

```

### 3.1.6.12 Headings (local)

```

11170 function parsers.parse_heading_text(s)
11171     local inlines = self.parser_functions.parse_inlines(s)
11172     local flatten_inlines = self.writer.flatten_inlines
11173     self.writer.flatten_inlines = true
11174     local flat_text = self.parser_functions.parse_inlines(s)
11175     flat_text = util.rope_to_string(flat_text)
11176     self.writer.flatten_inlines = flatten_inlines
11177     return {flat_text, inlines}
11178 end
11179
11180 -- parse atx header
11181 parsers.AtxHeading = parsers.check_trail_no_rem
11182                     * Cg(parsers.heading_start, "level")
11183                     * ((C( parsers.optionalspace
11184                         * parsers.hash^0
11185                         * parsers.optionalspace
11186                         * parsers.newline)
11187                     + parsers.spacechar^1
11188                     * C(parsers.line))

```

```

11189             / strip_atx_end
11190             / parsers.parse_heading_text)
11191         * Cb("level")
11192         / writer.heading
11193
11194     parsers.heading_line = parsers.linechar^1
11195                         - parsers.thematic_break_lines
11196
11197     parsers.heading_text = parsers.heading_line
11198                         * ( (V("Endline") / "\n")
11199                         * ( parsers.heading_line
11200                           - parsers.heading_level))^0
11201                         * parsers.newline^-1
11202
11203     parsers.SettextHeading = parsers.freeze_trail
11204                         * parsers.check_trail_no_rem
11205                         * #( parsers.heading_text
11206                           * parsers.check_minimal_indent
11207                           * parsers.check_trail
11208                           * parsers.heading_level)
11209                         * Cs(parsers.heading_text)
11210                         / parsers.parse_heading_text
11211                         * parsers.check_minimal_indent_and_trail
11212                         * parsers.heading_level
11213                         * parsers.newline
11214                         * parsers.unfreeze_trail
11215                         / writer.heading
11216
11217     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  $\text{\TeX}$  output.

```

11218     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.

```

11219     local walkable_syntax = (function(global_walkable_syntax)
11220         local local_walkable_syntax = {}
11221         for lhs, rule in pairs(global_walkable_syntax) do
11222             local_walkable_syntax[lhs] = util.table_copy(rule)
11223         end

```

```

11224         return local_walkable_syntax
11225     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.

```

11226     local current_extension_name = nil
11227     self.insert_pattern = function(selector, pattern, pattern_name)
11228         assert(pattern_name == nil or type(pattern_name) == "string")
11229         local _, _, lhs, pos, rhs
11230         = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
11231         assert(lhs ~= nil,
11232             [[Expected selector in form ]]
11233             .. [[ "LHS (before|after|instead of) RHS", not "]]
11234             .. selector .. [[ "]]])
11235         assert(walkable_syntax[lhs] ~= nil,
11236             [[Rule ]] .. lhs
11237             .. [[ -> ... does not exist in markdown grammar]])
11238         assert(pos == "before" or pos == "after" or pos == "instead of",
11239             [[Expected positional specifier "before", "after", ]]
11240             .. [[or "instead of", not "]]
11241             .. pos .. [[ "]]])
11242         local rule = walkable_syntax[lhs]
11243         local index = nil
11244         for current_index, current_rhs in ipairs(rule) do
11245             if type(current_rhs) == "string" and current_rhs == rhs then
11246                 index = current_index
11247                 if pos == "after" then
11248                     index = index + 1
11249                 end
11250                 break
11251             end
11252         end
11253         assert(index ~= nil,
11254             [[Rule ]] .. lhs .. [[ -> ]] .. rhs
11255             .. [[ does not exist in markdown grammar]])
11256         local accountable_pattern
11257         if current_extension_name then
11258             accountable_pattern
11259                 = {pattern, current_extension_name, pattern_name}
11260         else
11261             assert(type(pattern) == "string",
11262                 [[reader->insert_pattern() was called outside ]]
11263                 .. [[an extension with ]]
11264                 .. [[a PEG pattern instead of a rule name]])
11265             accountable_pattern = pattern
11266         end

```

```

11267         if pos == "instead of" then
11268             rule[index] = accountable_pattern
11269         else
11270             table.insert(rule, index, accountable_pattern)
11271         end
11272     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.

```

11273     local syntax =
11274         { "Blocks",
11275
11276           Blocks = V("InitializeState")
11277                 * V("ExpectedJekyllData")
11278                 * 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.

```

11279         * ( V("Block")
11280           * ( V("Blank")^0 * parsers.eof
11281             + ( V("Blank")^2 / writer.paragraphsep
11282               + V("Blank")^0 / writer.interblocksep
11283             )
11284           )
11285         + ( V("Paragraph") + V("Plain") )
11286         * ( V("Blank")^0 * parsers.eof
11287           + ( V("Blank")^2 / writer.paragraphsep
11288             + V("Blank")^0 / writer.interblocksep
11289           )
11290         )
11291         * V("Block")
11292         * ( V("Blank")^0 * parsers.eof
11293           + ( V("Blank")^2 / writer.paragraphsep
11294             + V("Blank")^0 / writer.interblocksep
11295           )
11296         )
11297         + ( V("Paragraph") + V("Plain") )
11298         * ( V("Blank")^0 * parsers.eof
11299           + V("Blank")^0 / writer.paragraphsep
11300         )
11301       )^0,
11302
11303       ExpectedJekyllData = parsers.succeed,
11304
11305       Blank                = parsers.Blank,
11306       Reference            = parsers.Reference,
11307

```

```

11308     Blockquote      = parsers.Blockquote,
11309     Verbatim        = parsers.Verbatim,
11310     ThematicBreak   = parsers.ThematicBreak,
11311     BulletList      = parsers.BulletList,
11312     OrderedList     = parsers.OrderedList,
11313     DisplayHtml     = parsers.DisplayHtml,
11314     Heading         = parsers.Heading,
11315     Paragraph       = parsers.Paragraph,
11316     Plain           = parsers.Plain,
11317
11318     ListStarter      = parsers.ListStarter,
11319     EndlineExceptions = parsers.EndlineExceptions,
11320     NoSoftLineBreakEndlineExceptions
11321                     = parsers.NoSoftLineBreakEndlineExceptions,
11322
11323     Str              = parsers.Str,
11324     Space            = parsers.Space,
11325     NoSoftLineBreakSpace
11326                     = parsers.NoSoftLineBreakSpace,
11327     OptionalIndent   = parsers.OptionalIndent,
11328     Endline          = parsers.Endline,
11329     EndlineNoSub     = parsers.EndlineNoSub,
11330     NoSoftLineBreakEndline
11331                     = parsers.NoSoftLineBreakEndline,
11332     EndlineBreak     = parsers.EndlineBreak,
11333     LinkAndEmph      = parsers.LinkAndEmph,
11334     Code             = parsers.Code,
11335     AutoLinkUrl      = parsers.AutoLinkUrl,
11336     AutoLinkEmail    = parsers.AutoLinkEmail,
11337     AutoLinkRelativeReference
11338                     = parsers.AutoLinkRelativeReference,
11339     InlineHtml       = parsers.InlineHtml,
11340     HtmlEntity       = parsers.HtmlEntity,
11341     EscapedChar      = parsers.EscapedChar,
11342     Smart            = parsers.Smart,
11343     Symbol           = parsers.Symbol,
11344     SpecialChar      = parsers.fail,
11345     InitializeState  = parsers.succeed,
11346 }

```

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]`.

```

11347     self.update_rule = function(rule_name, get_pattern)
11348         assert(current_extension_name ~= nil)

```

```

11349     assert(syntax[rule_name] ~= nil,
11350           [[Rule ]] .. rule_name
11351           .. [[ -> ... does not exist in markdown grammar]])
11352     local previous_pattern
11353     local extension_name
11354     if walkable_syntax[rule_name] then
11355         local previous_accountable_pattern
11356         = walkable_syntax[rule_name][1]
11357         previous_pattern = previous_accountable_pattern[1]
11358         extension_name
11359         = previous_accountable_pattern[2]
11360         .. ", " .. current_extension_name
11361     else
11362         previous_pattern = nil
11363         extension_name = current_extension_name
11364     end
11365     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

```

```

11366     if type(get_pattern) == "function" then
11367         pattern = get_pattern(previous_pattern)
11368     else
11369         assert(previous_pattern == nil,
11370               [[Rule ]] .. rule_name ..
11371               [[ has already been updated by ]] .. extension_name)
11372         pattern = get_pattern
11373     end
11374     local accountable_pattern = { pattern, extension_name, rule_name }
11375     walkable_syntax[rule_name] = { accountable_pattern }
11376 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.

```

11377     local special_characters = {}
11378     self.add_special_character = function(c)
11379         table.insert(special_characters, c)
11380         syntax.SpecialChar = S(table.concat(special_characters, ""))

```

```

11381     end
11382
11383     self.add_special_character("*")
11384     self.add_special_character("[")
11385     self.add_special_character("]")
11386     self.add_special_character("<")
11387     self.add_special_character("!")
11388     self.add_special_character("\\")

```

Add method `reader->initialize_named_group` that defines named groups with a default capture value.

```

11389     self.initialize_named_group = function(name, value)
11390         local pattern = Ct("")
11391         if value ~= nil then
11392             pattern = pattern / value
11393         end
11394         syntax.InitializeState = syntax.InitializeState
11395                                 * Cg(pattern, name)
11396     end

```

Add a named group for indentation.

```

11397     self.initialize_named_group("indent_info")

```

Apply syntax extensions.

```

11398     for _, extension in ipairs(extensions) do
11399         current_extension_name = extension.name
11400         extension.extend_writer(writer)
11401         extension.extend_reader(self)
11402     end
11403     current_extension_name = nil

```

If the `debugExtensions` option is enabled, serialize `walkable_syntax` to a JSON for debugging purposes.

```

11404     if options.debugExtensions then
11405         local sorted_lhs = {}
11406         for lhs, _ in pairs(walkable_syntax) do
11407             table.insert(sorted_lhs, lhs)
11408         end
11409         table.sort(sorted_lhs)
11410
11411         local output_lines = {"{"}
11412         for lhs_index, lhs in ipairs(sorted_lhs) do
11413             local encoded_lhs = util.encode_json_string(lhs)
11414             table.insert(output_lines, [[    ]] .. encoded_lhs .. [[:  ]])
11415             local rule = walkable_syntax[lhs]
11416             for rhs_index, rhs in ipairs(rule) do
11417                 local human_readable_rhs
11418                 if type(rhs) == "string" then

```



```

11419         human_readable_rhs = rhs
11420     else
11421         local pattern_name
11422         if rhs[3] then
11423             pattern_name = rhs[3]
11424         else
11425             pattern_name = "Anonymous Pattern"
11426         end
11427         local extension_name = rhs[2]
11428         human_readable_rhs = pattern_name .. [[ (]]
11429                               .. extension_name .. [[]]]
11430     end
11431     local encoded_rhs
11432     = util.encode_json_string(human_readable_rhs)
11433     local output_line = [[          ]] .. encoded_rhs
11434     if rhs_index < #rule then
11435         output_line = output_line .. ", "
11436     end
11437     table.insert(output_lines, output_line)
11438 end
11439 local output_line = "    ]"
11440 if lhs_index < #sorted_lhs then
11441     output_line = output_line .. ", "
11442 end
11443 table.insert(output_lines, output_line)
11444 end
11445 table.insert(output_lines, "}")
11446
11447 local output = table.concat(output_lines, "\n")
11448 local output_filename = options.debugExtensionsFileName
11449 local output_file = assert(io.open(output_filename, "w"),
11450     [[Could not open file ]] .. output_filename
11451     .. [[ for writing]])
11452 assert(output_file:write(output))
11453 assert(output_file:close())
11454 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](#).

```

11455     for lhs, rule in pairs(walkable_syntax) do
11456         syntax[lhs] = parsers.fail
11457         for _, rhs in ipairs(rule) do
11458             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.

```
11459         if type(rhs) == "string" then
11460             pattern = V(rhs)
11461         else
11462             pattern = rhs[1]
11463             if type(pattern) == "string" then
11464                 pattern = V(pattern)
11465             end
11466         end
11467         syntax[lhs] = syntax[lhs] + pattern
11468     end
11469 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.

```
11470     if options.underscores then
11471         self.add_special_character("_")
11472     end
11473
11474     if not options.codeSpans then
11475         syntax.Code = parsers.fail
11476     else
11477         self.add_special_character("`")
11478     end
11479
11480     if not options.html then
11481         syntax.DisplayHtml = parsers.fail
11482         syntax.InlineHtml = parsers.fail
11483         syntax.HtmlEntity = parsers.fail
11484     else
11485         self.add_special_character("&")
11486     end
11487
11488     if options.preserveTabs then
11489         options.stripIndent = false
11490     end
11491
11492     if not options.smartEllipses then
11493         syntax.Smart = parsers.fail
11494     else
11495         self.add_special_character(".")
11496     end
11497
11498     if not options.relativeReferences then
11499         syntax.AutoLinkRelativeReference = parsers.fail
```

```

11500     end
11501
11502     if options.contentLevel == "inline" then
11503         syntax[1] = "Inlines"
11504         syntax.Inlines = V("InitializeState")
11505             * parsers.Inline^0
11506             * ( parsers.spacing^0
11507             * parsers.eof / "" )
11508         syntax.Space = parsers.Space + parsers.blankline / writer.space
11509     end
11510
11511     local blocks_nested_t = util.table_copy(syntax)
11512     blocks_nested_t.ExpectedJekyllData = parsers.succeed
11513     parsers.blocks_nested = Ct(blocks_nested_t)
11514
11515     parsers.blocks = Ct(syntax)
11516
11517     local inlines_t = util.table_copy(syntax)
11518     inlines_t[1] = "Inlines"
11519     inlines_t.Inlines = V("InitializeState")
11520         * parsers.Inline^0
11521         * ( parsers.spacing^0
11522         * parsers.eof / "" )
11523     parsers.inlines = Ct(inlines_t)
11524
11525     local inlines_no_inline_note_t = util.table_copy(inlines_t)
11526     inlines_no_inline_note_t.InlineNote = parsers.fail
11527     parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
11528
11529     local inlines_no_html_t = util.table_copy(inlines_t)
11530     inlines_no_html_t.DisplayHtml = parsers.fail
11531     inlines_no_html_t.InlineHtml = parsers.fail
11532     inlines_no_html_t.HtmlEntity = parsers.fail
11533     parsers.inlines_no_html = Ct(inlines_no_html_t)
11534
11535     local inlines_nbsp_t = util.table_copy(inlines_t)
11536     inlines_nbsp_t.Endline = parsers.NonbreakingEndline
11537     inlines_nbsp_t.Space = parsers.NonbreakingSpace
11538     parsers.inlines_nbsp = Ct(inlines_nbsp_t)
11539
11540     local inlines_no_link_or_emphasis_t = util.table_copy(inlines_t)
11541     inlines_no_link_or_emphasis_t.LinkAndEmph = parsers.fail
11542     inlines_no_link_or_emphasis_t.EndlineExceptions
11543         = parsers.EndlineExceptions - parsers.eof
11544     parsers.inlines_no_link_or_emphasis
11545         = Ct(inlines_no_link_or_emphasis_t)

```

Return a function that converts markdown string `input` into a plain TeX output and returns it..

```
11546     return function(input)
```

Unicode-normalize the input.

```
11547         if options.unicodeNormalization then
11548             local form = options.unicodeNormalizationForm
11549             input = util.normalize(input, form)
11550         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.

```
11551         input = input:gsub("\r\n?", "\n")
11552         if input:sub(-1) ~= "\n" then
11553             input = input .. "\n"
11554         end
```

Clear the table of references.

```
11555         references = {}
11556         local document = self.parser_functions.parse_blocks(input)
11557         local output = util.ropetostring(writer.document(document))
```

Remove block element / paragraph separators immediately followed by the output of `writer->undosep`, possibly interleaved by section ends. Then, remove any leftover output of `writer->undosep`.

```
11558         local undosep_start, undosep_end
11559         local potential_secend_start, secend_start
11560         local potential_sep_start, sep_start
11561         while true do
11562             -- find a `writer->undosep`
11563             undosep_start, undosep_end
11564             = output:find(writer.undosep_text, 1, true)
11565             if undosep_start == nil then break end
11566             -- skip any preceding section ends
11567             secend_start = undosep_start
11568             while true do
11569                 potential_secend_start = secend_start - #writer.secend_text
11570                 if potential_secend_start < 1
11571                     or output:sub(potential_secend_start,
11572                                   secend_start - 1) ~= writer.secend_text
11573                 then
11574                     break
11575                 end
11576                 secend_start = potential_secend_start
11577             end
11578             -- find an immediately preceding
11579             -- block element / paragraph separator
11580             sep_start = secend_start
```

```

11581     potential_sep_start = sep_start - #writer.interblocksep_text
11582     if potential_sep_start >= 1
11583         and output:sub(potential_sep_start,
11584             sep_start - 1) == writer.interblocksep_text
11585         then
11586             sep_start = potential_sep_start
11587     else
11588         potential_sep_start = sep_start - #writer.paragraphsep_text
11589         if potential_sep_start >= 1
11590             and output:sub(potential_sep_start,
11591                 sep_start - 1) == writer.paragraphsep_text
11592             then
11593                 sep_start = potential_sep_start
11594             end
11595         end
11596         -- remove `writer->undosep` and immediately preceding
11597         -- block element / paragraph separator
11598         output = output:sub(1, sep_start - 1)
11599             .. output:sub(secend_start, undosep_start - 1)
11600             .. output:sub(undosep_end + 1)
11601     end
11602     return output
11603 end
11604 end
11605 return self
11606 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.

```

11607 M.extensions = {}

```

#### 3.1.7.1 Bracketed Spans

The `extensions.bracketed_spans` function implements the Pandoc bracketed span syntax extension.

```

11608 M.extensions.bracketed_spans = function()
11609     return {
11610         name = "built-in bracketed_spans syntax extension",
11611         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.

```

11612     function self.span(s, attr)
11613         if self.flatten_inlines then return s end
11614         return {"\\markdownRendererBracketedSpanAttributeContextBegin",
11615             self.attributes(attr),
11616             s,
11617             "\\markdownRendererBracketedSpanAttributeContextEnd{}}"}
11618     end
11619 end, extend_reader = function(self)
11620     local parsers = self.parsers
11621     local writer = self.writer
11622
11623     local span_label = parsers.lbracket
11624                     * (Cs((parsers.alphanumeric^1
11625                         + parsers.inticks
11626                         + parsers.autolink
11627                         + V("InlineHtml")
11628                         + ( parsers.backslash * parsers.backslash)
11629                         + ( parsers.backslash
11630                           * (parsers.lbracket + parsers.rbracket)
11631                           + V("Space") + V("Endline")
11632                           + (parsers.any
11633                             - ( parsers.newline
11634                               + parsers.lbracket
11635                               + parsers.rbracket
11636                               + parsers.blankline^2))))^1)
11637                     / self.parser_functions.parse_inlines)
11638                     * parsers.rbracket
11639
11640     local Span = span_label
11641                 * Ct(parsers.attributes)
11642                 / writer.span
11643
11644     self.insert_pattern("Inline before LinkAndEmph",
11645         Span, "Span")
11646 end
11647 }
11648 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.

```

11649 M.extensions.citations = function(citation_nbsps)
11650     return {
11651         name = "built-in citations syntax extension",

```

```
11652     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.

```
11653     function self.citations(text_cites, cites)
11654         local buffer = {}
11655         if self.flatten_inlines then
11656             for _,cite in ipairs(cites) do
11657                 if cite.prenote then
11658                     table.insert(buffer, {cite.prenote, " "})
11659                 end
11660                 table.insert(buffer, cite.name)
11661                 if cite.postnote then
11662                     table.insert(buffer, {" ", cite.postnote})
11663                 end
11664             end
11665         else
11666             table.insert(buffer,
11667                 {"\\markdownRenderer",
11668                     text_cites and "TextCite" or "Cite",
11669                     "{", #cites, "}"})
11670             for _,cite in ipairs(cites) do
11671                 table.insert(buffer,
11672                     {cite.suppress_author and "-" or "+", "{",
11673                     cite.prenote or "", "}{" ,
11674                     cite.postnote or "", "}{" , cite.name, "}"})
11675             end
11676         end
11677         return buffer
11678     end
11679 end, extend_reader = function(self)
11680     local parsers = self.parsers
11681     local writer = self.writer
11682
```

```

11683     local citation_chars
11684         = parsers.alphanumeric
11685         + S("#$%&-+<>~/_")
11686
11687     local citation_name
11688         = Cs(parsers.dash~-1) * parsers.at
11689         * Cs(citation_chars
11690             * ((( citation_chars
11691                 + parsers.internal_punctuation
11692                 - parsers.comma - parsers.semicolon)
11693             * -#(( parsers.internal_punctuation
11694                 - parsers.comma
11695                 - parsers.semicolon)^0
11696             * -( citation_chars
11697                 + parsers.internal_punctuation
11698                 - parsers.comma
11699                 - parsers.semicolon)))^0
11700             * citation_chars)^-1)
11701
11702     local citation_body_prenote
11703         = Cs((parsers.alphanumeric~1
11704             + parsers.bracketed
11705             + parsers.inticks
11706             + parsers.autolink
11707             + V("InlineHtml")
11708             + V("Space") + V("EndlineNoSub")
11709             + (parsers.anyescaped
11710                 - ( parsers.newline
11711                     + parsers.rbracket
11712                     + parsers.blankline^2))
11713             - ( parsers.spnl
11714                 * parsers.dash~-1
11715                 * parsers.at))^1)
11716
11717     local citation_body_postnote
11718         = Cs((parsers.alphanumeric~1
11719             + parsers.bracketed
11720             + parsers.inticks
11721             + parsers.autolink
11722             + V("InlineHtml")
11723             + V("Space") + V("EndlineNoSub")
11724             + (parsers.anyescaped
11725                 - ( parsers.newline
11726                     + parsers.rbracket
11727                     + parsers.semicolon
11728                     + parsers.blankline^2))
11729             - (parsers.spnl * parsers.rbracket))^1)

```



```

11730
11731     local citation_body_chunk
11732         = ( citation_body_prenote
11733             * parsers.spnlc_sep
11734             + Cc("")
11735             * parsers.spnlc
11736         )
11737         * citation_name
11738         * ( parsers.internal_punctuation
11739           - parsers.semicolon)^-1
11740         * ( parsers.spnlc / function(_) return end
11741           * citation_body_postnote
11742           + Cc("")
11743           * parsers.spnlc
11744         )
11745
11746     local citation_body
11747         = citation_body_chunk
11748         * ( parsers.semicolon
11749           * parsers.spnlc
11750           * citation_body_chunk
11751         )^0
11752
11753     local citation_headless_body_postnote
11754         = Cs((parsers.alphanumeric^1
11755             + parsers.bracketed
11756             + parsers.inticks
11757             + parsers.autolink
11758             + V("InlineHtml")
11759             + V("Space") + V("Endline")
11760             + (parsers.anyescaped
11761               - ( parsers.newline
11762                 + parsers.rbracket
11763                 + parsers.at
11764                 + parsers.semicolon + parsers.blankline^2))
11765             - (parsers.spnl * parsers.rbracket))^0)
11766
11767     local citation_headless_body
11768         = citation_headless_body_postnote
11769         * ( parsers.semicolon
11770           * parsers.spnlc
11771           * citation_body_chunk
11772         )^0
11773
11774     local citations
11775         = function(text_cites, raw_cites)
11776             local function normalize(str)

```

```

11777         if str == "" then
11778             str = nil
11779         else
11780             str = (citation_nbsps and
11781                 self.parser_functions.parse_inlines_nbsp or
11782                 self.parser_functions.parse_inlines)(str)
11783         end
11784         return str
11785     end
11786
11787     local cites = {}
11788     for i = 1,#raw_cites,4 do
11789         cites[#cites+1] = {
11790             prenote = normalize(raw_cites[i]),
11791             suppress_author = raw_cites[i+1] == "-",
11792             name = writer.identifier(raw_cites[i+2]),
11793             postnote = normalize(raw_cites[i+3]),
11794         }
11795     end
11796     return writer.citations(text_cites, cites)
11797 end
11798
11799 local TextCitations
11800     = Ct((parsers.spnlc
11801         * Cc("")
11802         * citation_name
11803         * ((parsers.spnlc
11804             * parsers.lbracket
11805             * citation_headless_body
11806             * parsers.rbracket) + Cc("")))~1)
11807 / function(raw_cites)
11808     return citations(true, raw_cites)
11809 end
11810
11811 local ParenthesizedCitations
11812     = Ct((parsers.spnlc
11813         * parsers.lbracket
11814         * citation_body
11815         * parsers.rbracket)^1)
11816 / function(raw_cites)
11817     return citations(false, raw_cites)
11818 end
11819
11820 local Citations = TextCitations + ParenthesizedCitations
11821
11822 self.insert_pattern("Inline before LinkAndEmph",
11823     Citations, "Citations")

```

```

11824
11825     self.add_special_character("@")
11826     self.add_special_character("-")
11827 end
11828 }
11829 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.

```

11830 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.

```

11831 local languages_json = (function()
11832     local base, prev, curr
11833     for _, pathname in ipairs(util.find_files(language_map)) do
11834         local file = io.open(pathname, "r")
11835         if not file then goto continue end
11836         local input = assert(file:read("*a"))
11837         assert(file:close())
11838         local json = input:gsub('[^\n]-'):','[%1]='')
11839         curr = load("_ENV = {}; return "..json)()
11840         if type(curr) == "table" then
11841             if base == nil then
11842                 base = curr
11843             else
11844                 setmetatable(prev, { __index = curr })
11845             end
11846             prev = curr
11847         end
11848         ::continue::
11849     end
11850     return base or {}
11851 end)()
11852
11853 return {
11854     name = "built-in content_blocks syntax extension",
11855     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.

```

11856     function self.contentblock(src,suf,type,tit)
11857         if not self.is_writing then return "" end
11858         src = src..".."..suf
11859         suf = suf:lower()
11860         if type == "onlineimage" then
11861             return {"\\markdownRendererContentBlockOnlineImage{"..suf.."},"",
11862                 {"",self.string(src),"},"",
11863                 {"",self.uri(src),"},"",
11864                 {"",self.string(tit or ""),"},""}
11865         elseif languages_json[suf] then
11866             return {"\\markdownRendererContentBlockCode{"..suf.."},"",
11867                 {"",self.string(languages_json[suf]),"},"",
11868                 {"",self.string(src),"},"",
11869                 {"",self.uri(src),"},"",
11870                 {"",self.string(tit or ""),"},""}
11871         else
11872             return {"\\markdownRendererContentBlock{"..suf.."},"",
11873                 {"",self.string(src),"},"",
11874                 {"",self.uri(src),"},"",
11875                 {"",self.string(tit or ""),"},""}
11876         end
11877     end
11878 end, extend_reader = function(self)
11879     local parsers = self.parsers
11880     local writer = self.writer
11881
11882     local contentblock_tail
11883         = parsers.optionaltitle
11884         * (parsers.newline + parsers.eof)
11885
11886     -- case insensitive online image suffix:
11887     local onlineimagesuffix
11888         = (function(...)
11889             local parser = nil
11890             for _, suffix in ipairs({...}) do
11891                 local pattern=nil
11892                 for i=1,#suffix do
11893                     local char=suffix:sub(i,i)
11894                     char = S(char:lower()..char:upper())
11895                     if pattern == nil then
11896                         pattern = char
11897                     else
11898                         pattern = pattern * char
11899                     end
11900                 end

```

```

11901         if parser == nil then
11902             parser = pattern
11903         else
11904             parser = parser + pattern
11905         end
11906     end
11907     return parser
11908 end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
11909
11910 -- online image url for iA Writer content blocks with
11911 -- mandatory suffix, allowing nested brackets:
11912 local onlineimageurl
11913     = (parsers.less
11914         * Cs((parsers.anyescaped
11915             - parsers.more
11916             - parsers.spacing
11917             - #(parsers.period
11918                 * onlineimagesuffix
11919                 * parsers.more
11920                 * contentblock_tail))^0)
11921         * parsers.period
11922         * Cs(onlineimagesuffix)
11923         * parsers.more
11924         + (Cs((parsers.inparens
11925             + (parsers.anyescaped
11926                 - parsers.spacing
11927                 - parsers.rparent
11928                 - #(parsers.period
11929                     * onlineimagesuffix
11930                     * contentblock_tail))))^0)
11931         * parsers.period
11932         * Cs(onlineimagesuffix))
11933     ) * Cc("onlineimage")
11934
11935 -- filename for iA Writer content blocks with mandatory suffix:
11936 local localfilepath
11937     = parsers.slash
11938     * Cs((parsers.anyescaped
11939         - parsers.tab
11940         - parsers.newline
11941         - #(parsers.period
11942             * parsers.alphanumeric^1
11943             * contentblock_tail))^1)
11944     * parsers.period
11945     * Cs(parsers.alphanumeric^1)
11946     * Cc("localfile")
11947

```

```

11948     local ContentBlock
11949         = parsers.check_trail_no_rem
11950         * (localfilepath + onlineimageurl)
11951         * contentblock_tail
11952         / writer.contentblock
11953
11954     self.insert_pattern("Block before Blockquote",
11955                       ContentBlock, "ContentBlock")
11956 end
11957 }
11958 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.

```

11959 M.extensions.definition_lists = function(tight_lists)
11960   return {
11961     name = "built-in definition_lists syntax extension",
11962     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.

```

11963     local function dlistem(term, defs)
11964       local retVal = {"\\markdownRendererDlItem{",term,""}
11965       for _, def in ipairs(defs) do
11966         retVal[#retVal+1]
11967           = {"\\markdownRendererDlDefinitionBegin ",def,
11968             "\\markdownRendererDlDefinitionEnd "}
11969       end
11970       retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
11971       return retVal
11972     end
11973
11974     function self.definitionlist(items,tight)
11975       if not self.is_writing then return "" end
11976       local buffer = {}
11977       for _,item in ipairs(items) do
11978         buffer[#buffer + 1] = dlistem(item.term, item.definitions)
11979       end
11980       if tight and tight_lists then
11981         return {"\\markdownRendererDlBeginTight\\n", buffer,
11982               "\\n\\markdownRendererDlEndTight"}
11983       else

```

```

11984         return {"\\markdownRendererDlBegin\\n", buffer,
11985                 "\\n\\markdownRendererDlEnd"}
11986     end
11987 end
11988 end, extend_reader = function(self)
11989     local parsers = self.parsers
11990     local writer = self.writer
11991
11992     local defstartchar = S("~:")
11993
11994     local defstart
11995         = parsers.check_trail_length(0) * defstartchar
11996         * #parsers.spacing
11997         * (parsers.tab + parsers.space~-3)
11998         + parsers.check_trail_length(1)
11999         * defstartchar * #parsers.spacing
12000         * (parsers.tab + parsers.space~-2)
12001         + parsers.check_trail_length(2)
12002         * defstartchar * #parsers.spacing
12003         * (parsers.tab + parsers.space~-1)
12004         + parsers.check_trail_length(3)
12005         * defstartchar * #parsers.spacing
12006
12007     local indented_line
12008         = (parsers.check_minimal_indent / "")
12009         * parsers.check_code_trail * parsers.line
12010
12011     local blank
12012         = parsers.check_minimal_blank_indent_and_any_trail
12013         * parsers.optionalspace * parsers.newline
12014
12015     local dlchunk = Cs(parsers.line * (indented_line - blank)^0)
12016
12017     local indented_blocks = function(bl)
12018         return Cs( bl
12019             * (blank^1 * (parsers.check_minimal_indent / "")
12020             * parsers.check_code_trail * -parsers.blankline * bl)^0
12021             * (blank^1 + parsers.eof))
12022     end
12023
12024     local function definition_list_item(term, defs, _)
12025         return { term = self.parser_functions.parse_inlines(term),
12026                 definitions = defs }
12027     end
12028
12029     local DefinitionListItemLoose
12030         = C(parsers.line) * blank^0

```

```

12031      * Ct((parsers.check_minimal_indent * (defstart
12032          * indented_blocks(dlchunk)
12033          / self.parser_functions.parse_blocks_nested))^1)
12034      * Cc(false) / definition_list_item
12035
12036      local DefinitionListItemTight
12037      = C(parsers.line)
12038      * Ct((parsers.check_minimal_indent * (defstart * dlchunk
12039          / self.parser_functions.parse_blocks_nested))^1)
12040      * Cc(true) / definition_list_item
12041
12042      local DefinitionList
12043      = ( Ct(DefinitionListItemLoose^1) * Cc(false)
12044        + Ct(DefinitionListItemTight^1)
12045        * (blank^0
12046          * -DefinitionListItemLoose * Cc(true))
12047        ) / writer.definitionlist
12048
12049      self.insert_pattern("Block after Heading",
12050                          DefinitionList, "DefinitionList")
12051    end
12052  }
12053 end

```

### 3.1.7.5 Fancy Lists

The `extensions.fancy_lists` function implements the Pandoc fancy list syntax extension.

```

12054 M.extensions.fancy_lists = function()
12055   return {
12056     name = "built-in fancy_lists syntax extension",
12057     extend_writer = function(self)
12058       local options = self.options
12059

```

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

- `items` is an array of the list items,
- `tight` specifies, whether the list is tight or not,
- `startnum` is the number of the first list item,
- `numstyle` is the style of the list item labels from among the following:
  - `Decimal` – decimal arabic numbers,
  - `LowerRoman` – lower roman numbers,
  - `UpperRoman` – upper roman numbers,
  - `LowerAlpha` – lower ASCII alphabetic characters, and



- **UpperAlpha** – upper ASCII alphabetic characters, and
- **numdelim** is the style of delimiters between list item labels and texts from among the following:
  - **Default** – default style,
  - **OneParen** – parentheses, and
  - **Period** – periods.

```

12060     function self.fancylis(items,tight,startnum,numstyle,numdelim)
12061         if not self.is_writing then return "" end
12062         local buffer = {}
12063         local num = startnum
12064         for _,item in ipairs(items) do
12065             if item ~= "" then
12066                 buffer[#buffer + 1] = self.fancyitem(item,num)
12067             end
12068             if num ~= nil and item ~= "" then
12069                 num = num + 1
12070             end
12071         end
12072         local contents = util.intersperse(buffer,"\n")
12073         if tight and options.tightLists then
12074             return {"\\markdownRendererFancyOlBeginTight{",
12075                 numstyle,"}{",numdelim,"}",contents,
12076                 "\\n\\markdownRendererFancyOlEndTight "}
12077         else
12078             return {"\\markdownRendererFancyOlBegin{",
12079                 numstyle,"}{",numdelim,"}",contents,
12080                 "\\n\\markdownRendererFancyOlEnd "}
12081         end
12082     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.

```

12083     function self.fancyitem(s,num)
12084         if num ~= nil then
12085             return {"\\markdownRendererFancyOlItemWithNumber{",num,"}",s,
12086                 "\\n\\markdownRendererFancyOlItemEnd "}
12087         else
12088             return {"\\markdownRendererFancyOlItem ",s,
12089                 "\\n\\markdownRendererFancyOlItemEnd "}
12090         end
12091     end
12092 end, extend_reader = function(self)
12093     local parsers = self.parsers

```

```

12094     local options = self.options
12095     local writer = self.writer
12096
12097     local function combine_markers_and_delims(markers, delims)
12098         local markers_table = {}
12099         for _,marker in ipairs(markers) do
12100             local start_marker
12101             local continuation_marker
12102             if type(marker) == "table" then
12103                 start_marker = marker[1]
12104                 continuation_marker = marker[2]
12105             else
12106                 start_marker = marker
12107                 continuation_marker = marker
12108             end
12109             for _,delim in ipairs(delims) do
12110                 table.insert(markers_table,
12111                     {start_marker, continuation_marker, delim})
12112             end
12113         end
12114         return markers_table
12115     end
12116
12117     local function join_table_with_func(func, markers_table)
12118         local pattern = func(table.unpack(markers_table[1]))
12119         for i = 2, #markers_table do
12120             pattern = pattern + func(table.unpack(markers_table[i]))
12121         end
12122         return pattern
12123     end
12124
12125     local lowercase_letter_marker = R("az")
12126     local uppercase_letter_marker = R("AZ")
12127
12128     local roman_marker = function(chars)
12129         local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])
12130         local l, x, v, i
12131         = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])
12132         return  m^-3
12133             * (c*m + c*d + d^-1 * c^-3)
12134             * (x*c + x*l + l^-1 * x^-3)
12135             * (i*x + i*v + v^-1 * i^-3)
12136     end
12137
12138     local lowercase_roman_marker
12139     = roman_marker({"m", "d", "c", "l", "x", "v", "i"})
12140     local uppercase_roman_marker

```

```

12141     = roman_marker({"M", "D", "C", "L", "X", "V", "I"})
12142
12143     local lowercase_opening_roman_marker = P("i")
12144     local uppercase_opening_roman_marker = P("I")
12145
12146     local digit_marker = parsers.dig * parsers.dig~-8
12147
12148     local markers = {
12149         {lowercase_opening_roman_marker, lowercase_roman_marker},
12150         {uppercase_opening_roman_marker, uppercase_roman_marker},
12151         lowercase_letter_marker,
12152         uppercase_letter_marker,
12153         lowercase_roman_marker,
12154         uppercase_roman_marker,
12155         digit_marker
12156     }
12157
12158     local delims = {
12159         parsers.period,
12160         parsers.rparent
12161     }
12162
12163     local markers_table = combine_markers_and_delims(markers, delims)
12164
12165     local function enumerator(start_marker, _,
12166                               delimiter_type, interrupting)
12167         local delimiter_range
12168         local allowed_end
12169         if interrupting then
12170             delimiter_range = P("1")
12171             allowed_end = C(parsers.spacechar~1) * #parsers.linechar
12172         else
12173             delimiter_range = start_marker
12174             allowed_end = C(parsers.spacechar~1)
12175                         + #(parsers.newline + parsers.eof)
12176         end
12177
12178         return parsers.check_trail
12179             * Ct(C(delimiter_range) * C(delimiter_type))
12180             * allowed_end
12181     end
12182
12183     local starter = join_table_with_func(enumerator, markers_table)
12184
12185     local TightListItem = function(starter)
12186         return parsers.add_indent(starter, "li")
12187             * parsers.indented_content_tight

```

```

12188     end
12189
12190     local LooseListItem = function(starter)
12191         return parsers.add_indent(starter, "li")
12192             * parsers.indented_content_loose
12193             * remove_indent("li")
12194     end
12195
12196     local function roman2number(roman)
12197         local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100,
12198             ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
12199         local numeral = 0
12200
12201         local i = 1
12202         local len = string.len(roman)
12203         while i < len do
12204             local z1, z2 = romans[ string.sub(roman, i, i) ],
12205                 romans[ string.sub(roman, i+1, i+1) ]
12206             if z1 < z2 then
12207                 numeral = numeral + (z2 - z1)
12208                 i = i + 2
12209             else
12210                 numeral = numeral + z1
12211                 i = i + 1
12212             end
12213         end
12214         if i <= len then
12215             numeral = numeral + romans[ string.sub(roman,i,i) ]
12216         end
12217         return numeral
12218     end
12219
12220     local function sniffstyle(numstr, delimend)
12221         local numdelim
12222         if delimend == ")" then
12223             numdelim = "OneParen"
12224         elseif delimend == "." then
12225             numdelim = "Period"
12226         else
12227             numdelim = "Default"
12228         end
12229
12230         local num
12231         num = numstr:match("^([I])$")
12232         if num then
12233             return roman2number(num), "UpperRoman", numdelim
12234         end

```

```

12235     num = numstr:match("^([i])$")
12236     if num then
12237         return roman2number(string.upper(num)), "LowerRoman", numdelim
12238     end
12239     num = numstr:match("^([A-Z])$")
12240     if num then
12241         return string.byte(num) - string.byte("A") + 1,
12242             "UpperAlpha", numdelim
12243     end
12244     num = numstr:match("^([a-z])$")
12245     if num then
12246         return string.byte(num) - string.byte("a") + 1,
12247             "LowerAlpha", numdelim
12248     end
12249     num = numstr:match("^([IVXLCDM]+)")
12250     if num then
12251         return roman2number(num), "UpperRoman", numdelim
12252     end
12253     num = numstr:match("^([ivxlcdm]+)")
12254     if num then
12255         return roman2number(string.upper(num)), "LowerRoman", numdelim
12256     end
12257     return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
12258 end
12259
12260 local function fancylist(items,tight,start)
12261     local startnum, numstyle, numdelim
12262     = sniffstyle(start[2][1], start[2][2])
12263     return writer.fancylist(items,tight,
12264                             options.startNumber and startnum or 1,
12265                             numstyle or "Decimal",
12266                             numdelim or "Default")
12267 end
12268
12269 local FancyListOfType
12270 = function(start_marker, continuation_marker, delimiter_type)
12271     local enumerator_start
12272     = enumerator(start_marker, continuation_marker,
12273                 delimiter_type)
12274     local enumerator_cont
12275     = enumerator(continuation_marker, continuation_marker,
12276                 delimiter_type)
12277     return Cg(enumerator_start, "listtype")
12278         * (Ct( TightListItem(Cb("listtype"))
12279             * ((parsers.check_minimal_indent / "")
12280             * TightListItem(enumerator_cont))^0)
12281         * Cc(true)

```

```

12282         * -#((parsers.conditionally_indented_blankline^0 / "")
12283         * parsers.check_minimal_indent * enumerator_cont)
12284     + Ct( LooseListItem(Cb("listtype")))
12285         * ((parsers.conditionally_indented_blankline^0 / "")
12286         * (parsers.check_minimal_indent / ""))
12287         * LooseListItem(enumerator_cont))^0)
12288     * Cc(false)
12289     ) * Ct(Cb("listtype")) / fancylist
12290 end
12291
12292 local FancyList
12293   = join_table_with_func(FancyListOfType, markers_table)
12294
12295 local ListStarter = starter
12296
12297 self.update_rule("OrderedList", FancyList)
12298 self.update_rule("ListStarter", ListStarter)
12299 end
12300 }
12301 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.

```

12302 M.extensions.fenced_code = function(blank_before_code_fence,
12303                                     allow_attributes,
12304                                     allow_raw_blocks)
12305   return {
12306     name = "built-in fenced_code syntax extension",
12307     extend_writer = function(self)
12308       local options = self.options
12309

```

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.

```

12310   function self.fencedCode(s, i, attr)
12311     if not self.is_writing then return "" end
12312     s = s:gsub("\n$", "")
12313     local buf = {}
12314     if attr ~= nil then

```

```

12315         table.insert(buf,
12316             {"\\markdownRendererFencedCodeAttributeContextBegin",
12317                 self.attributes(attr)}})
12318     end
12319     local name = util.cache_verbatim(options.cacheDir, s)
12320     table.insert(buf,
12321         {"\\markdownRendererInputFencedCode{",
12322             name,"}{",self.string(i),"}{",self.infostring(i),"}}")
12323     if attr ~= nil then
12324         table.insert(buf,
12325             "\\markdownRendererFencedCodeAttributeContextEnd{")
12326     end
12327     return buf
12328 end
12329

```

Define `writer->rawBlock` as a function that will transform an input raw block `s` with the raw attribute `attr` to the output format.

```

12330     if allow_raw_blocks then
12331         function self.rawBlock(s, attr)
12332             if not self.is_writing then return "" end
12333             s = s:gsub("\\n$", "")
12334             local name = util.cache_verbatim(options.cacheDir, s)
12335             return {"\\markdownRendererInputRawBlock{",
12336                 name,"}{", self.string(attr),"}}")
12337         end
12338     end
12339 end, extend_reader = function(self)
12340     local parsers = self.parsers
12341     local writer = self.writer
12342
12343     local function captures_geq_length(_,i,a,b)
12344         return #a >= #b and i
12345     end
12346
12347     local function strip_enclosing_whitespaces(str)
12348         return str:gsub("^%s*(.)%s*$", "%1")
12349     end
12350
12351     local tilde_infostring = Cs(Cs((V("HtmlEntity")
12352         + parsers.anyescaped
12353         - parsers.newline)^0)
12354         / strip_enclosing_whitespaces)
12355
12356     local backtick_infostring
12357         = Cs( Cs((V("HtmlEntity")
12358             + ( -#(parsers.backslash * parsers.backtick)

```

```

12359         * parsers.anyescaped)
12360         - parsers.newline
12361         - parsers.backtick)^0)
12362     / strip_enclosing_whitespaces)
12363
12364     local fenceindent
12365
12366     local function has_trail(indent_table)
12367         return indent_table ~= nil and
12368             indent_table.trail ~= nil and
12369             next(indent_table.trail) ~= nil
12370     end
12371
12372     local function has_indents(indent_table)
12373         return indent_table ~= nil and
12374             indent_table.indents ~= nil and
12375             next(indent_table.indents) ~= nil
12376     end
12377
12378     local function get_last_indent_name(indent_table)
12379         if has_indents(indent_table) then
12380             return indent_table.indents[#indent_table.indents].name
12381         end
12382     end
12383
12384     local count_fenced_start_indent =
12385     function(_, _, indent_table, trail)
12386         local last_indent_name = get_last_indent_name(indent_table)
12387         fenceindent = 0
12388         if last_indent_name ~= "li" then
12389             fenceindent = #trail
12390         end
12391         return true
12392     end
12393
12394     local fencehead = function(char, infostring)
12395         return Cmt( Cb("indent_info")
12396             * parsers.check_trail, count_fenced_start_indent)
12397             * Cg(char^3, "fencelength")
12398             * parsers.optionalspace
12399             * infostring
12400             * (parsers.newline + parsers.eof)
12401     end
12402
12403     local fencetail = function(char)
12404         return parsers.check_trail_no_rem
12405             * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)

```



```

12406         * parsers.optionalspace * (parsers.newline + parsers.eof)
12407         + parsers.eof
12408     end
12409
12410     local process_fenced_line =
12411     function(s, i, -- luacheck: ignore s i
12412             indent_table, line_content, is_blank)
12413         local remainder = ""
12414         if has_trail(indent_table) then
12415             remainder = indent_table.trail.internal_remainder
12416         end
12417
12418         if is_blank
12419             and get_last_indent_name(indent_table) == "li" then
12420             remainder = ""
12421         end
12422
12423         local str = remainder .. line_content
12424         local index = 1
12425         local remaining = fenceindent
12426
12427         while true do
12428             local c = str:sub(index, index)
12429             if c == " " and remaining > 0 then
12430                 remaining = remaining - 1
12431                 index = index + 1
12432             elseif c == "\t" and remaining > 3 then
12433                 remaining = remaining - 4
12434                 index = index + 1
12435             else
12436                 break
12437             end
12438         end
12439
12440         return true, str:sub(index)
12441     end
12442
12443     local fencedline = function(char)
12444         return Cmt( Cb("indent_info")
12445                     * C(parsers.line - fencetail(char))
12446                     * Cc(false), process_fenced_line)
12447     end
12448
12449     local blankfencedline
12450     = Cmt( Cb("indent_info")
12451           * C(parsers.blankline)
12452           * Cc(true), process_fenced_line)

```

```

12453
12454     local TildeFencedCode
12455         = fencehead(parsers.tilde, tilde_infostring)
12456         * Cs(( (parsers.check_minimal_blank_indent / "")
12457             * blankfencedline
12458             + ( parsers.check_minimal_indent / "")
12459             * fencedline(parsers.tilde))^0)
12460         * ( (parsers.check_minimal_indent / "")
12461             * fencetail(parsers.tilde) + parsers.succeed)
12462
12463     local BacktickFencedCode
12464         = fencehead(parsers.backtick, backtick_infostring)
12465         * Cs(( (parsers.check_minimal_blank_indent / "")
12466             * blankfencedline
12467             + (parsers.check_minimal_indent / "")
12468             * fencedline(parsers.backtick))^0)
12469         * ( (parsers.check_minimal_indent / "")
12470             * fencetail(parsers.backtick) + parsers.succeed)
12471
12472     local infostring_with_attributes
12473         = Ct(C((parsers.linechar
12474             - ( parsers.optionalspace
12475             * parsers.attributes))^0)
12476             * parsers.optionalspace
12477             * Ct(parsers.attributes))
12478
12479     local FencedCode
12480         = ((TildeFencedCode + BacktickFencedCode)
12481         / function(infostring, code)
12482             local expanded_code = self.expandtabs(code)
12483
12484             if allow_raw_blocks then
12485                 local raw_attr = lpeg.match(parsers.raw_attribute,
12486                                         infostring)
12487
12488                 if raw_attr then
12489                     return writer.rawBlock(expanded_code, raw_attr)
12490                 end
12491             end
12492
12493             local attr = nil
12494             if allow_attributes then
12495                 local match = lpeg.match(infostring_with_attributes,
12496                                         infostring)
12497
12498                 if match then
12499                     infostring, attr = table.unpack(match)
12500                 end
12501             end
12502         end)

```

```

12500         return writer.fencedCode(expanded_code, infostring, attr)
12501     end)
12502
12503     self.insert_pattern("Block after Verbatim",
12504                         FencedCode, "FencedCode")
12505
12506     local fencestart
12507     if blank_before_code_fence then
12508         fencestart = parsers.fail
12509     else
12510         fencestart = fencehead(parsers.backtick, backtick_infostring)
12511                     + fencehead(parsers.tilde, tilde_infostring)
12512     end
12513
12514     self.update_rule("EndlineExceptions", function(previous_pattern)
12515         if previous_pattern == nil then
12516             previous_pattern = parsers.EndlineExceptions
12517         end
12518         return previous_pattern + fencestart
12519     end)
12520
12521     self.add_special_character("`")
12522     self.add_special_character("~")
12523 end
12524 }
12525 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.

```

12526 M.extensions.fenced_divs = function(blank_before_div_fence)
12527     return {
12528         name = "built-in fenced_divs syntax extension",
12529         extend_writer = function(self)

```

Define `writer->div_begin` as a function that will transform the beginning of an input fenced div with with attributes `attributes` to the output format.

```

12530         function self.div_begin(attributes)
12531             local start_output
12532             = {"\\markdownRendererFencedDivAttributeContextBegin\n",
12533               self.attributes(attributes)}
12534             local end_output
12535             = {"\\markdownRendererFencedDivAttributeContextEnd{}}"}
12536             return self.push_attributes(

```

```

12537         "div", attributes, start_output, end_output)
12538     end

```

Define `writer->div_end` as a function that will produce the end of a fenced div in the output format.

```

12539     function self.div_end()
12540         return self.pop_attributes("div")
12541     end
12542 end, extend_reader = function(self)
12543     local parsers = self.parsers
12544     local writer = self.writer

```

Define basic patterns for matching the opening and the closing tag of a div.

```

12545     local fenced_div_infostring
12546         = C((parsers.linechar
12547             - ( parsers.spacechar^1
12548               * parsers.colon^1))^1)
12549
12550     local fenced_div_begin = parsers.nonindentspace
12551         * parsers.colon^3
12552         * parsers.optionalspace
12553         * fenced_div_infostring
12554         * ( parsers.spacechar^1
12555           * parsers.colon^1)^0
12556         * parsers.optionalspace
12557         * (parsers.newline + parsers.eof)
12558
12559     local fenced_div_end = parsers.nonindentspace
12560         * parsers.colon^3
12561         * parsers.optionalspace
12562         * (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.

```

12563     self.initialize_named_group("fenced_div_level", "0")
12564     self.initialize_named_group("fenced_div_num_opening_indents")
12565
12566     local function increment_div_level()
12567         local push_indent_table =
12568             function(s, i, indent_table, -- luacheck: ignore s i
12569                 fenced_div_num_opening_indents, fenced_div_level)
12570                 fenced_div_level = tonumber(fenced_div_level) + 1
12571                 local num_opening_indents = 0
12572                 if indent_table.indents ~= nil then

```

```

12573         num_opening_indents = #indent_table.indents
12574     end
12575     fenced_div_num_opening_indents[fenced_div_level]
12576         = num_opening_indents
12577     return true, fenced_div_num_opening_indents
12578 end
12579
12580 local increment_level =
12581     function(s, i, fenced_div_level) -- luacheck: ignore s i
12582         fenced_div_level = tonumber(fenced_div_level) + 1
12583         return true, tostring(fenced_div_level)
12584     end
12585
12586     return Cg( Cmt( Cb("indent_info")
12587         * Cb("fenced_div_num_opening_indents")
12588         * Cb("fenced_div_level"), push_indent_table)
12589         , "fenced_div_num_opening_indents")
12590     * Cg( Cmt( Cb("fenced_div_level"), increment_level)
12591         , "fenced_div_level")
12592 end
12593
12594 local function decrement_div_level()
12595     local pop_indent_table =
12596         function(s, i, -- luacheck: ignore s i
12597             fenced_div_indent_table, fenced_div_level)
12598             fenced_div_level = tonumber(fenced_div_level)
12599             fenced_div_indent_table[fenced_div_level] = nil
12600             return true, tostring(fenced_div_level - 1)
12601         end
12602
12603     return Cg( Cmt( Cb("fenced_div_num_opening_indents")
12604         * Cb("fenced_div_level"), pop_indent_table)
12605         , "fenced_div_level")
12606 end
12607
12608
12609 local non_fenced_div_block
12610     = parsers.check_minimal_indent * V("Block")
12611     - parsers.check_minimal_indent_and_trail * fenced_div_end
12612
12613 local non_fenced_div_paragraph
12614     = parsers.check_minimal_indent * V("Paragraph")
12615     - parsers.check_minimal_indent_and_trail * fenced_div_end
12616
12617 local blank = parsers.minimally_indented_blank
12618
12619 local block_separated = parsers.block_sep_group(blank)

```

```

12620             * non_fenced_div_block
12621
12622     local loop_body_pair
12623     = parsers.create_loop_body_pair(block_separated,
12624                                     non_fenced_div_paragraph,
12625                                     parsers.block_sep_group(blank),
12626                                     parsers.par_sep_group(blank))
12627
12628     local content_loop = ( non_fenced_div_block
12629                           * loop_body_pair.block^0
12630                           + non_fenced_div_paragraph
12631                           * block_separated
12632                           * loop_body_pair.block^0
12633                           + non_fenced_div_paragraph
12634                           * loop_body_pair.par^0)
12635     * blank^0
12636
12637     local FencedDiv = fenced_div_begin
12638     / function (infostring)
12639         local attr
12640         = lpeg.match(Ct(parsers.attributes),
12641                     infostring)
12642         if attr == nil then
12643             attr = {"." .. infostring}
12644         end
12645         return attr
12646     end
12647     / writer.div_begin
12648     * increment_div_level()
12649     * parsers.skipblanklines
12650     * Ct(content_loop)
12651     * parsers.minimally_indented_blank^0
12652     * parsers.check_minimal_indent_and_trail
12653     * fenced_div_end
12654     * decrement_div_level()
12655     * (Cc("") / writer.div_end)
12656
12657     self.insert_pattern("Block after Verbatim",
12658                       FencedDiv, "FencedDiv")
12659
12660     self.add_special_character(":")
12661

```

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.

```

12662     local function is_inside_div()

```

```

12663     local check_div_level =
12664         function(s, i, fenced_div_level) -- luacheck: ignore s i
12665             fenced_div_level = tonumber(fenced_div_level)
12666             return fenced_div_level > 0
12667         end
12668
12669     return Cmt(Cb("fenced_div_level"), check_div_level)
12670 end
12671
12672 local function check_indent()
12673     local compare_indent =
12674         function(s, i, indent_table, -- luacheck: ignore s i
12675             fenced_div_num_opening_indents, fenced_div_level)
12676             fenced_div_level = tonumber(fenced_div_level)
12677             local num_current_indents
12678                 = ( indent_table.current_line_indents ~= nil and
12679                     #indent_table.current_line_indents) or 0
12680             local num_opening_indents
12681                 = fenced_div_num_opening_indents[fenced_div_level]
12682             return num_current_indents == num_opening_indents
12683         end
12684
12685     return Cmt( Cb("indent_info")
12686         * Cb("fenced_div_num_opening_indents")
12687         * Cb("fenced_div_level"), compare_indent)
12688 end
12689
12690 local fencestart = is_inside_div()
12691     * fenced_div_end
12692     * check_indent()
12693
12694 if not blank_before_div_fence then
12695     self.update_rule("EndlineExceptions", function(previous_pattern)
12696         if previous_pattern == nil then
12697             previous_pattern = parsers.EndlineExceptions
12698         end
12699         return previous_pattern + fencestart
12700     end)
12701 end
12702 end
12703 }
12704 end

```

### 3.1.7.8 Header Attributes

The `extensions.header_attributes` function implements the Pandoc header attribute syntax extension.

```

12705 M.extensions.header_attributes = function()
12706   return {
12707     name = "built-in header_attributes syntax extension",
12708     extend_writer = function()
12709     end, extend_reader = function(self)
12710       local parsers = self.parsers
12711       local writer = self.writer
12712
12713       local function strip_atx_end(s)
12714         return s:gsub("%s+##%s*$", "")
12715       end
12716
12717       local AtxHeading = Cg(parsers.heading_start, "level")
12718         * parsers.optionalspace
12719         * (C(((parsers.linechar
12720           - (parsers.attributes
12721             * parsers.optionalspace
12722             * parsers.newline)))
12723           * (parsers.linechar
12724             - parsers.lbrace)^0)^1)
12725           / strip_atx_end
12726           / parsers.parse_heading_text)
12727         * Cg(Ct(parsers.newline
12728           + (parsers.attributes
12729             * parsers.optionalspace
12730             * parsers.newline))), "attributes")
12731         * Cb("level")
12732         * Cb("attributes")
12733         / writer.heading
12734
12735       local function strip_trailing_spaces(s)
12736         return s:gsub("%s*$", "")
12737       end
12738
12739       local heading_line = (parsers.linechar
12740         - (parsers.attributes
12741           * parsers.optionalspace
12742           * parsers.newline))^1
12743         - parsers.thematic_break_lines
12744
12745       local heading_text
12746         = heading_line
12747         * ( (V("Endline") / "\n")
12748           * (heading_line - parsers.heading_level))^0
12749         * parsers.newline^-1
12750
12751       local SettextHeading

```



```

12752     = parsers.freeze_trail * parsers.check_trail_no_rem
12753     * #(heading_text
12754       * (parsers.attributes
12755         * parsers.optionalspace
12756         * parsers.newline)~-1
12757       * parsers.check_minimal_indent
12758       * parsers.check_trail
12759       * parsers.heading_level)
12760     * Cs(heading_text) / strip_trailing_spaces
12761     / parsers.parse_heading_text
12762     * Cg(Ct((parsers.attributes
12763       * parsers.optionalspace
12764       * parsers.newline)~-1), "attributes")
12765     * parsers.check_minimal_indent_and_trail * parsers.heading_level
12766     * Cb("attributes")
12767     * parsers.newline
12768     * parsers.unfreeze_trail
12769     / writer.heading
12770
12771     local Heading = AtxHeading + SetextHeading
12772     self.update_rule("Heading", Heading)
12773 end
12774 }
12775 end

```

### 3.1.7.9 Inline Code Attributes

The `extensions.inline_code_attributes` function implements the Pandoc inline code attribute syntax extension.

```

12776 M.extensions.inline_code_attributes = function()
12777   return {
12778     name = "built-in inline_code_attributes syntax extension",
12779     extend_writer = function()
12780       end, extend_reader = function(self)
12781         local writer = self.writer
12782
12783         local CodeWithAttributes = parsers.inticks
12784           * Ct(parsers.attributes)
12785           / writer.code
12786
12787         self.insert_pattern("Inline before Code",
12788           CodeWithAttributes,
12789           "CodeWithAttributes")
12790       end
12791     }
12792 end

```

### 3.1.7.10 Line Blocks

The `extensions.line_blocks` function implements the Pandoc line block syntax extension.

```
12793 M.extensions.line_blocks = function()
12794   return {
12795     name = "built-in line_blocks syntax extension",
12796     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.

```
12797     function self.lineblock(lines)
12798       if not self.is_writing then return "" end
12799       local buffer = {}
12800       for i = 1, #lines - 1 do
12801         buffer[#buffer + 1] = { lines[i], self.hard_line_break }
12802       end
12803       buffer[#buffer + 1] = lines[#lines]
12804
12805       return {"\\markdownRendererLineBlockBegin\\n"
12806             ,buffer,
12807             "\\n\\markdownRendererLineBlockEnd "}
12808     end
12809   end, extend_reader = function(self)
12810     local parsers = self.parsers
12811     local writer = self.writer
12812
12813     local LineBlock
12814     = Ct((Cs(( (parsers.pipe * parsers.space) / ""
12815               * ((parsers.space)/entities.char_entity("nbsp"))^0
12816               * parsers.linechar^0 * (parsers.newline/""))
12817         * (-parsers.pipe
12818           * (parsers.space^1/" ")
12819           * parsers.linechar^1
12820           * (parsers.newline/"")
12821           )^0
12822         * (parsers.blankline/"")^0)
12823       / self.parser_functions.parse_inlines)^1)
12824     / writer.lineblock
12825
12826     self.insert_pattern("Block after Blockquote",
12827                       LineBlock, "LineBlock")
12828   end
12829 }
12830 end
```

### 3.1.7.11 Marked text

The `extensions.mark` function implements the Pandoc mark syntax extension.

```
12831 M.extensions.mark = function()
12832   return {
12833     name = "built-in mark syntax extension",
12834     extend_writer = function(self)
```

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

```
12835     function self.mark(s)
12836       if self.flatten_inlines then return s end
12837       return {"\\markdownRendererMark{" , s, "}"}
12838     end
12839   end, extend_reader = function(self)
12840     local parsers = self.parsers
12841     local writer = self.writer
12842
12843     local doubleequals = P("==")
12844
12845     local Mark
12846       = parsers.between(V("Inline"), doubleequals, doubleequals)
12847       / function (inlines) return writer.mark(inlines) end
12848
12849     self.add_special_character("=")
12850     self.insert_pattern("Inline before LinkAndEmph",
12851                       Mark, "Mark")
12852   end
12853 }
12854 end
```

### 3.1.7.12 Link Attributes

The `extensions.link_attributes` function implements the Pandoc link attribute syntax extension.

```
12855 M.extensions.link_attributes = function()
12856   return {
12857     name = "built-in link_attributes syntax extension",
12858     extend_writer = function()
12859     end, extend_reader = function(self)
12860       local parsers = self.parsers
12861       local options = self.options
12862
```

The following patterns define link reference definitions with attributes.

```
12863     local define_reference_parser
12864       = (parsers.check_trail / "")
12865       * parsers.link_label
12866       * parsers.colon
12867       * parsers.spnlc * parsers.url
```

```

12868         * ( parsers.spnlc_sep * parsers.title
12869         * (parsers.spnlc * Ct(parsers.attributes))
12870         * parsers.only_blank
12871         + parsers.spnlc_sep * parsers.title * parsers.only_blank
12872         + Cc("") * (parsers.spnlc * Ct(parsers.attributes))
12873         * parsers.only_blank
12874         + Cc("") * parsers.only_blank)
12875
12876     local ReferenceWithAttributes = define_reference_parser
12877                                   / self.register_link
12878
12879     self.update_rule("Reference", ReferenceWithAttributes)
12880

```

The following patterns define direct and indirect links with attributes.

```

12881
12882     local LinkWithAttributesAndEmph
12883     = Ct(parsers.link_and_emph_table * Cg(Cc(true),
12884       "match_link_attributes"))
12885     / self.defer_link_and_emphasis_processing
12886
12887     self.update_rule("LinkAndEmph", LinkWithAttributesAndEmph)
12888

```

The following patterns define autolinks with attributes.

```

12889     local AutoLinkUrlWithAttributes
12890     = parsers.auto_link_url
12891     * Ct(parsers.attributes)
12892     / self.auto_link_url
12893
12894     self.insert_pattern("Inline before AutoLinkUrl",
12895       AutoLinkUrlWithAttributes,
12896       "AutoLinkUrlWithAttributes")
12897
12898     local AutoLinkEmailWithAttributes
12899     = parsers.auto_link_email
12900     * Ct(parsers.attributes)
12901     / self.auto_link_email
12902
12903     self.insert_pattern("Inline before AutoLinkEmail",
12904       AutoLinkEmailWithAttributes,
12905       "AutoLinkEmailWithAttributes")
12906
12907     if options.relativeReferences then
12908
12909         local AutoLinkRelativeReferenceWithAttributes
12910         = parsers.auto_link_relative_reference
12911         * Ct(parsers.attributes)

```

```

12912             / self.auto_link_url
12913
12914         self.insert_pattern(
12915             "Inline before AutoLinkRelativeReference",
12916             AutoLinkRelativeReferenceWithAttributes,
12917             "AutoLinkRelativeReferenceWithAttributes")
12918
12919     end
12920
12921 end
12922 }
12923 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.

```

12924 M.extensions.notes = function(notes, inline_notes)
12925     assert(notes or inline_notes)
12926     return {
12927         name = "built-in notes syntax extension",
12928         extend_writer = function(self)

```

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

```

12929         function self.note(s)
12930             if self.flatten_inlines then return "" end
12931             return {"\\markdownRendererNote{",s,""}
12932         end
12933     end, extend_reader = function(self)
12934         local parsers = self.parsers
12935         local writer = self.writer
12936
12937         local rawnotes = parsers.rawnotes
12938
12939         if inline_notes then
12940             local InlineNote
12941                 = parsers.circumflex
12942                 * ( parsers.link_label
12943                   / self.parser_functions.parse_inlines_no_inline_note)
12944                 / writer.note
12945
12946             self.insert_pattern("Inline after LinkAndEmph",
12947                               InlineNote, "InlineNote")
12948         end

```

```

12949     if notes then
12950         local function strip_first_char(s)
12951             return s:sub(2)
12952         end
12953
12954         local RawNoteRef
12955             = #(parsers.lbracket * parsers.circumflex)
12956             * parsers.link_label / strip_first_char
12957
12958         -- like indirect_link
12959         local function lookup_note(ref)
12960             return writer.defer_call(function()
12961                 local found = rawnotes[self.normalize_tag(ref)]
12962                 if found then
12963                     return writer.note(
12964                         self.parser_functions.parse_blocks_nested(found))
12965                 else
12966                     local text = string.format(
12967                         'Undefined note reference "%s"', ref)
12968                     local more = string.format(
12969                         "Look for the text `[~%s]`.", ref)
12970                     return {writer.warning(text, more), "[",
12971                         self.parser_functions.parse_inlines("^" .. ref), ""]}
12972                 end
12973             end)
12974         end
12975
12976         local function register_note(ref,rawnote)
12977             local normalized_tag = self.normalize_tag(ref)
12978             if rawnotes[normalized_tag] == nil then
12979                 rawnotes[normalized_tag] = rawnote
12980                 return ""
12981             else
12982                 local text
12983                     = string.format('Multiply defined note reference "%s"',
12984                                   ref)
12985                 local more
12986                     = string.format("Look for the text `[~%s]: ...`.", ref)
12987                 return writer.warning(text, more)
12988             end
12989         end
12990
12991         local NoteRef = RawNoteRef / lookup_note
12992
12993         local optionally_indented_line
12994             = parsers.check_optional_indent_and_any_trail * parsers.line
12995

```

```

12996     local blank
12997         = parsers.check_optional_blank_indent_and_any_trail
12998         * parsers.optionalspace * parsers.newline
12999
13000     local chunk
13001         = Cs(parsers.line
13002             * (optionally_indented_line - blank)^0)
13003
13004     local indented_blocks = function(bl)
13005         return Cs( bl
13006             * ( blank^1 * (parsers.check_optional_indent / "")
13007               * parsers.check_code_trail
13008               * -parsers.blankline * bl)^0)
13009     end
13010
13011     local NoteBlock
13012         = parsers.check_trail_no_rem
13013         * RawNoteRef * parsers.colon
13014         * parsers.spnlc * indented_blocks(chunk)
13015         / register_note
13016
13017     self.update_rule("Reference", function(previous_pattern)
13018         if previous_pattern == nil then
13019             previous_pattern = parsers.Reference
13020         end
13021         return NoteBlock + previous_pattern
13022     end)
13023
13024     self.insert_pattern("Inline before LinkAndEmph",
13025                         NoteRef, "NoteRef")
13026 end
13027
13028 self.add_special_character("^")
13029 end
13030 }
13031 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.

```

13032 M.extensions.pipe_tables = function(table_captions, table_attributes)
13033

```

```

13034 local function make_pipe_table_rectangular(rows)
13035     local num_columns = #rows[2]
13036     local rectangular_rows = {}
13037     for i = 1, #rows do
13038         local row = rows[i]
13039         local rectangular_row = {}
13040         for j = 1, num_columns do
13041             rectangular_row[j] = row[j] or ""
13042         end
13043         table.insert(rectangular_rows, rectangular_row)
13044     end
13045     return rectangular_rows
13046 end
13047
13048 local function pipe_table_row(allow_empty_first_column
13049                             , nonempty_column
13050                             , column_separator
13051                             , column)
13052     local row_beginning
13053     if allow_empty_first_column then
13054         row_beginning = -- empty first column
13055                        #(parsers.spacechar^4
13056                        * column_separator)
13057                        * parsers.optionalspace
13058                        * column
13059                        * parsers.optionalspace
13060                        -- non-empty first column
13061                        + parsers.nonindentSPACE
13062                        * nonempty_column~-1
13063                        * parsers.optionalspace
13064     else
13065         row_beginning = parsers.nonindentSPACE
13066                        * nonempty_column~-1
13067                        * parsers.optionalspace
13068     end
13069
13070     return Ct(row_beginning
13071              * (-- single column with no leading pipes
13072                #(column_separator
13073                  * parsers.optionalspace
13074                  * parsers.newline)
13075                * column_separator
13076                * parsers.optionalspace
13077                -- single column with leading pipes or
13078                -- more than a single column
13079                + (column_separator
13080                  * parsers.optionalspace

```



```

13081             * column
13082             * parsers.optionalspace)^1
13083         * (column_separator
13084             * parsers.optionalspace)^-1))
13085     end
13086
13087     return {
13088         name = "built-in pipe_tables syntax extension",
13089         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.

```

13090         function self.table(rows, caption, attributes)
13091             if not self.is_writing then return "" end
13092             local buffer = {}
13093             if attributes ~= nil then
13094                 table.insert(buffer,
13095                     "\\markdownRendererTableAttributeContextBegin\n")
13096                 table.insert(buffer, self.attributes(attributes))
13097             end
13098             table.insert(buffer,
13099                 {"\\markdownRendererTable{",
13100                     caption or "", "}{" , #rows - 1, "}{" ,
13101                     #rows[1], "}"})
13102             local temp = rows[2] -- put alignments on the first row
13103             rows[2] = rows[1]
13104             rows[1] = temp
13105             for i, row in ipairs(rows) do
13106                 table.insert(buffer, "{")
13107                 for _, column in ipairs(row) do
13108                     if i > 1 then -- do not use braces for alignments
13109                         table.insert(buffer, "{")
13110                     end
13111                     table.insert(buffer, column)
13112                     if i > 1 then
13113                         table.insert(buffer, "}")
13114                     end
13115                 end
13116                 table.insert(buffer, "}")
13117             end
13118             if attributes ~= nil then
13119                 table.insert(buffer,
13120                     "\\markdownRendererTableAttributeContextEnd{")
13121             end
13122             return buffer
13123         end
13124     end, extend_reader = function(self)

```

```

13125     local parsers = self.parsers
13126     local writer = self.writer
13127
13128     local table_hline_separator = parsers.pipe + parsers.plus
13129
13130     local table_hline_column = (parsers.dash
13131                                - #(parsers.dash
13132                                   * (parsers.spacechar
13133                                      + table_hline_separator
13134                                      + parsers.newline)))^1
13135                                * (parsers.colon * Cc("r")
13136                                   + parsers.dash * Cc("d"))
13137                                + parsers.colon
13138                                * (parsers.dash
13139                                   - #(parsers.dash
13140                                      * (parsers.spacechar
13141                                         + table_hline_separator
13142                                         + parsers.newline)))^1
13143                                * (parsers.colon * Cc("c")
13144                                   + parsers.dash * Cc("l"))
13145
13146     local table_hline = pipe_table_row(false
13147                                         , table_hline_column
13148                                         , table_hline_separator
13149                                         , table_hline_column)
13150
13151     local table_caption_beginning
13152     = ( parsers.check_minimal_blank_indent_and_any_trail_no_rem
13153        * parsers.optionalspace * parsers.newline)^0
13154        * parsers.check_minimal_indent_and_trail
13155        * (P("Table")^-1 * parsers.colon)
13156        * parsers.optionalspace
13157
13158     local function strip_trailing_spaces(s)
13159         return s:gsub("%s*$","")
13160     end
13161
13162     local table_row
13163     = pipe_table_row(true
13164                      , (C((parsers.linechar - parsers.pipe)^1)
13165                         / strip_trailing_spaces
13166                         / self.parser_functions.parse_inlines)
13167                      , parsers.pipe
13168                      , (C((parsers.linechar - parsers.pipe)^0)
13169                         / strip_trailing_spaces
13170                         / self.parser_functions.parse_inlines))
13171

```

```

13172     local table_caption
13173     if table_captions then
13174         table_caption = #table_caption_beginning
13175             * table_caption_beginning
13176         if table_attributes then
13177             table_caption = table_caption
13178                 * (C(((( parsers.linechar
13179                     - (parsers.attributes
13180                         * parsers.optionalspace
13181                         * parsers.newline
13182                         * -( parsers.optionalspace
13183                             * parsers.linechar)))
13184                     + ( parsers.newline
13185                         * #( parsers.optionalspace
13186                             * parsers.linechar)
13187                         * C(parsers.optionalspace)
13188                             / writer.space))
13189                     * (parsers.linechar
13190                         - parsers.lbrace)^0)^1)
13191                     / self.parser_functions.parse_inlines)
13192                 * (parsers.newline
13193                     + ( Ct(parsers.attributes)
13194                         * parsers.optionalspace
13195                         * parsers.newline))
13196         else
13197             table_caption = table_caption
13198                 * C(( parsers.linechar
13199                     + ( parsers.newline
13200                         * #( parsers.optionalspace
13201                             * parsers.linechar)
13202                         * C(parsers.optionalspace)
13203                             / writer.space))^1)
13204                     / self.parser_functions.parse_inlines
13205                 * parsers.newline
13206         end
13207     else
13208         table_caption = parsers.fail
13209     end
13210
13211     local PipeTable
13212     = Ct( table_row * parsers.newline
13213         * (parsers.check_minimal_indent_and_trail / {})
13214         * table_hline * parsers.newline
13215         * ( (parsers.check_minimal_indent / {})
13216             * table_row * parsers.newline)^0)
13217     / make_pipe_table_rectangular
13218     * table_caption^~1

```

```

13219         / writer.table
13220
13221         self.insert_pattern("Block after Blockquote",
13222                             PipeTable, "PipeTable")
13223     end
13224 }
13225 end

```

### 3.1.7.15 Raw Attributes

The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```

13226 M.extensions.raw_inline = function()
13227   return {
13228     name = "built-in raw_inline syntax extension",
13229     extend_writer = function(self)
13230       local options = self.options
13231

```

Define `writer->rawInline` as a function that will transform an input inline raw span `s` with the raw attribute `attr` to the output format.

```

13232     function self.rawInline(s, attr)
13233       if not self.is_writing then return "" end
13234       if self.flatten_inlines then return s end
13235       local name = util.cache_verbatim(options.cacheDir, s)
13236       return {"\\markdownRendererInputRawInline{" ,
13237              name,"}{" , self.string(attr),"}" }
13238     end
13239   end, extend_reader = function(self)
13240     local writer = self.writer
13241
13242     local RawInline = parsers.inticks
13243                       * parsers.raw_attribute
13244                       / writer.rawInline
13245
13246     self.insert_pattern("Inline before Code",
13247                         RawInline, "RawInline")
13248   end
13249 }
13250 end

```

### 3.1.7.16 Strike-Through

The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```

13251 M.extensions.strike_through = function()
13252   return {
13253     name = "built-in strike_through syntax extension",

```

```
13254     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.

```
13255         function self.strike_through(s)
13256             if self.flatten_inlines then return s end
13257             return {"\\markdownRendererStrikeThrough{" ,s,"}"}
13258         end
13259     end, extend_reader = function(self)
13260         local parsers = self.parsers
13261         local writer = self.writer
13262
13263         local StrikeThrough = (
13264             parsers.between(parsers.Inline, parsers.doubletildes,
13265                             parsers.doubletildes)
13266         ) / writer.strike_through
13267
13268         self.insert_pattern("Inline after LinkAndEmph",
13269                             StrikeThrough, "StrikeThrough")
13270
13271         self.add_special_character("~")
13272     end
13273 }
13274 end
```

### 3.1.7.17 Subscripts

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

```
13275 M.extensions.subscripts = function()
13276     return {
13277         name = "built-in subscripts syntax extension",
13278         extend_writer = function(self)
```

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

```
13279         function self.subscript(s)
13280             if self.flatten_inlines then return s end
13281             return {"\\markdownRendererSubscript{" ,s,"}"}
13282         end
13283     end, extend_reader = function(self)
13284         local parsers = self.parsers
13285         local writer = self.writer
13286
13287         local Subscript = (
13288             parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
13289         ) / writer.subscript
13290
```

```

13291         self.insert_pattern("Inline after LinkAndEmph",
13292                               Subscript, "Subscript")
13293
13294         self.add_special_character("~")
13295     end
13296 }
13297 end

```

### 3.1.7.18 Superscripts

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

```

13298 M.extensions.superscripts = function()
13299     return {
13300         name = "built-in superscripts syntax extension",
13301         extend_writer = function(self)

```

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

```

13302             function self.superscript(s)
13303                 if self.flatten_inlines then return s end
13304                 return {"\\markdownRendererSuperscript{" ,s,""} }
13305             end
13306         end, extend_reader = function(self)
13307             local parsers = self.parsers
13308             local writer = self.writer
13309
13310             local Superscript = (
13311                 parsers.between(parsers.Str, parsers.circumflex,
13312                                parsers.circumflex)
13313             ) / writer.superscript
13314
13315             self.insert_pattern("Inline after LinkAndEmph",
13316                               Superscript, "Superscript")
13317
13318             self.add_special_character("^")
13319         end
13320     }
13321 end

```

### 3.1.7.19 T<sub>E</sub>X Math

The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```

13322 M.extensions.tex_math = function(tex_math_dollars,
13323                                   tex_math_single_backslash,
13324                                   tex_math_double_backslash)
13325     return {

```

```

13326     name = "built-in tex_math syntax extension",
13327     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.

```

13328         function self.display_math(s)
13329             if self.flatten_inlines then return s end
13330             return {"\\markdownRendererDisplayMath{" ,self.math(s),"}"}
13331         end

```

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

```

13332         function self.inline_math(s)
13333             if self.flatten_inlines then return s end
13334             return {"\\markdownRendererInlineMath{" ,self.math(s),"}"}
13335         end
13336     end, extend_reader = function(self)
13337         local parsers = self.parsers
13338         local writer = self.writer
13339
13340         local function between(p, starter, ender)
13341             return (starter * Cs(p * (p - ender)^0) * ender)
13342         end
13343
13344         local function strip_preceding_whitespaces(str)
13345             return str:gsub("^%s*(.)$", "%1")
13346         end
13347
13348         local allowed_before_closing
13349         = B( parsers.backslash * parsers.any
13350           + parsers.any * (parsers.any - parsers.backslash))
13351
13352         local allowed_before_closing_no_space
13353         = B( parsers.backslash * parsers.any
13354           + parsers.any * (parsers.nonspacechar - parsers.backslash))
13355

```

The following patterns implement the Pandoc dollar math syntax extension.

```

13356     local dollar_math_content
13357     = (parsers.newline * (parsers.check_optional_indent / "")
13358       + parsers.backslash^1
13359       * parsers.linechar)
13360       - parsers.blankline^2
13361       - parsers.dollar
13362
13363     local inline_math_opening_dollars = parsers.dollar
13364                                         * #(parsers.nonspacechar)
13365

```

```

13366     local inline_math_closing_dollars
13367         = allowed_before_closing_no_space
13368         * parsers.dollar
13369         * -#(parsers.digit)
13370
13371     local inline_math_dollars = between(Cs( dollar_math_content),
13372                                         inline_math_opening_dollars,
13373                                         inline_math_closing_dollars)
13374
13375     local display_math_opening_dollars = parsers.dollar
13376                                         * parsers.dollar
13377
13378     local display_math_closing_dollars = parsers.dollar
13379                                         * parsers.dollar
13380
13381     local display_math_dollars = between(Cs( dollar_math_content),
13382                                         display_math_opening_dollars,
13383                                         display_math_closing_dollars)

```

The following patterns implement the Pandoc single and double backslash math syntax extensions.

```

13384     local backslash_math_content
13385         = (parsers.newline * (parsers.check_optional_indent / ""))
13386         + parsers.linechar)
13387         - parsers.blankline^2

```

The following patterns implement the Pandoc double backslash math syntax extension.

```

13388     local inline_math_opening_double = parsers.backslash
13389                                     * parsers.backslash
13390                                     * parsers.lparent
13391
13392     local inline_math_closing_double = allowed_before_closing
13393                                       * parsers.spacechar^0
13394                                       * parsers.backslash
13395                                       * parsers.backslash
13396                                       * parsers.rparent
13397
13398     local inline_math_double = between(Cs( backslash_math_content),
13399                                       inline_math_opening_double,
13400                                       inline_math_closing_double)
13401                                   / strip_preceding_whitespaces
13402
13403     local display_math_opening_double = parsers.backslash
13404                                       * parsers.backslash
13405                                       * parsers.lbracket
13406
13407     local display_math_closing_double = allowed_before_closing

```



```

13408             * parsers.spacechar~0
13409             * parsers.backslash
13410             * parsers.backslash
13411             * parsers.rbracket
13412
13413     local display_math_double = between(Cs( backslash_math_content),
13414                                         display_math_opening_double,
13415                                         display_math_closing_double)
13416                                         / strip_preceding_whitespaces

```

The following patterns implement the Pandoc single backslash math syntax extension.

```

13417     local inline_math_opening_single = parsers.backslash
13418                                     * parsers.lparent
13419
13420     local inline_math_closing_single = allowed_before_closing
13421                                     * parsers.spacechar~0
13422                                     * parsers.backslash
13423                                     * parsers.rparent
13424
13425     local inline_math_single = between(Cs( backslash_math_content),
13426                                         inline_math_opening_single,
13427                                         inline_math_closing_single)
13428                                         / strip_preceding_whitespaces
13429
13430     local display_math_opening_single = parsers.backslash
13431                                     * parsers.lbracket
13432
13433     local display_math_closing_single = allowed_before_closing
13434                                     * parsers.spacechar~0
13435                                     * parsers.backslash
13436                                     * parsers.rbracket
13437
13438     local display_math_single = between(Cs( backslash_math_content),
13439                                         display_math_opening_single,
13440                                         display_math_closing_single)
13441                                         / strip_preceding_whitespaces
13442
13443     local display_math = parsers.fail
13444
13445     local inline_math = parsers.fail
13446
13447     if tex_math_dollars then
13448         display_math = display_math + display_math_dollars
13449         inline_math = inline_math + inline_math_dollars
13450     end
13451
13452     if tex_math_double_backslash then
13453         display_math = display_math + display_math_double

```

```

13454         inline_math = inline_math + inline_math_double
13455     end
13456
13457     if tex_math_single_backslash then
13458         display_math = display_math + display_math_single
13459         inline_math = inline_math + inline_math_single
13460     end
13461
13462     local TexMath = display_math / writer.display_math
13463                   + inline_math / writer.inline_math
13464
13465     self.insert_pattern("Inline after LinkAndEmph",
13466                       TexMath, "TexMath")
13467
13468     if tex_math_dollars then
13469         self.add_special_character("$")
13470     end
13471
13472     if tex_math_single_backslash or tex_math_double_backslash then
13473         self.add_special_character("\\")
13474         self.add_special_character("[")
13475         self.add_special_character("]")
13476         self.add_special_character("(")
13477         self.add_special_character("(")
13478     end
13479 end
13480 }
13481 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.

```

13482 M.extensions.jekyll_data = function(expect_jekyll_data,
13483                                     ensure_jekyll_data)
13484     return {
13485         name = "built-in jekyll_data syntax extension",
13486         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.

```

13487     function self.jekyllData(d, t, p)
13488         if not self.is_writing then return "" end
13489
13490         local buf = {}
13491
13492         local keys = {}
13493         for k, _ in pairs(d) do
13494             table.insert(keys, k)
13495         end

```

For reproducibility, sort the keys. For mixed string-and-numeric keys, sort numeric keys before string keys.

```

13496         table.sort(keys, function(first, second)
13497             if type(first) ~= type(second) then
13498                 return type(first) < type(second)
13499             else
13500                 return first < second
13501             end
13502         end)
13503
13504         if not p then
13505             table.insert(buf, "\\markdownRendererJekyllDataBegin")
13506         end
13507
13508         local is_sequence = false
13509         if #d > 0 and #d == #keys then
13510             for i=1, #d do
13511                 if d[i] == nil then
13512                     goto not_a_sequence
13513                 end
13514             end
13515             is_sequence = true
13516         end
13517         ::not_a_sequence::
13518
13519         if is_sequence then
13520             table.insert(buf,
13521                 "\\markdownRendererJekyllDataSequenceBegin{")
13522             table.insert(buf, self.identifier(p or "null"))
13523             table.insert(buf, "}{" )
13524             table.insert(buf, #keys)
13525             table.insert(buf, "}")
13526         else
13527             table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
13528             table.insert(buf, self.identifier(p or "null"))

```

```

13529         table.insert(buf, "{")
13530         table.insert(buf, #keys)
13531         table.insert(buf, "}")
13532     end
13533
13534     for _, k in ipairs(keys) do
13535         local v = d[k]
13536         local typ = type(v)
13537         k = tostring(k or "null")
13538         if typ == "table" and next(v) ~= nil then
13539             table.insert(
13540                 buf,
13541                 self.jekyllData(v, t, k)
13542             )
13543         else
13544             k = self.identifier(k)
13545             v = tostring(v)
13546             if typ == "boolean" then
13547                 table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
13548                 table.insert(buf, k)
13549                 table.insert(buf, "{")
13550                 table.insert(buf, v)
13551                 table.insert(buf, "}")
13552             elseif typ == "number" then
13553                 table.insert(buf, "\\markdownRendererJekyllDataNumber{")
13554                 table.insert(buf, k)
13555                 table.insert(buf, "{")
13556                 table.insert(buf, v)
13557                 table.insert(buf, "}")
13558             elseif typ == "string" then
13559                 table.insert(buf,
13560                     "\\markdownRendererJekyllDataProgrammaticString{")
13561                 table.insert(buf, k)
13562                 table.insert(buf, "{")
13563                 table.insert(buf, self.identifier(v))
13564                 table.insert(buf, "}")
13565                 table.insert(buf,
13566                     "\\markdownRendererJekyllDataTypographicString{")
13567                 table.insert(buf, k)
13568                 table.insert(buf, "{")
13569                 table.insert(buf, t(v))
13570                 table.insert(buf, "}")
13571             elseif typ == "table" then
13572                 table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
13573                 table.insert(buf, k)
13574                 table.insert(buf, "}")
13575             else

```

```

13576         local error = self.error(format(
13577             "Unexpected type %s for value of "
13578             .. "YAML key %s.", typ, k))
13579         table.insert(buf, error)
13580     end
13581 end
13582 end
13583
13584 if is_sequence then
13585     table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
13586 else
13587     table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
13588 end
13589
13590 if not p then
13591     table.insert(buf, "\\markdownRendererJekyllDataEnd")
13592 end
13593
13594 return buf
13595 end
13596 end, extend_reader = function(self)
13597     local parsers = self.parsers
13598     local writer = self.writer
13599
13600     local JekyllData
13601     = Cmt( C((parsers.line - P("---") - P("..."))^0)
13602         , function(s, i, text) -- luacheck: ignore s i
13603             local data
13604             local ran_ok, _ = pcall(function()
13605                 local tinyyaml = require("tinyyaml")
13606                 data = tinyyaml.parse(text, {timestamps=false})
13607             end)
13608             if ran_ok and data ~= nil then
13609                 return true, writer.jekyllData(data, function(s)
13610                     return self.parser_functions.parse_blocks_nested(s)
13611                 end, nil)
13612             else
13613                 return false
13614             end
13615         end
13616     )
13617
13618     local UnexpectedJekyllData
13619     = P("---")
13620     * parsers.blankline / 0
13621     -- if followed by blank, it's thematic break
13622     * #(-parsers.blankline)

```

```

13623      * JekyllData
13624      * (P("----") + P("..."))
13625
13626      local ExpectedJekyllData
13627      = ( P("----")
13628          * parsers.blankline / 0
13629          -- if followed by blank, it's thematic break
13630          * #(-parsers.blankline)
13631          )^-1
13632      * JekyllData
13633      * (P("----") + P("..."))^-1
13634
13635      if ensure_jekyll_data then
13636          ExpectedJekyllData = ExpectedJekyllData
13637                              * parsers.eof
13638      else
13639          ExpectedJekyllData = ( ExpectedJekyllData
13640                                * (V("Blank")^0 / writer.interblocksep)
13641                                )^-1
13642      end
13643
13644      self.insert_pattern("Block before Blockquote",
13645                          UnexpectedJekyllData, "UnexpectedJekyllData")
13646      if expect_jekyll_data then
13647          self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
13648      end
13649  end
13650 }
13651 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`.

```
13652 function M.new(options)
```

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

```

13653   options = options or {}
13654   setmetatable(options, { __index = function (_, key)
13655       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.

```
13656   local parser_convert = nil
```

```

13657     return function(input, include_flat_output)
13658         local function convert(input)
13659             if parser_convert == nil then

```

Lazy-load [markdown-parser.lua](#) and check that it originates from the same version of the Markdown package.

```

13660         local parser = require("markdown-parser")
13661         if metadata.version ~= parser.metadata.version then
13662             warn("markdown.lua " .. metadata.version .. " used with " ..
13663                 "markdown-parser.lua " .. parser.metadata.version .. ".")
13664         end
13665         parser_convert = parser.new(options)
13666     end
13667     return parser_convert(input)
13668 end

```

If we cache markdown documents, produce the cache file and transform its filename to plain T<sub>E</sub>X 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).

```

13669     local raw_output, flat_output
13670     if options.eagerCache or options.finalizeCache then
13671         local salt = util.salt(options)
13672         local name, result = util.cache(options.cacheDir, input, salt,
13673                                         convert, ".md.tex")
13674         raw_output = [[\input{}} .. name .. [[\}relax]]
13675         flat_output = function()
13676             if result == nil then
13677                 local input_file = assert(io.open(name, "r"),
13678                     [[Could not open file "]] .. name .. [[ for reading]])
13679                 result = assert(input_file:read("*a"))
13680                 assert(input_file:close())
13681             end
13682             return result
13683         end

```

Otherwise, return the result of the conversion directly.

```

13684     else
13685         raw_output = convert(input)
13686         flat_output = function()
13687             return raw_output
13688         end
13689     end

```

If the [finalizeCache](#) option is enabled, populate the frozen cache in the file [frozenCacheFileName](#) with an entry for markdown document number [frozenCacheCounter](#).

```

13690     if options.finalizeCache then
13691         local file, mode
13692         if options.frozenCacheCounter > 0 then
13693             mode = "a"
13694         else
13695             mode = "w"
13696         end
13697         file = assert(io.open(options.frozenCacheFileName, mode),
13698             [[Could not open file ]] .. options.frozenCacheFileName
13699             .. [[ for writing]])
13700         assert(file:write(
13701             [[\expandafter\global\expandafter\def\csname ]]
13702             .. [[markdownFrozenCache]] .. options.frozenCacheCounter
13703             .. [[\endcsname{}]] .. raw_output .. [[]] .. "\n"))
13704         assert(file:close())
13705     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.

```

13706     if include_flat_output then
13707         return raw_output, flat_output
13708     else
13709         return raw_output
13710     end
13711 end
13712 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.

```

13713 function M.new(options)

```

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

```

13714     options = options or {}
13715     setmetatable(options, { __index = function (_, key)
13716         return defaultOptions[key] end })

```

If the singleton cache contains a conversion function for the same `options`, reuse it.

```

13717     if options.singletonCache and singletonCache.convert then
13718         for k, v in pairs(defaultOptions) do
13719             if type(v) == "table" then
13720                 for i = 1, math.max(#singletonCache.options[k], #options[k]) do
13721                     if singletonCache.options[k][i] ~= options[k][i] then
13722                         goto miss
13723                     end
13724                 end

```

The `cacheDir` option is disregarded.

```

13725         elseif k ~= "cacheDir"

```



```

13726         and singletonCache.options[k] ~= options[k] then
13727             goto miss
13728         end
13729     end
13730     return singletonCache.convert
13731 end
13732 ::miss::

```

Apply built-in syntax extensions based on `options`.

```

13733     local extensions = {}
13734
13735     if options.bracketedSpans then
13736         local bracketed_spans_extension = M.extensions.bracketed_spans()
13737         table.insert(extensions, bracketed_spans_extension)
13738     end
13739
13740     if options.contentBlocks then
13741         local content_blocks_extension = M.extensions.content_blocks(
13742             options.contentBlocksLanguageMap)
13743         table.insert(extensions, content_blocks_extension)
13744     end
13745
13746     if options.definitionLists then
13747         local definition_lists_extension = M.extensions.definition_lists(
13748             options.tightLists)
13749         table.insert(extensions, definition_lists_extension)
13750     end
13751
13752     if options.fencedCode then
13753         local fenced_code_extension = M.extensions.fenced_code(
13754             options.blankBeforeCodeFence,
13755             options.fencedCodeAttributes,
13756             options.rawAttribute)
13757         table.insert(extensions, fenced_code_extension)
13758     end
13759
13760     if options.fencedDivs then
13761         local fenced_div_extension = M.extensions.fenced_divs(
13762             options.blankBeforeDivFence)
13763         table.insert(extensions, fenced_div_extension)
13764     end
13765
13766     if options.headerAttributes then
13767         local header_attributes_extension = M.extensions.header_attributes()
13768         table.insert(extensions, header_attributes_extension)
13769     end
13770
13771     if options.inlineCodeAttributes then

```

```

13772     local inline_code_attributes_extension =
13773         M.extensions.inline_code_attributes()
13774     table.insert(extensions, inline_code_attributes_extension)
13775 end
13776
13777 if options.jekyllData then
13778     local jekyll_data_extension = M.extensions.jekyll_data(
13779         options.expectJekyllData, options.ensureJekyllData)
13780     table.insert(extensions, jekyll_data_extension)
13781 end
13782
13783 if options.linkAttributes then
13784     local link_attributes_extension =
13785         M.extensions.link_attributes()
13786     table.insert(extensions, link_attributes_extension)
13787 end
13788
13789 if options.lineBlocks then
13790     local line_block_extension = M.extensions.line_blocks()
13791     table.insert(extensions, line_block_extension)
13792 end
13793
13794 if options.mark then
13795     local mark_extension = M.extensions.mark()
13796     table.insert(extensions, mark_extension)
13797 end
13798
13799 if options.pipeTables then
13800     local pipe_tables_extension = M.extensions.pipe_tables(
13801         options.tableCaptions, options.tableAttributes)
13802     table.insert(extensions, pipe_tables_extension)
13803 end
13804
13805 if options.rawAttribute then
13806     local raw_inline_extension = M.extensions.raw_inline()
13807     table.insert(extensions, raw_inline_extension)
13808 end
13809
13810 if options.strikeThrough then
13811     local strike_through_extension = M.extensions.strike_through()
13812     table.insert(extensions, strike_through_extension)
13813 end
13814
13815 if options.subscripts then
13816     local subscript_extension = M.extensions.subscripts()
13817     table.insert(extensions, subscript_extension)
13818 end

```

```

13819
13820 if options.superscripts then
13821     local superscript_extension = M.extensions.superscripts()
13822     table.insert(extensions, superscript_extension)
13823 end
13824
13825 if options.texMathDollars or
13826     options.texMathSingleBackslash or
13827     options.texMathDoubleBackslash then
13828     local tex_math_extension = M.extensions.tex_math(
13829         options.texMathDollars,
13830         options.texMathSingleBackslash,
13831         options.texMathDoubleBackslash)
13832     table.insert(extensions, tex_math_extension)
13833 end
13834
13835 if options.notes or options.inlineNotes then
13836     local notes_extension = M.extensions.notes(
13837         options.notes, options.inlineNotes)
13838     table.insert(extensions, notes_extension)
13839 end
13840
13841 if options.citations then
13842     local citations_extension
13843         = M.extensions.citations(options.citationNbsps)
13844     table.insert(extensions, citations_extension)
13845 end
13846
13847 if options.fancyLists then
13848     local fancy_lists_extension = M.extensions.fancy_lists()
13849     table.insert(extensions, fancy_lists_extension)
13850 end

```

Apply user-defined syntax extensions based on `options.extensions`.

```

13851 for _, user_extension_filename in ipairs(options.extensions) do
13852     local user_extension = (function(filename)

```

First, load and compile the contents of the user-defined syntax extension.

```

13853     local pathname = assert(util.find_file(filename),
13854         [[Could not locate user-defined syntax extension "]]
13855         .. filename)
13856     local input_file = assert(io.open(pathname, "r"),
13857         [[Could not open user-defined syntax extension "]]
13858         .. pathname .. [[" for reading]])
13859     local input = assert(input_file:read("*a"))
13860     assert(input_file:close())
13861     local user_extension, err = load([[
13862         local sandbox = {}

```

```

13863     setmetatable(sandbox, {__index = _G})
13864     _ENV = sandbox
13865 ]] .. input)()
13866 assert(user_extension,
13867     [[Failed to compile user-defined syntax extension "]]
13868     .. pathname .. [[: ]] .. (err or [[]]))

```

Then, validate the user-defined syntax extension.

```

13869     assert(user_extension.api_version ~= nil,
13870         [[User-defined syntax extension "]] .. pathname
13871         .. [[" does not specify mandatory field "api_version"]])
13872     assert(type(user_extension.api_version) == "number",
13873         [[User-defined syntax extension "]] .. pathname
13874         .. [[" specifies field "api_version" of type "]]
13875         .. type(user_extension.api_version)
13876         .. [[" but "number" was expected]])
13877     assert(user_extension.api_version > 0
13878         and user_extension.api_version
13879         <= metadata.user_extension_api_version,
13880         [[User-defined syntax extension "]] .. pathname
13881         .. [[" uses syntax extension API version "]]
13882         .. user_extension.api_version .. [[" but markdown.lua ]]
13883         .. metadata.version .. [[" uses API version ]]
13884         .. metadata.user_extension_api_version
13885         .. [[" which is incompatible]])
13886
13887     assert(user_extension.grammar_version ~= nil,
13888         [[User-defined syntax extension "]] .. pathname
13889         .. [[" does not specify mandatory field "grammar_version"]])
13890     assert(type(user_extension.grammar_version) == "number",
13891         [[User-defined syntax extension "]] .. pathname
13892         .. [[" specifies field "grammar_version" of type "]]
13893         .. type(user_extension.grammar_version)
13894         .. [[" but "number" was expected]])
13895     assert(user_extension.grammar_version == metadata.grammar_version,
13896         [[User-defined syntax extension "]] .. pathname
13897         .. [[" uses grammar version "]]
13898         .. user_extension.grammar_version
13899         .. [[" but markdown.lua ]] .. metadata.version
13900         .. [[" uses grammar version ]] .. metadata.grammar_version
13901         .. [[" which is incompatible]])
13902
13903     assert(user_extension.finalize_grammar ~= nil,
13904         [[User-defined syntax extension "]] .. pathname
13905         .. [[" does not specify mandatory "finalize_grammar" field]])
13906     assert(type(user_extension.finalize_grammar) == "function",
13907         [[User-defined syntax extension "]] .. pathname
13908         .. [[" specifies field "finalize_grammar" of type "]]

```

```

13909         .. type(user_extension.finalize_grammar)
13910         .. [{" 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.)

```

13911     local extension = {
13912         name = [[user-defined "]] .. pathname .. [{" syntax extension}],
13913         extend_reader = user_extension.finalize_grammar,
13914         extend_writer = function() end,
13915     }
13916     return extension
13917 end)(user_extension_filename)
13918 table.insert(extensions, user_extension)
13919 end

```

Produce a conversion function from markdown to plain TeX.

```

13920 local writer = M.writer.new(options)
13921 local reader = M.reader.new(writer, options)
13922 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`.

```

13923 collectgarbage("collect")

```

Update the singleton cache.

```

13924 if options.singletonCache then
13925     local singletonCacheOptions = {}
13926     for k, v in pairs(options) do
13927         singletonCacheOptions[k] = v
13928     end
13929     setmetatable(singletonCacheOptions,
13930         { __index = function (_, key)
13931             return defaultOptions[key] end })
13932     singletonCache.options = singletonCacheOptions
13933     singletonCache.convert = convert
13934 end

```

Return the conversion function from markdown to plain TeX.

```

13935 return convert
13936 end

13937 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.

```

13938
13939 local input

```

```

13940 if input_filename then
13941   local input_file = assert(io.open(input_filename, "r"),
13942     [[Could not open file ]] .. input_filename .. [[ for reading]])
13943   input = assert(input_file:read("*a"))
13944   assert(input_file:close())
13945 else
13946   input = assert(io.read("*a"))
13947 end
13948

```

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

```

13949 local lfs = require("lfs")
13950 if options.cacheDir and not lfs.isdir(options.cacheDir) then
13951   assert(lfs.mkdir(options["cacheDir"]))
13952 end

```

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

```

13953 local kpse
13954 (function()
13955   local should_initialize = package.loaded.kpse == nil
13956                       or tex.initialize ~= nil
13957   kpse = require("kpse")
13958   if should_initialize then
13959     kpse.set_program_name("luatex")
13960   end
13961 end)()
13962 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.

```

13963 if metadata.version ~= md.metadata.version then
13964   warn("markdown-cli.lua " .. metadata.version .. " used with " ..
13965     "markdown.lua " .. md.metadata.version .. ".")
13966 end
13967
13968 local convert = md.new(options)
13969 local raw_output, flat_output = convert(input, true)
13970 local output
13971 if flat_output == nil then
13972   if options.eagerCache then
13973     warn("markdown.lua has not produced flat output, so I am using " ..
13974       "backwards-compatible raw output instead. This may cause " ..
13975       "the conversion result to be hidden behind '\\input'.")
13976   end
13977   output = raw_output
13978 else
13979   output = flat_output()

```

```

13980 end
13981
13982 if output_filename then
13983   local output_file = assert(io.open(output_filename, "w"),
13984     [[Could not open file "]] .. output_filename .. [[" for writing]])
13985   assert(output_file:write(output))
13986   assert(output_file:close())
13987 else
13988   assert(io.write(output))
13989 end

```

Remove the `options.cacheDir` directory if it is empty.

```

13990 if options.cacheDir then
13991   lfs.rmdir(options.cacheDir)
13992 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

```

13993 \ExplSyntaxOn
13994 \cs_if_free:NT
13995   \markdownInfo
13996   {
13997     \cs_new:Npn
13998       \markdownInfo #1
13999       {
14000         \msg_info:nne
14001           { markdown }
14002           { generic-message }
14003           { #1 }
14004       }
14005   }
14006 \cs_if_free:NT
14007   \markdownWarning
14008   {
14009     \cs_new:Npn
14010       \markdownWarning #1
14011       {
14012         \msg_warning:nne
14013           { markdown }
14014           { generic-message }

```

```

14015         { #1 }
14016     }
14017 }
14018 \cs_if_free:NT
14019   \markdownError
14020   {
14021     \cs_new:Npn
14022       \markdownError #1 #2
14023       {
14024         \msg_error:nnee
14025           { markdown }
14026           { generic-message-with-help-text }
14027           { #1 }
14028           { #2 }
14029       }
14030   }
14031 \msg_new:nnn
14032   { markdown }
14033   { generic-message }
14034   { #1 }
14035 \msg_new:nnnn
14036   { markdown }
14037   { generic-message-with-help-text }
14038   { #1 }
14039   { #2 }
14040 \cs_generate_variant:Nn
14041   \msg_info:nnn
14042   { nne }
14043 \cs_generate_variant:Nn
14044   \msg_warning:nnn
14045   { nne }
14046 \cs_generate_variant:Nn
14047   \msg_error:nnnn
14048   { nnee }
14049 \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.

```

14050 \ExplSyntaxOn
14051 \prop_new:N \g_@@_plain_tex_loaded_themes_linenos_prop
14052 \prop_new:N \g_@@_plain_tex_loaded_themes_versions_prop
14053 \cs_new:Nn
14054   \@@_plain_tex_load_theme:nnn
14055   {

```



```

14056 \prop_get:NnNTF
14057 \g_@@_plain_tex_loaded_themes_linenos_prop
14058 { #1 }
14059 \l_tmpa_tl
14060 {
14061   \prop_get:NnN
14062   \g_@@_plain_tex_loaded_themes_versions_prop
14063   { #1 }
14064   \l_tmpb_tl
14065   \str_if_eq:nVTF
14066   { #2 }
14067   \l_tmpb_tl
14068   {
14069     \msg_warning:nnnVn
14070     { markdown }
14071     { repeatedly-loaded-plain-tex-theme }
14072     { #1 }
14073     \l_tmpa_tl
14074     { #2 }
14075   }
14076   {
14077     \msg_error:nnnnVV
14078     { markdown }
14079     { different-versions-of-plain-tex-theme }
14080     { #1 }
14081     { #2 }
14082     \l_tmpb_tl
14083     \l_tmpa_tl
14084   }
14085 }
14086 {
14087   \prop_gput:Nnx
14088   \g_@@_plain_tex_loaded_themes_linenos_prop
14089   { #1 }
14090   { \tex_the:D \tex_inputlineno:D } % noqa: W200
14091   \prop_gput:Nnn
14092   \g_@@_plain_tex_loaded_themes_versions_prop
14093   { #1 }
14094   { #2 }

```

Load built-in plain TeX themes from the prop `\g_@@_plain_tex_built_in_themes_prop` and from the filesystem otherwise.

```

14095 \prop_if_in:NnTF
14096 \g_@@_plain_tex_built_in_themes_prop
14097 { #1 }
14098 {
14099   \msg_info:nnnn

```

```

14100             { markdown }
14101             { loading-built-in-plain-tex-theme }
14102             { #1 }
14103             { #2 }
14104             \prop_item:Nn
14105             \g_@@_plain_tex_built_in_themes_prop
14106             { #1 }
14107         }
14108     {
14109         \msg_info:nnnn
14110         { markdown }
14111         { loading-plain-tex-theme }
14112         { #1 }
14113         { #2 }
14114         \file_input:n
14115         { markdown theme #3 }
14116     }
14117 }
14118 }
14119 \msg_new:nnn
14120 { markdown }
14121 { loading-plain-tex-theme }
14122 { Loading~version~#2~of~plain~TeX~Markdown~theme~#1 }
14123 \msg_new:nnn
14124 { markdown }
14125 { loading-built-in-plain-tex-theme }
14126 { Loading~version~#2~of~built-in-plain~TeX~Markdown~theme~#1 }
14127 \msg_new:nnn
14128 { markdown }
14129 { repeatedly-loaded-plain-tex-theme }
14130 {
14131     Version~#3~of~plain~TeX~Markdown~theme~#1~was~previously~
14132     loaded~on~line~#2,~not~loading~it~again
14133 }
14134 \msg_new:nnn
14135 { markdown }
14136 { different-versions-of-plain-tex-theme }
14137 {
14138     Tried~to~load~version~#2~of~plain~TeX~Markdown~theme~#1~
14139     but~version~#3~has~already~been~loaded~on~line~#4
14140 }
14141 \cs_generate_variant:Nn
14142 \prop_gput:Nnn
14143 { Nnx }
14144 \cs_gset_eq:NN
14145 \@@_load_theme:nnn
14146 \@@_plain_tex_load_theme:nnn

```

```

14147 \cs_generate_variant:Nn
14148   \@@_load_theme:nnn
14149   { VeV }
14150 \cs_generate_variant:Nn
14151   \msg_error:nnnnnn
14152   { nnnnVV }
14153 \cs_generate_variant:Nn
14154   \msg_warning:nnnnn
14155   { nnnVn }

```

Developers can use the `\markdownLoadPlainTeXTheme` macro to load a corresponding plain TeX theme from within themes for higher-level TeX formats such as L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt.

```

14156 \cs_new:Npn
14157   \markdownLoadPlainTeXTheme
14158   {

```

First, we extract the name of the current theme from the `\g_@@_current_theme_tl` macro.

```

14159     \tl_set:NV
14160       \l_tmpa_tl
14161       \g_@@_current_theme_tl
14162     \tl_reverse:N
14163       \l_tmpa_tl
14164     \tl_set:Ne
14165       \l_tmpb_tl
14166       {
14167         \tl_tail:V
14168         \l_tmpa_tl
14169       }
14170     \tl_reverse:N
14171       \l_tmpb_tl

```

Next, we munge the theme name.

```

14172     \str_set:NV
14173       \l_tmpa_str
14174       \l_tmpb_tl
14175     \str_replace_all:Nnn
14176       \l_tmpa_str
14177       { / }
14178       { _ }

```

Finally, we load the plain TeX theme.

```

14179     \@@_plain_tex_load_theme:VeV
14180     \l_tmpb_tl
14181     { \markdownThemeVersion }
14182     \l_tmpa_str
14183   }

```

```

14184 \cs_generate_variant:Nn
14185   \tl_set:Nn
14186   { Ne }
14187 \cs_generate_variant:Nn
14188   \@@_plain_tex_load_theme:nnn
14189   { VeV }

```

The `witiko/dot` theme nags users that they should use the name `witiko/diagrams@v1` instead.

```

14190 \prop_gput:Nnn
14191   \g_@@_plain_tex_built_in_themes_prop
14192   { witiko / dot }
14193   {
14194     \str_if_eq:enF
14195       { \markdownThemeVersion }
14196       { silent }
14197       {
14198         \markdownWarning
14199         {
14200           The~theme~name~"witiko/dot"~has~been~soft-deprecated.
14201           \iow_newline:
14202           Consider~changing~the~name~to~"witiko/diagrams@v1".
14203         }
14204       }

```

We enable the `fencedCode` Lua option.

```

14205   \markdownSetup { fencedCode }

```

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

```

14206   \cs_set_eq:NN
14207   \@@_dot_previous_definition:nnn
14208   \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:

```

14209   \regex_const:Nn
14210   \c_@@_dot_infostring_regex
14211   { ^dot(\s+(.+))? }
14212   \seq_new:N
14213   \l_@@_dot_matches_seq
14214   \markdownSetup {
14215     rendererPrototypes = {
14216       inputFencedCode = {
14217         \regex_extract_once:NnNTF
14218         \c_@@_dot_infostring_regex
14219         { #2 }

```

```

14220 \l_@@_dot_matches_seq
14221 {
14222   \@@_if_option:nF
14223     { frozenCache }
14224     {
14225       \sys_shell_now:n
14226       {
14227         if~!~test~-e~#1.pdf.source~
14228         ||~!~diff~#1~#1.pdf.source;
14229         then~
14230           dot~-Tpdf~-o~#1.pdf~#1;
14231           cp~#1~#1.pdf.source;
14232         fi
14233       }
14234     }

```

We include the typeset image using the image token renderer:

```

14235 \exp_args:NNne
14236 \exp_last_unbraced:No
14237 \markdownRendererImage
14238 {
14239   { Graphviz~image }
14240   { #1.pdf }
14241   { #1.pdf }
14242 }
14243 {
14244   \seq_item:Nn
14245     \l_@@_dot_matches_seq
14246     { 3 }
14247 }
14248 }

```

If the infostring does not start with `dot ...`, we use the previous definition of the fenced code token renderer prototype:

```

14249 {
14250   \@@_dot_previous_definition:nnn
14251   { #1 }
14252   { #2 }
14253   { #3 }
14254 }
14255 },
14256 },
14257 }
14258 }

```

The theme `witiko/diagrams` loads either the theme `witiko/dot` for version `v1` or the theme `witiko/diagrams/v2` for version `v2`.

```

14259 \prop_gput:Nnn

```

```

14260 \g_@@_plain_tex_built_in_themes_prop
14261 { witiko / diagrams }
14262 {
14263   \str_case:enF
14264     { \markdownThemeVersion }
14265     {
14266       { latest }
14267       {
14268         \markdownWarning
14269         {
14270           Write~"witiko/diagrams@v2"~to~pin~version~"v2"~of~the~
14271           theme~"witiko/diagrams".~This~will~keep~your~documents~
14272           from~suddenly~breaking~when~we~have~released~future~
14273           versions~of~the~theme~with~backwards~incompatible~
14274           syntax~and~behavior.
14275         }
14276         \markdownSetup
14277         {
14278           import = witiko/diagrams/v2,
14279         }
14280       }
14281     { v2 }
14282     {
14283       \markdownSetup
14284       {
14285         import = witiko/diagrams/v2,
14286       }
14287     }
14288     { v1 }
14289     {
14290       \markdownSetup
14291       {
14292         import = witiko/dot@silent,
14293       }
14294     }
14295   }
14296   {
14297     \msg_error:nnnn
14298     { markdown }
14299     { unknown-theme-version }
14300     { witiko/diagrams }
14301     { \markdownThemeVersion }
14302     { v1 }
14303   }
14304 }
14305 \cs_generate_variant:Nn
14306 \msg_error:nnnnn

```

```

14307 { nnnen }
14308 \msg_new:nnnn
14309 { markdown }
14310 { unknown-theme-version }
14311 { Unknown~version~"#2"~of~theme~"#1"~has~been~requested. }
14312 { Known~versions~are:~#3 }

```

Next, we implement the theme [witiko/diagrams/v2](#).

```

14313 \prop_gput:Nnn
14314 \g_@@_plain_tex_built_in_themes_prop
14315 { witiko / diagrams / v2 }
14316 {

```

We enable the `fencedCode` and `fencedCodeAttributes` Lua option.

```

14317 \@@_setup:n
14318 {
14319     fencedCode = true,
14320     fencedCodeAttributes = true,
14321 }

```

Store the previous fenced code token renderer prototype.

```

14322 \cs_set_eq:NN
14323 \@@_diagrams_previous_fenced_code:nnn
14324 \markdownRendererInputFencedCodePrototype

```

Store the caption and the desired format of the diagram.

```

14325 \tl_new:N
14326 \l_@@_diagrams_caption_tl
14327 \tl_new:N
14328 \l_@@_diagrams_format_tl
14329 \tl_set:Nn
14330 \l_@@_diagrams_format_tl
14331 { pdf }
14332 \@@_setup:n
14333 {
14334     rendererPrototypes = {

```

Route attributes on fenced code blocks to the image attribute renderer prototypes.

```

14335     fencedCodeAttributeContextBegin = {
14336         \group_begin:
14337         \markdownRendererImageAttributeContextBegin
14338         \cs_set_eq:NN
14339         \@@_diagrams_previous_key_value:nn
14340         \markdownRendererAttributeKeyValuePrototype
14341         \@@_setup:n
14342         {
14343             rendererPrototypes = {
14344                 attributeKeyValue = {
14345                     \str_case:nnF

```

```

14346         { ##1 }
14347     {
14348         { caption }
14349         {
14350             \tl_set:Nn
14351                 \l_@@_diagrams_caption_tl
14352                 { ##2 }
14353         }
14354         { format }
14355         {
14356             \tl_set:Nn
14357                 \l_@@_diagrams_format_tl
14358                 { ##2 }
14359         }
14360     }
14361     {
14362         \@@_diagrams_previous_key_value:nn
14363         { ##1 }
14364         { ##2 }
14365     }
14366 },
14367 },
14368 }
14369 },
14370 fencedCodeAttributeContextEnd = {
14371     \markdownRendererImageAttributeContextEnd
14372     \group_end:
14373 },
14374 },
14375 }
14376 \cs_new:Nn
14377     \@@_diagrams_render_diagram:nnnn
14378     {
14379         \@@_if_option:nF
14380         { frozenCache }
14381         {
14382             \sys_shell_now:n
14383             {
14384                 if~!~test~-e~#2.source~
14385                 ||~!~diff~#1~#2.source;
14386                 then~
14387                     (#3);
14388                     cp~#1~#2.source;
14389                 fi
14390             }
14391         \exp_args:NNnV
14392         \exp_last_unbraced:No

```



```

14393         \markdownRendererImage
14394         {
14395             { #4 }
14396             { #2 }
14397             { #2 }
14398         }
14399         \l_@@_diagrams_caption_tl
14400     }
14401 }

```

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.

```

14402     \prop_new:N
14403     \g_markdown_diagrams_infostrings_prop

```

If we know a processing function for a given infostring, use it.

```

14404     \@@_setup:n
14405     {
14406         rendererPrototypes = {
14407             inputFencedCode = {
14408                 \prop_get:NnNTF
14409                 \g_markdown_diagrams_infostrings_prop
14410                 { #2 }
14411                 \l_tmpa_tl
14412                 {
14413                     \cs_set:NV
14414                     \@@_diagrams_infostrings_current:n
14415                     \l_tmpa_tl
14416                     \@@_diagrams_infostrings_current:n
14417                     { #1 }
14418                 }

```

Otherwise, use the previous fenced code token renderer prototype.

```

14419         {
14420             \@@_diagrams_previous_fenced_code:nnn
14421             { #1 }
14422             { #2 }
14423             { #3 }
14424         }
14425     },
14426 },
14427 }
14428 \cs_generate_variant:Nn
14429 \cs_set:Nn
14430 { NV }

```

Typeset fenced code with infostring `dot` using the command `dot` from the package `Graphviz`.

```

14431 \cs_set:Nn
14432 \@@_diagrams_infostrings_current:n
14433 {
14434 \@@_diagrams_render_diagram:nnnn
14435 { #1 }
14436 { #1.pdf }
14437 { dot--Tpdf~-o~#1.pdf~#1 }
14438 { Graphviz~image }
14439 }
14440 \@@_tl_set_from_cs:NNn
14441 \l_tmpa_tl
14442 \@@_diagrams_infostrings_current:n
14443 { 1 }
14444 \prop_gput:NnV
14445 \g_markdown_diagrams_infostrings_prop
14446 { dot }
14447 \l_tmpa_tl

```

Typeset fenced code with infostring [mermaid](#) using the command [mmdc](#) from the npm package [@mermaid-js/mermaid-cli](#).

```

14448 \cs_set:Nn
14449 \@@_diagrams_infostrings_current:n
14450 {
14451 \@@_diagrams_render_diagram:nnnn
14452 { #1 }
14453 { #1.pdf }
14454 { mmdc---pdfFit~-i~#1~-o~#1.pdf }
14455 { Mermaid~image }
14456 }
14457 \@@_tl_set_from_cs:NNn
14458 \l_tmpa_tl
14459 \@@_diagrams_infostrings_current:n
14460 { 1 }
14461 \prop_gput:NnV
14462 \g_markdown_diagrams_infostrings_prop
14463 { mermaid }
14464 \l_tmpa_tl

```

Typeset fenced code with infostring [plantuml](#) using the command [plantuml](#) from the package PlantUML.

```

14465 \regex_const:Nn
14466 \c_@@_diagrams_filename_suffix_regex
14467 { \.[^.]*$ }
14468 \cs_set:Nn
14469 \@@_diagrams_infostrings_current:n
14470 {

```

Use the output format provided by the user.

```

14471 \tl_set:Nn
14472   \l_tmpa_tl
14473   { #1 }
14474 \regex_replace_once:NxN
14475   \c_@@_diagrams_filename_suffix_regex
14476   {
14477     .
14478     \tl_use:N
14479       \l_@@_diagrams_format_tl
14480   }
14481   \l_tmpa_tl
14482 \tl_set:Nn
14483   \l_tmpb_tl
14484   { plantuml~-t }
14485 \tl_put_right:NV
14486   \l_tmpb_tl
14487   \l_@@_diagrams_format_tl
14488 \tl_put_right:Nn
14489   \l_tmpb_tl
14490   { ~#1 }

```

For the SVG format, use Inkscape to convert the resulting image to PDF.

```

14491 \str_if_eq:VnT
14492   \l_@@_diagrams_format_tl
14493   { svg }
14494   {
14495     \tl_put_right:Nn
14496       \l_tmpb_tl
14497       { ;~inkscape~ }
14498     \tl_put_right:NV
14499       \l_tmpb_tl
14500       \l_tmpa_tl
14501     \tl_put_right:Nn
14502       \l_tmpb_tl
14503       { ~---export-area-drawing~---export-dpi=300~-o~ }
14504     \tl_set:Nn
14505       \l_tmpa_tl
14506       { #1 }
14507     \regex_replace_once:NnN
14508       \c_@@_diagrams_filename_suffix_regex
14509       { .pdf }
14510       \l_tmpa_tl
14511     \tl_put_right:NV
14512       \l_tmpb_tl
14513       \l_tmpa_tl
14514   }
14515 \@@_diagrams_render_diagram:nVVn
14516   { #1 }

```

```

14517         \l_tmpa_tl
14518         \l_tmpb_tl
14519         { PlantUML~image }
14520     }
14521     \cs_generate_variant:Nn
14522     \@@_diagrams_render_diagram:nnnn
14523     { nVVn }
14524     \cs_generate_variant:Nn
14525     \regex_replace_once:NnN
14526     { NxN }
14527     \@@_tl_set_from_cs:NNn
14528     \l_tmpa_tl
14529     \@@_diagrams_infostrings_current:n
14530     { 1 }
14531     \prop_gput:NnV
14532     \g_markdown_diagrams_infostrings_prop
14533     { plantuml }
14534     \l_tmpa_tl
14535 }

```

We locally change the category code of percent signs, so that we can use them in the shell code:

```

14536 \group_begin:
14537 \char_set_catcode_other:N \%

```

The [witiko/graphicx/http](#) theme stores the previous definition of the image token renderer prototype:

```

14538 \prop_gput:Nnn
14539 \g_@@_plain_tex_built_in_themes_prop
14540 { witiko / graphicx / http }
14541 {
14542     \cs_set_eq:NN
14543     \@@_graphicx_http_previous_definition:nnnn
14544     \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.

```

14545     \int_new:N
14546     \g_@@_graphicx_http_image_number_int
14547     \int_gset:Nn
14548     \g_@@_graphicx_http_image_number_int
14549     { 0 }
14550     \cs_new:Nn
14551     \@@_graphicx_http_filename:
14552     {
14553         \markdownOptionCacheDir
14554         / witiko_graphicx_http .
14555         \int_use:N

```

```

14556         \g_@@_graphicx_http_image_number_int
14557     }

```

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.

```

14558     \cs_new:Nn
14559         \@@_graphicx_http_download:nn
14560     {
14561         wget~--0~#2~#1~
14562         ||~curl~---location~--o~#2~#1~
14563         ||~rm~--f~#2
14564     }

```

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

```

14565     \str_new:N
14566         \l_@@_graphicx_http_filename_str
14567     \ior_new:N
14568         \g_@@_graphicx_http_filename_ior
14569     \markdownSetup {
14570         rendererPrototypes = {
14571             image = {
14572                 \@@_if_option:nF
14573                 { frozenCache }
14574                 {

```

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:

```

14575         \sys_shell_now:e
14576         {
14577             mkdir~~p~" \markdownOptionCacheDir ";
14578             if~printf~'%s'~"#3"~|~grep~--q~--E~'^https?:';
14579             then~

```

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

```

14580             OUTPUT_PREFIX=" \markdownOptionCacheDir ";
14581             OUTPUT_BODY="$(printf~'%s'~'#3'
14582                 |~md5sum~|~cut~-d'~'~-f1)";
14583             OUTPUT_SUFFIX="$(printf~'%s'~'#3'
14584                 |~sed~'s/.*[.]//')";
14585             OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";

```

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

```

14586             if~!~[~--e~"$OUTPUT"~];
14587             then~

```

```

14588         \@@_graphicx_http_download:nn
14589         { '#3' }
14590         { "$OUTPUT" } ;
14591         printf~'%s'~"$OUTPUT"~
14592         >~" \@@_graphicx_http_filename: ";
14593     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:

```

14594         else~
14595         printf~'%s'~'#3'~
14596         >~" \@@_graphicx_http_filename: ";
14597     fi
14598 }
14599 }

```

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

```

14600     \ior_open:Ne
14601     \g_@@_graphicx_http_filename_ior
14602     { \@@_graphicx_http_filename: }
14603     \ior_str_get:NN
14604     \g_@@_graphicx_http_filename_ior
14605     \l_@@_graphicx_http_filename_str
14606     \ior_close:N
14607     \g_@@_graphicx_http_filename_ior
14608     \@@_graphicx_http_previous_definition:nnVn
14609     { #1 }
14610     { #2 }
14611     \l_@@_graphicx_http_filename_str
14612     { #4 }
14613     \int_gincr:N
14614     \g_@@_graphicx_http_image_number_int
14615 }
14616 }
14617 }
14618 \cs_generate_variant:Nn
14619 \ior_open:Nn
14620 { Ne }
14621 \cs_generate_variant:Nn
14622 \@@_graphicx_http_previous_definition:nnnn
14623 { nnVn }
14624 }
14625 \group_end:

```

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

```

14626 \prop_gput:Nnn

```

```

14627 \g_@@_plain_tex_built_in_themes_prop
14628 { witiko / tilde }
14629 {
14630   \markdownSetup {
14631     rendererPrototypes = {
14632       tilde = {~},
14633     },
14634   }
14635 }

```

The themes `witiko/example/foo` and `witiko/example/bar` are supposed to be used in code examples. They don't do anything.

```

14636 \clist_map_inline:nn
14637 { foo, bar }
14638 {
14639   \prop_gput:Nnn
14640   \g_@@_plain_tex_built_in_themes_prop
14641   { witiko / example / #1 }
14642   {
14643     \markdownWarning
14644     {
14645       The~theme~witiko/example/#1~is~supposed~to~be~used~in~code~
14646       examples.~Using~it~in~actual~code~has~no~effect,~except~
14647       this~warning~message,~and~is~usually~a~mistake.
14648     }
14649   }
14650 }
14651 \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.

```

14652 \def\markdownRendererInterblockSeparatorPrototype{\par}%
14653 \def\markdownRendererParagraphSeparatorPrototype{%
14654   \markdownRendererInterblockSeparator}%
14655 \def\markdownRendererHardLineBreakPrototype{\hfil\break}%
14656 \def\markdownRendererSoftLineBreakPrototype{ }%
14657 \let\markdownRendererEllipsisPrototype\dots
14658 \def\markdownRendererNbspPrototype{~}%
14659 \def\markdownRendererLeftBracePrototype{\char`{}\}%
14660 \def\markdownRendererRightBracePrototype{\char`\}\}%
14661 \def\markdownRendererDollarSignPrototype{\char`\$}%
14662 \def\markdownRendererPercentSignPrototype{\char`\}%}%
14663 \def\markdownRendererAmpersandPrototype{\&}%
14664 \def\markdownRendererUnderscorePrototype{\char`\_}%

```

```

14665 \def\markdownRendererHashPrototype{\char`\\}%
14666 \def\markdownRendererCircumflexPrototype{\char`^}%
14667 \def\markdownRendererBackslashPrototype{\char`\\}%
14668 \def\markdownRendererTildePrototype{\char`~}%
14669 \def\markdownRendererPipePrototype{|}%
14670 \def\markdownRendererCodeSpanPrototype#1{\tt#1}%
14671 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
14672 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
14673   \markdownInput{#3}}%
14674 \def\markdownRendererContentBlockOnlineImagePrototype{%
14675   \markdownRendererImage}%
14676 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
14677   \markdownRendererInputFencedCode{#3}{#2}{#2}}%
14678 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
14679 \def\markdownRendererUlBeginPrototype{}%
14680 \def\markdownRendererUlBeginTightPrototype{}%
14681 \def\markdownRendererUlItemPrototype{}%
14682 \def\markdownRendererUlItemEndPrototype{}%
14683 \def\markdownRendererUlEndPrototype{}%
14684 \def\markdownRendererUlEndTightPrototype{}%
14685 \def\markdownRendererOlBeginPrototype{}%
14686 \def\markdownRendererOlBeginTightPrototype{}%
14687 \def\markdownRendererFancyOlBeginPrototype#1#2{%
14688   \markdownRendererOlBegin}%
14689 \def\markdownRendererFancyOlBeginTightPrototype#1#2{%
14690   \markdownRendererOlBeginTight}%
14691 \def\markdownRendererOlItemPrototype{}%
14692 \def\markdownRendererOlItemWithNumberPrototype#1{}%
14693 \def\markdownRendererOlItemEndPrototype{}%
14694 \def\markdownRendererFancyOlItemPrototype{\markdownRendererOlItem}%
14695 \def\markdownRendererFancyOlItemWithNumberPrototype{%
14696   \markdownRendererOlItemWithNumber}%
14697 \def\markdownRendererFancyOlItemEndPrototype{}%
14698 \def\markdownRendererOlEndPrototype{}%
14699 \def\markdownRendererOlEndTightPrototype{}%
14700 \def\markdownRendererFancyOlEndPrototype{\markdownRendererOlEnd}%
14701 \def\markdownRendererFancyOlEndTightPrototype{%
14702   \markdownRendererOlEndTight}%
14703 \def\markdownRendererDlBeginPrototype{}%
14704 \def\markdownRendererDlBeginTightPrototype{}%
14705 \def\markdownRendererDlItemPrototype#1{#1}%
14706 \def\markdownRendererDlItemEndPrototype{}%
14707 \def\markdownRendererDlDefinitionBeginPrototype{}%
14708 \def\markdownRendererDlDefinitionEndPrototype{\par}%
14709 \def\markdownRendererDlEndPrototype{}%
14710 \def\markdownRendererDlEndTightPrototype{}%
14711 \def\markdownRendererEmphasisPrototype#1{\it#1}%

```



```

14712 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}}%
14713 \def\markdownRendererBlockQuoteBeginPrototype{\begingroup\it}%
14714 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
14715 \def\markdownRendererLineBlockBeginPrototype{\begingroup\parindent=0pt}%
14716 \def\markdownRendererLineBlockEndPrototype{\endgroup}%
14717 \def\markdownRendererInputVerbatimPrototype#1{%
14718   \par{\tt\input#1\relax{}}\par}%
14719 \def\markdownRendererInputFencedCodePrototype#1#2#3{%
14720   \markdownRendererInputVerbatim{#1}}%
14721 \def\markdownRendererHeadingOnePrototype#1{#1}%
14722 \def\markdownRendererHeadingTwoPrototype#1{#1}%
14723 \def\markdownRendererHeadingThreePrototype#1{#1}%
14724 \def\markdownRendererHeadingFourPrototype#1{#1}%
14725 \def\markdownRendererHeadingFivePrototype#1{#1}%
14726 \def\markdownRendererHeadingSixPrototype#1{#1}%
14727 \def\markdownRendererThematicBreakPrototype{}%
14728 \def\markdownRendererNotePrototype#1{#1}%
14729 \def\markdownRendererCitePrototype#1{}%
14730 \def\markdownRendererTextCitePrototype#1{}%
14731 \def\markdownRendererTickedBoxPrototype{[X]}%
14732 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
14733 \def\markdownRendererUntickedBoxPrototype{[ ]}%
14734 \def\markdownRendererStrikeThroughPrototype#1{#1}%
14735 \def\markdownRendererSuperscriptPrototype#1{#1}%
14736 \def\markdownRendererSubscriptPrototype#1{#1}%
14737 \def\markdownRendererDisplayMathPrototype#1{$$$#1$$$}%
14738 \def\markdownRendererInlineMathPrototype#1{$$#1$}%
14739 \ExplSyntaxOn
14740 \cs_gset:Npn
14741   \markdownRendererHeaderAttributeContextBeginPrototype
14742   {
14743     \group_begin:
14744     \color_group_begin:
14745   }
14746 \cs_gset:Npn
14747   \markdownRendererHeaderAttributeContextEndPrototype
14748   {
14749     \color_group_end:
14750     \group_end:
14751   }
14752 \cs_gset_eq:NN
14753   \markdownRendererBracketedSpanAttributeContextBeginPrototype
14754   \markdownRendererHeaderAttributeContextBeginPrototype
14755 \cs_gset_eq:NN
14756   \markdownRendererBracketedSpanAttributeContextEndPrototype
14757   \markdownRendererHeaderAttributeContextEndPrototype
14758 \cs_gset_eq:NN

```

```

14759 \markdownRendererFencedDivAttributeContextBeginPrototype
14760 \markdownRendererHeaderAttributeContextBeginPrototype
14761 \cs_gset_eq:NN
14762 \markdownRendererFencedDivAttributeContextEndPrototype
14763 \markdownRendererHeaderAttributeContextEndPrototype
14764 \cs_gset_eq:NN
14765 \markdownRendererFencedCodeAttributeContextBeginPrototype
14766 \markdownRendererHeaderAttributeContextBeginPrototype
14767 \cs_gset_eq:NN
14768 \markdownRendererFencedCodeAttributeContextEndPrototype
14769 \markdownRendererHeaderAttributeContextEndPrototype
14770 \cs_gset:Npn
14771 \markdownRendererReplacementCharacterPrototype
14772 { \codepoint_str_generate:n { fffd } }
14773 \ExplSyntaxOff
14774 \def\markdownRendererSectionBeginPrototype{%
14775 \def\markdownRendererSectionEndPrototype{%
14776 \ExplSyntaxOn
14777 \cs_gset:Npn
14778 \markdownRendererWarningPrototype
14779 #1#2#3#4
14780 {
14781 \tl_set:Nn
14782 \l_tmpa_tl
14783 { #2 }
14784 \tl_if_empty:nF
14785 { #4 }
14786 {
14787 \tl_put_right:Nn
14788 \l_tmpa_tl
14789 { \iow_newline: #4 }
14790 }
14791 \exp_args:NV
14792 \markdownWarning
14793 \l_tmpa_tl
14794 }
14795 \ExplSyntaxOff
14796 \def\markdownRendererErrorPrototype#1#2#3#4{%
14797 \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.

```

14798 \ExplSyntaxOn
14799 \cs_new:Nn

```

```

14800 \@@_plain_tex_default_input_raw_inline:nn
14801 {
14802   \str_case:nn
14803     { #2 }
14804     {
14805       { md } { \markdownInput{#1} }
14806       { tex } { \markdownEscape{#1} \unskip }
14807     }
14808   }
14809 \cs_new:Nn
14810 \@@_plain_tex_default_input_raw_block:nn
14811 {
14812   \str_case:nn
14813     { #2 }
14814     {
14815       { md } { \markdownInput{#1} }
14816       { tex } { \markdownEscape{#1} }
14817     }
14818   }
14819 \cs_gset:Npn
14820 \markdownRendererInputRawInlinePrototype#1#2
14821 {
14822   \@@_plain_tex_default_input_raw_inline:nn
14823     { #1 }
14824     { #2 }
14825   }
14826 \cs_gset:Npn
14827 \markdownRendererInputRawBlockPrototype#1#2
14828 {
14829   \@@_plain_tex_default_input_raw_block:nn
14830     { #1 }
14831     { #2 }
14832   }
14833 \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_t1` 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  $p$ .

```
14834 \ExplSyntaxOn
14835 \seq_new:N \g_@@_jekyll_data_datatypes_seq
14836 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
14837 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
14838 \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.

```
14839 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
14840 \cs_new:Nn \@@_jekyll_data_push_address_segment:n
14841 {
14842   \seq_if_empty:NF
14843     \g_@@_jekyll_data_datatypes_seq
14844     {
14845       \seq_get_right:NN
14846       \g_@@_jekyll_data_datatypes_seq
14847       \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.

```
14848   \str_if_eq:NNTF
14849     \l_tmpa_tl
14850     \c_@@_jekyll_data_sequence_tl
14851     {
14852       \seq_put_right:Nn
14853       \g_@@_jekyll_data_wildcard_absolute_address_seq
14854       { * }
14855     }
14856     {
14857       \seq_put_right:Nn
14858       \g_@@_jekyll_data_wildcard_absolute_address_seq
14859       { #1 }
14860     }
14861   }
14862 }
```

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.

```
14863 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
14864 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
14865 \cs_new:Nn \@@_jekyll_data_concatenate_address:NN
14866 {
14867   \seq_pop_left:NN #1 \l_tmpa_tl
14868   \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
14869   \seq_put_left:NV #1 \l_tmpa_tl
14870 }
14871 \cs_new:Nn \@@_jekyll_data_update_address_tls:
14872 {
14873   \@@_jekyll_data_concatenate_address:NN
14874   \g_@@_jekyll_data_wildcard_absolute_address_seq
14875   \g_@@_jekyll_data_wildcard_absolute_address_tl
14876   \seq_get_right:NN
14877   \g_@@_jekyll_data_wildcard_absolute_address_seq
14878   \g_@@_jekyll_data_wildcard_relative_address_tl
14879 }
```

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.

```
14880 \cs_new:Nn \@@_jekyll_data_push:nN
14881 {
14882   \@@_jekyll_data_push_address_segment:n
14883   { #1 }
```

```

14884     \seq_put_right:NV
14885     \g_@@_jekyll_data_datatypes_seq
14886     #2
14887     \@@_jekyll_data_update_address_tls:
14888   }
14889   \cs_new:Nn \@@_jekyll_data_pop:
14890   {
14891     \seq_pop_right:NN
14892     \g_@@_jekyll_data_wildcard_absolute_address_seq
14893     \l_tmpa_tl
14894     \seq_pop_right:NN
14895     \g_@@_jekyll_data_datatypes_seq
14896     \l_tmpa_tl
14897     \@@_jekyll_data_update_address_tls:
14898   }

```

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.

```

14899   \cs_new:Nn \@@_jekyll_data_set_keyval_known:nn
14900   {
14901     \keys_set_known:nn
14902     { markdown/jekyllData }
14903     { { #1 } = { #2 } }
14904   }
14905   \cs_generate_variant:Nn
14906   \@@_jekyll_data_set_keyval_known:nn
14907   { Vn }
14908   \cs_new:Nn \@@_jekyll_data_set_keyvals_known:nn
14909   {
14910     \@@_jekyll_data_push:nN
14911     { #1 }
14912     \c_@@_jekyll_data_scalar_tl
14913     \@@_jekyll_data_set_keyval_known:Vn
14914     \g_@@_jekyll_data_wildcard_absolute_address_tl
14915     { #2 }
14916     \@@_jekyll_data_set_keyval_known:Vn
14917     \g_@@_jekyll_data_wildcard_relative_address_tl
14918     { #2 }
14919     \@@_jekyll_data_pop:
14920   }

```

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

```

14921   \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
14922     \@@_jekyll_data_push:nN
14923     { #1 }
14924     \c_@@_jekyll_data_sequence_tl

```

```

14925 }
14926 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
14927   \@@_jekyll_data_push:nn
14928     { #1 }
14929     \c_@@_jekyll_data_mapping_tl
14930 }
14931 \def\markdownRendererJekyllDataSequenceEndPrototype{
14932   \@@_jekyll_data_pop:
14933 }
14934 \def\markdownRendererJekyllDataMappingEndPrototype{
14935   \@@_jekyll_data_pop:
14936 }
14937 \def\markdownRendererJekyllDataBooleanPrototype#1#2{
14938   \@@_jekyll_data_set_keyvals_known:nn
14939     { #1 }
14940     { #2 }
14941 }
14942 \def\markdownRendererJekyllDataEmptyPrototype#1{}
14943 \def\markdownRendererJekyllDataNumberPrototype#1#2{
14944   \@@_jekyll_data_set_keyvals_known:nn
14945     { #1 }
14946     { #2 }
14947 }

```

We will process all string scalar values assuming that they may contain markdown markup and are intended for typesetting.

```

14948 \def\markdownRendererJekyllDataProgrammaticStringPrototype#1#2{}
14949 \def\markdownRendererJekyllDataTypographicStringPrototype#1#2{
14950   \@@_jekyll_data_set_keyvals_known:nn
14951     { #1 }
14952     { #2 }
14953 }
14954 \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 `jekyllDataKeyValue=<module>`. See also Section 2.2.6.1.

```

14955 \ExplSyntaxOn
14956 \@@_with_various_cases:nn
14957   { jekyllDataKeyValue }
14958   {
14959     \keys_define:nn
14960       { markdown/options }
14961       {
14962         #1 .code:n = {

```

```

14963         \@@_route_jekyll_data_to_key_values:n
14964         { ##1 }
14965     },

```

When no *<module>* has been provided, assume that the YAML metadata specify absolute paths to key-values.

```

14966         #1 .default:n = { },
14967     }
14968 }
14969 \seq_new:N
14970   \l_@@_jekyll_data_current_position_seq
14971 \tl_new:N
14972   \l_@@_jekyll_data_current_position_tl
14973 \cs_new:Nn
14974   \@@_route_jekyll_data_to_key_values:n
14975   {
14976     \markdownSetup
14977     {
14978       renderers = {
14979         jekyllData(Sequence|Mapping)Begin = {
14980           \bool_lazy_and:nnTF
14981             {
14982               \seq_if_empty_p:N
14983                 \l_@@_jekyll_data_current_position_seq
14984             }
14985             {
14986               \str_if_eq_p:nn
14987                 { ##1 }
14988                 { null }
14989             }
14990             {
14991               \tl_if_empty:nF
14992                 { #1 }
14993                 {
14994                   \seq_put_right:Nn
14995                     \l_@@_jekyll_data_current_position_seq
14996                     { #1 }
14997                 }
14998             }
14999             {
15000               \seq_put_right:Nn
15001                 \l_@@_jekyll_data_current_position_seq
15002                 { ##1 }
15003             }
15004         },
15005         jekyllData(Sequence|Mapping)End = {
15006           \seq_pop_right:NN

```



```

15007         \l_@@_jekyll_data_current_position_seq
15008         \l_tmpa_tl
15009     },

```

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`.

```

15010     jekyllDataBoolean = {
15011         \tl_set:Nx
15012         \l_@@_jekyll_data_current_position_tl
15013         {
15014             \seq_use:Nn
15015             \l_@@_jekyll_data_current_position_seq
15016             { / }
15017         }
15018     \keys_if_exist:VnTF
15019     \l_@@_jekyll_data_current_position_tl
15020     { ##1 / boolean }
15021     {
15022         \@@_keys_set:xn
15023         {
15024             \tl_use:N
15025             \l_@@_jekyll_data_current_position_tl
15026             / ##1 / boolean
15027         }
15028         { ##2 }
15029     }
15030     {
15031         \@@_keys_set:xn
15032         {
15033             \tl_use:N
15034             \l_@@_jekyll_data_current_position_tl
15035             / ##1
15036         }
15037         { ##2 }
15038     }
15039 },
15040     jekyllDataNumber = {
15041         \tl_set:Nx
15042         \l_@@_jekyll_data_current_position_tl
15043         {
15044             \seq_use:Nn
15045             \l_@@_jekyll_data_current_position_seq
15046             { / }
15047         }
15048     \keys_if_exist:VnTF

```

```

15049         \l_@@_jekyll_data_current_position_tl
15050     { ##1 / number }
15051     {
15052         \@@_keys_set:xn
15053         {
15054             \tl_use:N
15055                 \l_@@_jekyll_data_current_position_tl
15056             / ##1 / number
15057         }
15058         { ##2 }
15059     }
15060     {
15061         \@@_keys_set:xn
15062         {
15063             \tl_use:N
15064                 \l_@@_jekyll_data_current_position_tl
15065             / ##1
15066         }
15067         { ##2 }
15068     }
15069 },

```

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`.

```

15070     jekyllDataEmpty = {
15071         \tl_set:Nx
15072             \l_@@_jekyll_data_current_position_tl
15073         {
15074             \seq_use:Nn
15075                 \l_@@_jekyll_data_current_position_seq
15076             { / }
15077         }
15078         \keys_if_exist:VnTF
15079             \l_@@_jekyll_data_current_position_tl
15080         { ##1 / empty }
15081         {
15082             \keys_set:xn
15083             {
15084                 \tl_use:N
15085                     \l_@@_jekyll_data_current_position_tl
15086                 / ##1
15087             }
15088             { empty }
15089         }
15090     {
15091         \keys_set:Vn

```

```

15092         \l_@@_jekyll_data_current_position_tl
15093         { ##1 }
15094     }
15095 },

```

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.

```

15096     jekyllDataTypographicString = {
15097         \tl_set:Nx
15098         \l_@@_jekyll_data_current_position_tl
15099         {
15100             \seq_use:Nn
15101             \l_@@_jekyll_data_current_position_seq
15102             { / }
15103         }
15104         \keys_if_exist:VnTF
15105         \l_@@_jekyll_data_current_position_tl
15106         { ##1 / typographicString }
15107         {
15108             \@@_keys_set:xn
15109             {
15110                 \tl_use:N
15111                 \l_@@_jekyll_data_current_position_tl
15112                 / ##1 / typographicString
15113             }
15114             { ##2 }
15115         }
15116         {
15117             \keys_if_exist:VnTF
15118             \l_@@_jekyll_data_current_position_tl
15119             { ##1 / programmaticString }
15120             {
15121                 \@@_keys_set_known:xn
15122                 {
15123                     \tl_use:N
15124                     \l_@@_jekyll_data_current_position_tl
15125                     / ##1
15126                 }
15127                 { ##2 }

```

```

15128         }
15129     {
15130         \@@_keys_set:xn
15131     {
15132         \tl_use:N
15133         \l_@@_jekyll_data_current_position_tl
15134         / ##1
15135     }
15136     { ##2 }
15137 }
15138 }
15139 },
15140 jekyllDataProgrammaticString = {
15141     \tl_set:Nx
15142     \l_@@_jekyll_data_current_position_tl
15143     {
15144         \seq_use:Nn
15145         \l_@@_jekyll_data_current_position_seq
15146         { / }
15147     }
15148     \keys_if_exist:VnT
15149     \l_@@_jekyll_data_current_position_tl
15150     { ##1 / programmaticString }
15151     {
15152         \@@_keys_set:xn
15153     {
15154         \tl_use:N
15155         \l_@@_jekyll_data_current_position_tl
15156         / ##1 / programmaticString
15157     }
15158     { ##2 }
15159     }
15160     },
15161 },
15162 }
15163 }
15164 \cs_new:Nn
15165 \@@_keys_set:nn
15166 {
15167     \keys_set:nn
15168     { }
15169     { { #1 } = { #2 } }
15170 }
15171 \cs_new:Nn
15172 \@@_keys_set_known:nn
15173 {
15174     \keys_set_known:nn

```

```

15175      { }
15176      { { #1 } = { #2 } }
15177    }
15178    \cs_generate_variant:Nn
15179      \@@_keys_set:nn
15180      { xn }
15181    \cs_generate_variant:Nn
15182      \@@_keys_set_known:nn
15183      { xn }
15184    \cs_generate_variant:Nn
15185      \keys_set:nn
15186      { xn, Vn }
15187    \prg_generate_conditional_variant:Nnn
15188      \keys_if_exist:nn
15189      { Vn }
15190      { T, TF }
15191    \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).

```

15192    \ExplSyntaxOn
15193    \str_if_eq:VVT
15194      \c_@@_top_layer_tl
15195      \c_@@_option_layer_plain_tex_tl
15196      {
15197        \use:c
15198          { ExplSyntaxOff }
15199        \@@_if_option:nF
15200          { noDefaults }
15201        {
15202          \@@_if_option:nTF
15203            { experimental }
15204            {
15205              \@@_setup:n
15206                { theme = witiko/markdown/defaults@experimental }
15207            }
15208          {
15209            \@@_setup:n
15210              { theme = witiko/markdown/defaults }
15211          }
15212        }
15213        \use:c
15214          { ExplSyntaxOn }
15215      }
15216    \ExplSyntaxOff

```

### 3.2.4 Lua Snippets

After the `\markdownPrepareLuaOptions` macro has been fully expanded, the `\markdownLuaOptions` macro will expand to a Lua table that contains the plain TeX options (see Section 2.2.2) in a format recognized by Lua (see Section 2.1.3).

```
15217 \ExplSyntaxOn
15218 \tl_new:N \g_@@_formatted_lua_options_tl
15219 \cs_new:Nn \@@_format_lua_options:
15220 {
15221   \tl_gclear:N
15222     \g_@@_formatted_lua_options_tl
15223   \seq_map_function:NN
15224     \g_@@_lua_options_seq
15225     \@@_format_lua_option:n
15226 }
15227 \cs_new:Nn \@@_format_lua_option:n
15228 {
15229   \@@_typecheck_option:n
15230     { #1 }
15231   \@@_get_option_type:nN
15232     { #1 }
15233   \l_tmpa_tl
15234   \bool_case_true:nF
15235     {
15236       {
15237         \str_if_eq_p:VV
15238           \l_tmpa_tl
15239           \c_@@_option_type_boolean_tl ||
15240         \str_if_eq_p:VV
15241           \l_tmpa_tl
15242           \c_@@_option_type_number_tl ||
15243         \str_if_eq_p:VV
15244           \l_tmpa_tl
15245           \c_@@_option_type_counter_tl
15246       }
15247       {
15248         \@@_get_option_value:nN
15249           { #1 }
15250         \l_tmpa_tl
15251         \tl_gput_right:Nx
15252           \g_@@_formatted_lua_options_tl
15253           { #1~== \l_tmpa_tl ,~ }
15254       }
15255     }
15256   \str_if_eq_p:VV
15257     \l_tmpa_tl
15258     \c_@@_option_type_clist_tl
```

```

15259     }
15260     {
15261         \@_get_option_value:nN
15262         { #1 }
15263         \l_tmpa_tl
15264         \tl_gput_right:Nx
15265         \g_@@_formatted_lua_options_tl
15266         { #1~::~\c_left_brace_str }
15267         \clist_map_inline:Vn
15268         \l_tmpa_tl
15269         {
15270             \@_lua_escape:xN
15271             { ##1 }
15272             \l_tmpb_tl
15273             \tl_gput_right:Nn
15274             \g_@@_formatted_lua_options_tl
15275             { " }
15276             \tl_gput_right:NV
15277             \g_@@_formatted_lua_options_tl
15278             \l_tmpb_tl
15279             \tl_gput_right:Nn
15280             \g_@@_formatted_lua_options_tl
15281             { " ,~ }
15282         }
15283         \tl_gput_right:Nx
15284         \g_@@_formatted_lua_options_tl
15285         { \c_right_brace_str ,~ }
15286     }
15287 }
15288 {
15289     \@_get_option_value:nN
15290     { #1 }
15291     \l_tmpa_tl
15292     \@_lua_escape:xN
15293     { \l_tmpa_tl }
15294     \l_tmpb_tl
15295     \tl_gput_right:Nn
15296     \g_@@_formatted_lua_options_tl
15297     { #1~::~ " }
15298     \tl_gput_right:NV
15299     \g_@@_formatted_lua_options_tl
15300     \l_tmpb_tl
15301     \tl_gput_right:Nn
15302     \g_@@_formatted_lua_options_tl
15303     { " ,~ }
15304 }
15305 }

```

```

15306 \cs_generate_variant:Nn
15307   \clist_map_inline:nn
15308   { Vn }
15309 \let
15310   \markdownPrepareLuaOptions
15311   \@@_format_lua_options:
15312 \def
15313   \markdownLuaOptions
15314   {
15315     {
15316       \g_@@_formatted_lua_options_tl
15317     }
15318   }
15319 \sys_if_engine luatex:TF
15320 {
15321   \cs_new:Nn
15322     \@@_lua_escape:nN
15323     {
15324       \tl_set:Nx
15325         #2
15326         {
15327           \lua_escape:n
15328             { #1 }
15329         }
15330     }
15331 }
15332 {
15333   \regex_const:Nn
15334     \c_@@_lua_escape_regex
15335     { [\\"] }
15336   \cs_new:Nn
15337     \@@_lua_escape:nN
15338     {
15339       \tl_set:Nn
15340         #2
15341         { #1 }
15342       \regex_replace_all:NnN
15343         \c_@@_lua_escape_regex
15344         { \u { c_backslash_str } \0 }
15345         #2
15346     }
15347 }
15348 \cs_generate_variant:Nn
15349   \@@_lua_escape:nN
15350   { 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.

```

15351 \tl_new:N
15352   \markdownInputFilename
15353 \cs_new:Npn
15354   \markdownPrepareInputFilename
15355   #1
15356   {
15357     \@@_lua_escape:xN
15358     { #1 }
15359     \markdownInputFilename
15360   \tl_gset:Nx
15361     \markdownInputFilename
15362     { " \markdownInputFilename " }
15363   }

```

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.

```

15364 \cs_new:Npn
15365   \markdownPrepare
15366   {

```

First, ensure that the `cacheDir` directory exists.

```

15367     local~lfs = require("lfs")
15368     local~options = \markdownLuaOptions
15369     if~not~lfs.isdir(options.cacheDir) then~
15370       assert(lfs.mkdir(options.cacheDir))
15371     end~

```

Next, load the `markdown` module and create a converter function using the plain T<sub>E</sub>X options, which were serialized to a Lua table via the `\markdownLuaOptions` macro.

```

15372     local~md = require("markdown")
15373     local~convert = md.new(options)
15374   }

```

The `\markdownConvert` macro contains the Lua code that is executed during the conversion from markdown to plain T<sub>E</sub>X. It opens the input file, converts it, and prints the conversion result.

```

15375 \cs_new:Npn
15376   \markdownConvert
15377   {
15378     local~filename = \markdownInputFilename
15379     local~file = assert(io.open(filename, "r"),
15380       [[Could~not~open~file~]] .. filename .. [[~for~reading]])
15381     local~input = assert(file:read("*a"))
15382     assert(file:close())
15383     print(convert(input))

```

```

15384 }
15385 \ExplSyntaxOff

```

The `\markdownCleanup` macro contains the Lua code that is executed after any conversion from markdown to plain TeX.

```

15386 \def\markdownCleanup{%
Remove the options.cacheDir directory if it is empty.
15387   if options.cacheDir then
15388     lfs.rmdir(options.cacheDir)
15389   end
15390 }%

```

### 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.

```

15391 \csname newread\endcsname\markdownInputFileStream
15392 \csname newwrite\endcsname\markdownOutputFileStream

```

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

```

15393 \begingroup
15394   \catcode\^^I=12%
15395   \gdef\markdownReadAndConvertTab{^^I}%
15396 \endgroup

```

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

```

15397 \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.

```

15398   \catcode\^^M=13%
15399   \catcode\^^I=13%
15400   \catcode|=0%
15401   \catcode\=12%
15402   |catcode@=14%
15403   |catcode%=12@
15404   |gdef|markdownReadAndConvert#1#2{@
15405     |begingroup@

```

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

```

15406     |markdownIfOption{frozenCache}{-}{@

```

```

15407      |immediate|openout|markdownOutputFileStream@
15408      |markdownOptionInputTempFileName|relax@
15409      |markdownInfo{@
15410      Buffering block-level markdown input into the temporary @
15411      input file "|markdownOptionInputTempFileName" and scanning @
15412      for the closing token sequence "#1"}@
15413      }@

```

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.

```

15414      |def|do##1{|catcode`##1=12}|dospecials@
15415      |catcode`| =12@
15416      |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 ( $\sim$ M) are produced.

```

15417      |def|markdownReadAndConvertStripPercentSign##1{@
15418      |markdownIfOption{stripPercentSigns}{@
15419      |if##1%@
15420      |expandafter|expandafter|expandafter@
15421      |markdownReadAndConvertProcessLine@
15422      |else@
15423      |expandafter|expandafter|expandafter@
15424      |markdownReadAndConvertProcessLine@
15425      |expandafter|expandafter|expandafter##1@
15426      |fi@
15427      }{@
15428      |expandafter@
15429      |markdownReadAndConvertProcessLine@
15430      |expandafter##1@
15431      }@
15432      }@

```

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 ( $\sim$ M) are produced.

```

15433      |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.

```

15434      |ifx|relax##3|relax@
15435      |markdownIfOption{frozenCache}{-}{-}@

```

```

15436         |immediate|write|markdownOutputStream{##1}@
15437     }@
15438     |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.

```

15439         |def^^M{@
15440             |markdownInfo{The ending token sequence was found}@
15441             |markdownIfOption{frozenCache}{-}{@
15442                 |immediate|closeout|markdownOutputStream@
15443             }@
15444         |endgroup@
15445         |markdownInput{@
15446             |markdownOptionOutputDir@
15447             /|markdownOptionInputTempFileName@
15448         }@
15449         #2}@
15450     |fi@

```

Repeat with the next line.

```

15451     ^^M}@

```

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

```

15452     |catcode`|^I=13@
15453     |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.

```

15454     |catcode`|^M=13@
15455     |def^^M##1^^M{@
15456         |def^^M###1^^M{@
15457             |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
15458         ^^M}@
15459     ^^M}@

```

Reset the character categories back to the former state.

```

15460 |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.

```

15461 \ExplSyntaxOn
15462 \cs_new:Npn
15463     \markdownLuaExecute
15464     #1
15465     {

```

```

15466 \int_compare:nNtT
15467 { \g_luabridge_method_int }
15468 =
15469 { \c_luabridge_method_shell_int }
15470 {
15471   \sys_if_shell_unrestricted:F
15472   {
15473     \sys_if_shell:TF
15474     {
15475       \msg_error:nn
15476       { markdown }
15477       { restricted-shell-access }
15478     }
15479     {
15480       \msg_error:nn
15481       { markdown }
15482       { disabled-shell-access }
15483     }
15484   }
15485 }
15486 \str_gset:NV
15487   \g_luabridge_output_dirname_str
15488   \markdownOptionOutputDir
15489 \luabridge_now:e
15490 { #1 }
15491 }
15492 \cs_generate_variant:Nn
15493   \msg_new:nnnn
15494   { nnnV }
15495 \tl_set:Nn
15496   \l_tmpa_tl
15497   {
15498     You-may~need~to~run~TeX~with~the~--shell-escape~or~the~
15499     --enable-write18~flag,~or~write~shell_escape=t~in~the~
15500     texmf.cnf~file.
15501   }
15502 \msg_new:nnnV
15503   { markdown }
15504   { restricted-shell-access }
15505   { Shell~escape~is~restricted }
15506   \l_tmpa_tl
15507 \msg_new:nnnV
15508   { markdown }
15509   { disabled-shell-access }
15510   { Shell~escape~is~disabled }
15511   \l_tmpa_tl
15512 \ExplSyntaxOff

```

### 3.2.6 Buffering Inline Markdown Input

This section describes the implementation of the macro `\markinline`.

```
15513 \ExplSyntaxOn
15514 \tl_new:N
15515   \g_@@_after_markinline_tl
15516 \tl_gset:Nn
15517   \g_@@_after_markinline_tl
15518   { \unskip }
15519 \cs_new:Npn
15520   \markinline
15521   {
```

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 markdown text as T<sub>E</sub>X code.

```
15522   \group_begin:
15523   \cctab_select:N
15524     \c_other_cctab
```

Unless we are reading markdown documents from the frozen cache, open the file `inputTempFileName` for writing.

```
15525   \@@_if_option:nF
15526     { frozenCache }
15527     {
15528       \immediate
15529       \openout
15530         \markdownOutputFileStream
15531         \markdownOptionInputTempFileName
15532         \relax
15533       \msg_info:nne
15534         { markdown }
15535         { buffering-markinline }
15536         { \markdownOptionInputTempFileName }
15537     }
```

Peek ahead and extract the inline markdown text.

```
15538   \peek_regex_replace_once:nnF
15539     { { (.*) } }
15540     {
```

Unless we are reading markdown documents from the frozen cache, store the text in the file `inputTempFileName` and close it.

```
15541     \c { @@_if_option:nF }
15542     \cB { frozenCache \cE }
15543     \cB {
15544       \c { immediate }
15545       \c { write }
15546       \c { markdownOutputFileStream }
```

```

15547         \cB { \1 \cE }
15548         \c { immediate }
15549         \c { closeout }
15550         \c { markdownOutputFileStream }
15551         \cE }

```

Reset the category codes and `\input` the result of the conversion.

```

15552         \c { group_end: }
15553         \c { group_begin: }
15554         \c { @@_setup:n }
15555         \cB { contentLevel = inline \cE }
15556         \c { markdownInput }
15557         \cB {
15558             \c { markdownOptionOutputDir } /
15559             \c { markdownOptionInputTempFileName }
15560         \cE }
15561         \c { group_end: }
15562         \c { tl_use:N }
15563         \c { g_@@_after_markinline_tl }
15564     }
15565     {
15566         \msg_error:nn
15567         { markdown }
15568         { markinline-peek-failure }
15569         \group_end:
15570         \tl_use:N
15571         \g_@@_after_markinline_tl
15572     }
15573 }
15574 \msg_new:nnn
15575 { markdown }
15576 { buffering-markinline }
15577 { Buffering~inline~markdown~input~into~
15578   the~temporary~input~file~"#1". }
15579 \msg_new:nnnn
15580 { markdown }
15581 { markinline-peek-failure }
15582 { Use-of~\iow_char:N \\ markinline~doesn't~match~its~definition }
15583 { The~macro~should~be~followed~by~inline~
15584   markdown~text~in~curly~braces }
15585 \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.

```

15586 \ExplSyntaxOn
15587 \cs_new:Npn
15588   \markdownInput
15589   #1
15590   {
15591     \@@_if_option:nTF
15592       { frozenCache }
15593       {
15594         \markdownInputRaw
15595         { #1 }
15596       }
15597   }

```

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`.

```

15598     \tl_set:Nx
15599     \l_tmpa_tl
15600     { #1 }
15601     \file_get_full_name:VNTF
15602     \l_tmpa_tl
15603     \l_tmpb_tl
15604     {
15605       \exp_args:NV
15606       \markdownInputRaw
15607       \l_tmpb_tl
15608     }
15609     {
15610       \msg_error:nnV
15611       { markdown }
15612       { markdown-file-does-not-exist }
15613       \l_tmpa_tl
15614     }
15615   }
15616 }
15617 \msg_new:nnn
15618 { markdown }
15619 { markdown-file-does-not-exist }
15620 {
15621   Markdown~file~#1~does~not~exist
15622 }
15623 \ExplSyntaxOff
15624 \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.



```

15625 \catcode`|=0%
15626 \catcode`\|=12%
15627 \catcode`|&=6%
15628 |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.

```

15629 |begingroup
15630 |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.

```

15631 |catcode`|#=12

```

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

```

15632 |markdownIfOption{frozenCache}{%
15633 |ifnum|markdownOptionFrozenCacheCounter=0|relax
15634 |markdownInfo{Reading frozen cache from
15635 "||markdownOptionFrozenCacheFileName"}%
15636 |input|markdownOptionFrozenCacheFileName|relax
15637 |fi
15638 |markdownInfo{Including markdown document number
15639 "|the|markdownOptionFrozenCacheCounter" from frozen cache}%
15640 |csname markdownFrozenCache%
15641 |the|markdownOptionFrozenCacheCounter|endcsname
15642 |global|advance|markdownOptionFrozenCacheCounter by 1|relax
15643 }{%
15644 |markdownInfo{Including markdown document "&1"}%

```

Attempt to open the markdown document to record it in the `.log` and `.fls` files. This allows external programs such as L<sup>A</sup>T<sub>E</sub>X Mk to track changes to the markdown document.

```

15645 |openin|markdownInputFileStream{&1}%
15646 |closein|markdownInputFileStream
15647 |markdownPrepareLuaOptions
15648 |markdownPrepareInputFilename{&1}%
15649 |markdownLuaExecute{%
15650 |markdownPrepare
15651 |markdownConvert
15652 |markdownCleanup}%

```

If we are finalizing the frozen cache, increment `frozenCacheCounter`.

```

15653 |markdownIfOption{finalizeCache}{%
15654 |global|advance|markdownOptionFrozenCacheCounter by 1|relax}{}%
15655 }%

```

```

15656 |endgroup
15657 }%
15658 |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.

```

15659 \gdef\markdownEscape#1{%
15660   \catcode`\%=14\relax
15661   \catcode`\#=6\relax
15662   \input #1\relax
15663   \catcode`\%=12\relax
15664   \catcode`\#=12\relax
15665 }%

```

### 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 [19, Section 9]. As a consequence, we can directly reuse the existing plain  $\text{\TeX}$  implementation.

```

15666 \def\markdownVersionSpace{ }%
15667 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
15668   \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  $\text{\LaTeX}$  interface (see Section 2.3.3).

```

15669 \ExplSyntaxOn
15670 \cs_gset_eq:NN
15671   \markinlinePlainTeX
15672   \markinline
15673 \cs_gset:Npn
15674   \markinline
15675   {
15676     \peek_regex_replace_once:nn
15677       { ( \[ (.*) \] ) ? }
15678     {

```

Apply the options locally.

```

15679       \c { group_begin: }
15680       \c { @@_setup:n }
15681       \cB { \2 \cE }

```

```

15682         \c { tl_put_right:Nn }
15683         \c { g_@@_after_markinline_tl }
15684         \cB { \c { group_end: } \cE }
15685         \c { markinlinePlainTeX }
15686     }
15687 }
15688 \ExplSyntaxOff

```

The `\markdownInputPlainTeX` macro is used to store the original plain  $\TeX$  implementation of the `\yamlInput` macro. The `\markdownInput` and `\yamlInput` macros are then redefined to accept an optional argument with options recognized by the  $\LaTeX$  interface (see Section 2.3.3).

```

15689 \let\markdownInputPlainTeX\markdownInput
15690 \renewcommand\markdownInput[2] [] {%
15691     \begingroup
15692         \markdownSetup{#1}%
15693         \markdownInputPlainTeX{#2}%
15694     \endgroup}%
15695 \renewcommand\yamlInput[2] [] {%
15696     \begingroup
15697         \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData, #1}%
15698         \markdownInputPlainTeX{#2}%
15699     \endgroup}%

```

The `markdown`, `markdown*`, and `yaml`  $\LaTeX$  environments are implemented using the `\markdownReadAndConvert` macro.

```

15700 \ExplSyntaxOn
15701 \renewenvironment
15702 { markdown }
15703 {

```

In our implementation of the `markdown`  $\LaTeX$  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  $\LaTeX$  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`  $\LaTeX$  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 TeX via the `\endlinechar` plain TeX macro and it needs to have the correct category code, so that `\markdownReadAndConvert` processes it correctly.

```
15704     \group_begin:
15705     \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).

```
15706     \char_set_catcode_letter:n { 35 }
```

After we have matched the opening `[` that begins the optional argument, we accept carriage returns as well.

```
15707     \peek_regex_replace_once:nnF
15708     { \ *[\r*([~]*)\][^\r]* }
15709     {
```

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.

```
15710         \c { group_end: }
15711         \c { tl_set_rescan:Nnn } \c { l_tmpa_tl } { } { \1 }
```

Then, we pass the retokenized content to the `\markdownSetup` macro.

```
15712         \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 L<sup>A</sup>T<sub>E</sub>X environment.

We also make provision for using the `\markdown` command as a part of a different L<sup>A</sup>T<sub>E</sub>X environment as follows:

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

```
15713         \c { exp_args:NV }
15714         \c { markdownReadAndConvert@ }
15715         \c { @currenvir }
15716     }
15717     {
15718         \group_end:
```

```

15719         \exp_args:NV
15720         \markdownReadAndConvert@
15721         \@currenvir
15722     }
15723 }
15724 { \markdownEnd }
15725 \renewenvironment
15726 { markdown* }
15727 [ 1 ]
15728 {
15729     \@@_if_option:nTF
15730     { experimental }
15731     {
15732         \msg_error:nn
15733         { markdown }
15734         { latex-markdown-star-deprecated }
15735     }
15736     {
15737         \msg_warning:nn
15738         { markdown }
15739         { latex-markdown-star-deprecated }
15740     }
15741     \@@_setup:n
15742     { #1 }
15743     \markdownReadAndConvert@
15744     { markdown* }
15745 }
15746 { \markdownEnd }
15747 \renewenvironment
15748 { yaml }
15749 {
15750     \group_begin:
15751     \yamlSetup
15752     { jekyllData, expectJekyllData, ensureJekyllData }
15753     \markdown
15754 }
15755 { \yamlEnd }
15756 \msg_new:nnn
15757 { markdown }
15758 { latex-markdown-star-deprecated }
15759 {
15760     The~markdown*~LaTeX~environment~has~been~deprecated~and~will~
15761     be~removed~in~the~next~major~version~of~the~Markdown~package.
15762 }
15763 \cs_generate_variant:Nn % noqa: w402
15764 \@@_setup:n
15765 { V }

```

```

15766 \ExplSyntaxOff
15767 \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*.

```

15768 \catcode`\|=0\catcode`\<=1\catcode`\>=2%
15769 \catcode`\=12\catcode`\{=12\catcode`\}=12%
15770 |gdef|markdownReadAndConvert@#1<%
15771 |markdownReadAndConvert<\end{#1}>%
15772 <|end<#1>>>%
15773 |endgroup

```

### 3.3.2 Themes

This section overrides the plain  $\TeX$  implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in  $\LaTeX$  themes provided with the Markdown package.

```

15774 \ExplSyntaxOn
15775 \prop_new:N \g_@@_latex_loaded_themes_linenos_prop
15776 \prop_new:N \g_@@_latex_loaded_themes_versions_prop
15777 \cs_gset:Nn
15778 \@@_load_theme:nnn
15779 {

```

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.

```

15780 \ifmarkdownLaTeXLoaded
15781 \ifx\@onlypreamble\@notprerr

```

If both conditions are true, end with an error, since we cannot load  $\LaTeX$  themes after the preamble.

```

15782 \bool_if:nTF
15783 {
15784 \bool_lazy_or_p:nn
15785 {
15786 \prop_if_in_p:Nn
15787 \g_@@_latex_built_in_themes_prop
15788 { #1 }
15789 }
15790 {
15791 \file_if_exist_p:n
15792 { markdown theme #3.sty }
15793 }

```

```

15794     }
15795     {
15796         \msg_error:nn
15797         { markdown }
15798         { latex-theme-after-preamble }
15799     }

```

Otherwise, try loading a plain T<sub>E</sub>X theme instead.

```

15800     {
15801         \@@_plain_tex_load_theme:nnn
15802         { #1 }
15803         { #2 }
15804         { #3 }
15805     }
15806     \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.

```

15807     \bool_if:nTF
15808     {
15809         \bool_lazy_or_p:nn
15810         {
15811             \prop_if_in_p:Nn
15812             \g_@@_latex_built_in_themes_prop
15813             { #1 }
15814         }
15815         {
15816             \file_if_exist_p:n
15817             { markdown theme #3.sty }
15818         }
15819     }
15820     {
15821         \prop_get:NnNTF
15822         \g_@@_latex_loaded_themes_linenos_prop
15823         { #1 }
15824         \l_tmpa_tl
15825         {
15826             \prop_get:NnN
15827             \g_@@_latex_loaded_themes_versions_prop
15828             { #1 }
15829             \l_tmpb_tl
15830             \str_if_eq:nVTF
15831             { #2 }
15832             \l_tmpb_tl
15833             {
15834                 \msg_warning:nnnVn
15835                 { markdown }
15836                 { repeatedly-loaded-latex-theme }

```

```

15837         { #1 }
15838         \l_tmpa_tl
15839         { #2 }
15840     }
15841     {
15842         \msg_error:nnnnVV
15843         { markdown }
15844         { different-versions-of-latex-theme }
15845         { #1 }
15846         { #2 }
15847         \l_tmpb_tl
15848         \l_tmpa_tl
15849     }
15850 }
15851 {
15852     \prop_gput:Nnx
15853     \g_@@_latex_loaded_themes_linenos_prop
15854     { #1 }
15855     { \tex_the:D \tex_inputlineno:D } % noqa: W200
15856     \prop_gput:Nnn
15857     \g_@@_latex_loaded_themes_versions_prop
15858     { #1 }
15859     { #2 }

```

Load built-in plain TeX themes from the prop `\g_@@_latex_built_in_themes_prop` and from the filesystem otherwise.

```

15860     \prop_if_in:NnTF
15861     \g_@@_latex_built_in_themes_prop
15862     { #1 }
15863     {
15864         \msg_info:nnnn
15865         { markdown }
15866         { loading-built-in-latex-theme }
15867         { #1 }
15868         { #2 }
15869         \prop_item:Nn
15870         \g_@@_latex_built_in_themes_prop
15871         { #1 }
15872     }
15873     {
15874         \msg_info:nnnn
15875         { markdown }
15876         { loading-latex-theme }
15877         { #1 }
15878         { #2 }
15879         \RequirePackage
15880         { markdown theme #3 }

```



```

15881         }
15882     }
15883 }
15884 {
15885     \@@_plain_tex_load_theme:nnn
15886     { #1 }
15887     { #2 }
15888     { #3 }
15889 }
15890 \fi
15891 \else

```

If the Markdown package has not yet been loaded, postpone the loading until the Markdown package has finished loading.

```

15892     \msg_info:nnnn
15893     { markdown }
15894     { theme-loading-postponed }
15895     { #1 }
15896     { #2 }
15897     \AtEndOfPackage
15898     {
15899         \@@_set_theme:n
15900         { #1 @ #2 }
15901     }
15902 \fi
15903 }
15904 \msg_new:nnn
15905 { markdown }
15906 { theme-loading-postponed }
15907 {
15908     Postponing~loading~version~#2~of~Markdown~theme~#1~until~
15909     Markdown~package~has~finished~loading
15910 }
15911 \msg_new:nnn
15912 { markdown }
15913 { loading-built-in-latex-theme }
15914 { Loading~version~#2~of~built-in~LaTeX~Markdown~theme~#1 }
15915 \msg_new:nnn
15916 { markdown }
15917 { loading-latex-theme }
15918 { Loading~version~#2~of~LaTeX~Markdown~theme~#1 }
15919 \msg_new:nnn
15920 { markdown }
15921 { repeatedly-loaded-latex-theme }
15922 {
15923     Version~#3~of~LaTeX~Markdown~theme~#1~was~previously~
15924     loaded~on~line~#2,~not~loading~it~again

```

```

15925 }
15926 \msg_new:nnn
15927 { markdown }
15928 { different-versions-of-latex-theme }
15929 {
15930     Tried-to-load-version-#2-of-LaTeX-Markdown-theme-#1~
15931     but-version-#3-has-already-been-loaded-on-line-#4
15932 }
15933 \cs_generate_variant:Nn
15934 \msg_new:nnnn
15935 { nnVV }
15936 \tl_set:Nn
15937 \l_tmpa_tl
15938 { Cannot-load-LaTeX-Markdown-theme-#1-after~ }
15939 \tl_put_right:NV
15940 \l_tmpa_tl
15941 \c_backslash_str
15942 \tl_put_right:Nn
15943 \l_tmpa_tl
15944 { begin { document } }
15945 \tl_set:Nn
15946 \l_tmpb_tl
15947 { Load-Markdown-theme-#1-before~ }
15948 \tl_put_right:NV
15949 \l_tmpb_tl
15950 \c_backslash_str
15951 \tl_put_right:Nn
15952 \l_tmpb_tl
15953 { begin { document } }
15954 \msg_new:nnVV
15955 { markdown }
15956 { latex-theme-after-preamble }
15957 \l_tmpa_tl
15958 \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.

```

15959 \tl_set:Nn
15960 \l_tmpa_tl
15961 {
15962     \RequirePackage
15963     { graphicx }
15964     \markdownLoadPlainTeXTheme
15965 }
15966 \prop_gput:NnV
15967 \g_@@_latex_built_in_themes_prop
15968 { witiko / dot }

```

```

15969 \l_tmpa_tl
15970 \prop_gput:NnV
15971 \g_@@_latex_built_in_themes_prop
15972 { witiko / graphicx / http }
15973 \l_tmpa_tl
15974 \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.

```

15975 \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.

```

15976 \DeclareOption*{%
15977   \expandafter\markdownSetup\expandafter{\CurrentOption}}%
15978 \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.

```

15979 \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 the command `\DocumentMetadata` have been used, use the package `enumitem`. Otherwise, use the package `paralist`.

```

15980 \ExplSyntaxOn
15981 \bool_new:N
15982 \g_@@_tight_or_fancy_lists_bool
15983 \bool_gset_false:N
15984 \g_@@_tight_or_fancy_lists_bool
15985 \@@_if_option:nTF
15986 { tightLists }
15987 {
15988   \bool_gset_true:N
15989   \g_@@_tight_or_fancy_lists_bool
15990 }
15991 {

```

```

15992 \@@_if_option:nT
15993 { fancyLists }
15994 {
15995     \bool_gset_true:N
15996     \g_@@_tight_or_fancy_lists_bool
15997 }
15998 }
15999 \bool_new:N
16000 \g_@@_beamer_paralist_or_enumitem_bool
16001 \bool_gset_true:N
16002 \g_@@_beamer_paralist_or_enumitem_bool
16003 \ifclassloaded
16004 { beamer }
16005 { }
16006 {
16007     \ifpackageloaded
16008     { paralist }
16009     { }
16010     {
16011         \ifpackageloaded
16012         { enumitem }
16013         { }
16014         {
16015             \bool_gset_false:N
16016             \g_@@_beamer_paralist_or_enumitem_bool
16017         }
16018     }
16019 }
16020 \prg_generate_conditional_variant:Nnn
16021 \str_if_eq:nn
16022 { en }
16023 { TF }
16024 \bool_if:nT
16025 {
16026     \g_@@_tight_or_fancy_lists_bool &&
16027     ! \g_@@_beamer_paralist_or_enumitem_bool
16028 }
16029 {
16030     \str_if_eq:enTF
16031     { \markdownThemeVersion }
16032     { experimental }
16033     {
16034         \RequirePackage
16035         { enumitem }
16036     }
16037     {
16038         \IfDocumentMetadataTF

```

```

16039      {
16040          \RequirePackage
16041              { enumitem }
16042      }
16043      {
16044          \RequirePackage
16045              { paralist }
16046      }
16047  }
16048  }
16049  \ExplSyntaxOff

```

If we loaded the enumitem package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```

16050  \ExplSyntaxOn
16051  \cs_new:Nn
16052      \@@_latex_fancy_list_item_label_number:nn
16053  {
16054      \str_case:nn
16055          { #1 }
16056          {
16057              { Decimal } { #2 }
16058              { LowerRoman } { \int_to_roman:n { #2 } }
16059              { UpperRoman } { \int_to_Roman:n { #2 } }
16060              { LowerAlpha } { \int_to_alph:n { #2 } }
16061              { UpperAlpha } { \int_to_Alph:n { #2 } }
16062          }
16063  }
16064  \cs_new:Nn
16065      \@@_latex_fancy_list_item_label_delimiter:n
16066  {
16067      \str_case:nn
16068          { #1 }
16069          {
16070              { Default } { . }
16071              { OneParen } { ) }
16072              { Period } { . }
16073          }
16074  }
16075  \cs_new:Nn
16076      \@@_latex_fancy_list_item_label:nnn
16077  {
16078      \@@_latex_fancy_list_item_label_number:nn
16079          { #1 }
16080          { #3 }
16081      \@@_latex_fancy_list_item_label_delimiter:n
16082          { #2 }

```

```

16083 }
16084 \cs_generate_variant:Nn
16085 \@@_latex_fancy_list_item_label:nnn
16086 { VVn }
16087 \tl_new:N
16088 \l_@@_latex_fancy_list_item_label_number_style_tl
16089 \tl_new:N
16090 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16091 \@ifpackageloaded { enumitem } {
16092 \markdownSetup { rendererPrototypes = {

```

First, let's define the tight list item renderer prototypes.

```

16093     ulBeginTight = {
16094         \begin
16095             { itemize }
16096             [ noitemsep ]
16097     },
16098     ulEndTight = {
16099         \end
16100             { itemize }
16101     },
16102     olBeginTight = {
16103         \begin
16104             { enumerate }
16105             [ noitemsep ]
16106     },
16107     olEndTight = {
16108         \end
16109             { enumerate }
16110     },
16111     dlBeginTight = {
16112         \begin
16113             { description }
16114             [ noitemsep ]
16115     },
16116     dlEndTight = {
16117         \end
16118             { description }
16119     },

```

Second, let's define the fancy list item renderer prototypes.

```

16120     fancyOlBegin = {
16121         \group_begin:
16122         \tl_set:Nn
16123             \l_@@_latex_fancy_list_item_label_number_style_tl
16124             { #1 }
16125         \tl_set:Nn
16126             \l_@@_latex_fancy_list_item_label_delimiter_style_tl

```

```

16127         { #2 }
16128     \begin
16129         { enumerate }
16130 },
16131 fancyOlBeginTight = {
16132     \group_begin:
16133     \tl_set:Nn
16134         \l_@@_latex_fancy_list_item_label_number_style_tl
16135         { #1 }
16136     \tl_set:Nn
16137         \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16138         { #2 }
16139     \begin
16140         { enumerate }
16141         [ noitemsep ]
16142 },
16143 fancyOlEnd(|Tight) = {
16144     \end { enumerate }
16145     \group_end:
16146 },
16147 fancyOlItemWithNumber = {
16148     \item
16149     [
16150         \@@_latex_fancy_list_item_label:VVn
16151         \l_@@_latex_fancy_list_item_label_number_style_tl
16152         \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16153         { #1 }
16154     ]
16155 },
16156 } }

```

Otherwise, if we loaded the paralist package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```

16157 }
16158 { \ifpackageloaded { paralist } {
16159     \markdownSetup { rendererPrototypes = {

```

Make tight bullet lists a little less compact by adding extra vertical space above and below them.

```

16160     ulBeginTight = {
16161         \group_begin:
16162         \pltopsep=\topsep
16163         \plpartopsep=\partopsep
16164         \begin { compactitem }
16165     },
16166     ulEndTight = {
16167         \end { compactitem }
16168         \group_end:

```

```

16169 },
16170 fancyOlBegin = {
16171   \group_begin:
16172   \tl_set:Nn
16173     \l_@@_latex_fancy_list_item_label_number_style_tl
16174     { #1 }
16175   \tl_set:Nn
16176     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16177     { #2 }
16178   \begin { enumerate }
16179 },
16180 fancyOlEnd = {
16181   \end { enumerate }
16182   \group_end:
16183 },

```

Make tight ordered lists a little less compact by adding extra vertical space above and below them.

```

16184 olBeginTight = {
16185   \group_begin:
16186   \plpartopsep=\partopsep
16187   \pltopsep=\topsep
16188   \begin { compactenum }
16189 },
16190 olEndTight = {
16191   \end { compactenum }
16192   \group_end:
16193 },
16194 fancyOlBeginTight = {
16195   \group_begin:
16196   \tl_set:Nn
16197     \l_@@_latex_fancy_list_item_label_number_style_tl
16198     { #1 }
16199   \tl_set:Nn
16200     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16201     { #2 }
16202   \plpartopsep=\partopsep
16203   \pltopsep=\topsep
16204   \begin { compactenum }
16205 },
16206 fancyOlEndTight = {
16207   \end { compactenum }
16208   \group_end:
16209 },
16210 fancyOlItemWithNumber = {
16211   \item
16212   [

```



```

16213         \@@_latex_fancy_list_item_label:VVn
16214         \l_@@_latex_fancy_list_item_label_number_style_tl
16215         \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16216         { #1 }
16217     ]
16218 },

```

Make tight definition lists a little less compact by adding extra vertical space above and below them.

```

16219     dlBeginTight = {
16220         \group_begin:
16221         \plpartopsep=\partopsep
16222         \pltopsep=\topsep
16223         \begin { compactdesc }
16224     },
16225     dlEndTight = {
16226         \end { compactdesc }
16227         \group_end:
16228     }
16229 } }
16230 }
16231 {

```

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.

```

16232 \markdownSetup
16233 {
16234     rendererPrototypes = {
16235         ulBeginTight = \markdownRendererUlBegin,
16236         ulEndTight = \markdownRendererUlEnd,
16237         fancyOlBegin = \markdownRendererOlBegin,
16238         fancyOlEnd = \markdownRendererOlEnd,
16239         olBeginTight = \markdownRendererOlBegin,
16240         olEndTight = \markdownRendererOlEnd,
16241         fancyOlBeginTight = \markdownRendererOlBegin,
16242         fancyOlEndTight = \markdownRendererOlEnd,
16243         dlBeginTight = \markdownRendererDlBegin,
16244         dlEndTight = \markdownRendererDlEnd,
16245     },
16246 }
16247 } }
16248 \ExplSyntaxOff
16249 \RequirePackage{amsmath}

```

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

```

16250 \@ifpackageloaded{unicode-math}{

```

```

16251 \markdownSetup{rendererPrototypes={
16252     untickedBox = {\mdlgwhtsquare$},
16253 }}
16254 }{
16255     \RequirePackage{amssymb}
16256     \markdownSetup{rendererPrototypes={
16257         untickedBox = {\square$},
16258     }}
16259 }
16260 \RequirePackage{csvsimple}
16261 \RequirePackage{fancyvrb}
16262 \RequirePackage{graphicx}
16263 \markdownSetup{rendererPrototypes={
16264     hardLineBreak = {\},
16265     leftBrace = {\textbraceleft},
16266     rightBrace = {\textbraceright},
16267     dollarSign = {\textdollar},
16268     underscore = {\textunderscore},
16269     circumflex = {\textasciicircum},
16270     backslash = {\textbackslash},
16271     tilde = {\textasciitilde},
16272     pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by T<sub>E</sub>X during the typesetting. Therefore, even if we don't know whether a span of text is part of math formula or not when we are parsing markdown,<sup>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.

```

16273 codeSpan = {%
16274     \ifmmode
16275         \text{#1}%
16276     \else
16277         \texttt{#1}%
16278     \fi
16279 }

```

### 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

---

<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.

the content with TeX when the raw attribute is `tex`, and display the content as markdown otherwise.

```

16280 \ExplSyntaxOn
16281 \markdownSetup{
16282   rendererPrototypes = {
16283     contentBlock = {
16284       \str_case:nnF
16285       { #1 }
16286       {
16287         { csv }
16288         {
16289           \begin { table }
16290             \begin { center }
16291               \csvautotabular { #3 }
16292             \end{ center }
16293             \tl_if_empty:nF
16294             { #4 }
16295             { \caption { #4 } }
16296           \end { table }
16297         }
16298         { html }
16299         {

```

If we are using TeX4ht<sup>37</sup>, we will pass HTML elements to the output HTML document unchanged.

```

16300       \cs_if_exist:NTF
16301       \HCode
16302       {
16303         \if_mode_vertical:
16304         \IgnorePar
16305         \fi:
16306         \EndP
16307         \special
16308         { t4ht* < #3 }
16309         \par
16310         \ShowPar
16311       }
16312       {
16313         \@@_luaxml_print_html:n
16314         { #3 }
16315       }
16316     }
16317   { tex }
16318   {
16319     \markdownEscape

```

---

<sup>37</sup>See <https://tug.org/tex4ht/>.

```

16320         { #3 }
16321     }
16322 }
16323 {
16324     \markdownInput
16325     { #3 }
16326 }
16327 },
16328 },
16329 }
16330 \ExplSyntaxOff
16331 \markdownSetup{rendererPrototypes={
16332     ulBegin = {\begin{itemize}},
16333     ulEnd = {\end{itemize}},
16334     olBegin = {\begin{enumerate}},
16335     olItem = {\item{}},
16336     olItemWithNumber = {\item[#1.]},
16337     olEnd = {\end{enumerate}},
16338     dlBegin = {\begin{description}},
16339     dlItem = {\item[#1]},
16340     dlEnd = {\end{description}},
16341     emphasis = {\emph{#1}},
16342     tickedTextBox = {$\boxtimes$},
16343     halfTickedTextBox = {$\boxdot$}}

```

If HTML identifiers appear after a heading, we make them produce `\label` macros.

```

16344 \ExplSyntaxOn
16345 \seq_new:N
16346   \g_@@_header_identifiers_seq
16347 \markdownSetup
16348 {
16349     rendererPrototypes = {
16350         headerAttributeContextBegin = {
16351             \markdownSetup
16352             {
16353                 rendererPrototypes = {
16354                     attributeIdentifier = {
16355                         \seq_gput_right:Nn
16356                         \g_@@_header_identifiers_seq
16357                         { ##1 }
16358                     },
16359                 },
16360             }
16361         },
16362         headerAttributeContextEnd = {
16363             \seq_map_inline:Nn
16364             \g_@@_header_identifiers_seq
16365             { \label { ##1 } }

```

```

16366         \seq_gclear:N
16367         \g_@@_header_identifiers_seq
16368     },
16369 },
16370 }

```

If the `unnumbered` HTML class (or the `{-}` shorthand) appears after a heading the heading and all its subheadings will be unnumbered.

```

16371 \bool_new:N
16372 \l_@@_header_unnumbered_bool
16373 \markdownSetup
16374 {
16375     rendererPrototypes = {
16376         headerAttributeContextBegin += {
16377             \markdownSetup
16378             {
16379                 rendererPrototypes = {
16380                     attributeClassName = {
16381                         \bool_if:nT
16382                         {
16383                             \str_if_eq_p:nn
16384                             { ##1 }
16385                             { unnumbered } &&
16386                             ! \l_@@_header_unnumbered_bool
16387                         }
16388                     {
16389                         \group_begin:
16390                         \bool_set_true:N
16391                         \l_@@_header_unnumbered_bool
16392                         \c@secnumdepth = 0
16393                         \markdownSetup
16394                         {
16395                             rendererPrototypes = {
16396                                 sectionBegin = {
16397                                     \group_begin:
16398                                 },
16399                                 sectionEnd = {
16400                                     \group_end:
16401                                 },
16402                             },
16403                         }
16404                     }
16405                 },
16406             },
16407         },
16408     },
16409 }

```

```

16410 }
16411 \ExplSyntaxOff
16412 \markdownSetup{rendererPrototypes={
16413   superscript = {\textsuperscript{#1}},
16414   subscript = {\textsubscript{#1}},
16415   blockQuoteBegin = {\begin{quotation}},
16416   blockQuoteEnd = {\end{quotation}},
16417   inputVerbatim = {\VerbatimInput{#1}},
16418   thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
16419   note = {\footnote{#1}}}}

```

### 3.3.4.3 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

16420 \RequirePackage{ltxcmds}
16421 \ExplSyntaxOn
16422 \cs_gset_protected:Npn
16423   \markdownRendererInputFencedCodePrototype#1#2#3
16424   {
16425     \tl_if_empty:nTF
16426       { #2 }
16427       { \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.

```

16428     {
16429       \regex_extract_once:nnN
16430         { \w* }
16431         { #2 }
16432         \l_tmpa_seq
16433       \seq_pop_left:NN
16434         \l_tmpa_seq
16435         \l_tmpa_tl

```

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

```

16436     \ltx@ifpackageloaded
16437       { minted }
16438       {
16439         \catcode`\%=14\relax
16440         \catcode`\#=6\relax
16441         \exp_args:NV
16442           \inputminted
16443             \l_tmpa_tl
16444             { #1 }
16445         \catcode`\%=12\relax
16446         \catcode`\#=12\relax
16447       }
16448     {

```

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

```
16449         \ltx@ifpackageloaded
16450         { listings }
16451         { \lstinputlisting [ language = \l_tmpa_tl ] { #1 } }
```

When neither the listings package nor the minted package is loaded, act as though no infostring were given.

```
16452         { \markdownRendererInputFencedCode { #1 } { } { } }
16453     }
16454 }
16455 }
16456 \ExplSyntaxOff
```

Support the nesting of strong emphasis.

```
16457 \ExplSyntaxOn
16458 \def\markdownLATEXStrongEmphasis#1{
16459     \str_if_in:NnTF
16460         \f@series
16461         { b }
16462         { \textnormal{#1} }
16463         { \textbf{#1} }
16464 }
16465 \ExplSyntaxOff
16466 \markdownSetup{rendererPrototypes={strongEmphasis={%
16467     \protect\markdownLATEXStrongEmphasis{#1}}}}
```

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

```
16468 \@ifundefined{chapter}{%
16469     \markdownSetup{rendererPrototypes = {
16470         headingOne = {\section{#1}},
16471         headingTwo = {\subsection{#1}},
16472         headingThree = {\subsubsection{#1}},
16473         headingFour = {\paragraph{#1}},
16474         headingFive = {\subparagraph{#1}}}}
16475 }{%
16476     \markdownSetup{rendererPrototypes = {
16477         headingOne = {\chapter{#1}},
16478         headingTwo = {\section{#1}},
16479         headingThree = {\subsection{#1}},
16480         headingFour = {\subsubsection{#1}},
16481         headingFive = {\paragraph{#1}},
16482         headingSix = {\subparagraph{#1}}}}
16483 }%
```

#### 3.3.4.4 Tickboxes

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

```

16484 \markdownSetup{
16485   rendererPrototypes = {
16486     ulItem = {%
16487       \futurelet\markdownLaTeXCheckbox\markdownLaTeXUItem
16488     },
16489   },
16490 }
16491 \def\markdownLaTeXUItem{%
16492   \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
16493     \item[\markdownLaTeXCheckbox]%
16494     \expandafter\@gobble
16495   \else
16496     \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
16497       \item[\markdownLaTeXCheckbox]%
16498       \expandafter\expandafter\expandafter\@gobble
16499     \else
16500       \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
16501         \item[\markdownLaTeXCheckbox]%
16502         \expandafter\expandafter\expandafter\expandafter
16503         \expandafter\expandafter\expandafter\@gobble
16504       \else
16505         \item{}%
16506       \fi
16507     \fi
16508   \fi
16509 }

```

### 3.3.4.5 HTML elements

If the `html` option is enabled and we are using  $\text{\TeX 4ht}$ <sup>38</sup>, we will pass HTML elements to the output HTML document unchanged.

```

16510 \@ifundefined{HCode}{}{
16511   \markdownSetup{
16512     rendererPrototypes = {
16513       inlineHtmlTag = {%
16514         \ifvmode
16515           \IgnorePar
16516         \EndP
16517         \fi
16518         \HCode{#1}%
16519       },
16520       inputBlockHtmlElement = {%
16521         \ifvmode
16522           \IgnorePar
16523         \fi
16524         \EndP

```

---

<sup>38</sup>See <https://tug.org/tex4ht/>.



```

16525     \special{t4ht*<#1}%
16526     \par
16527     \ShowPar
16528   },
16529 },
16530 }
16531 }

```

### 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.

```

16532 \newcount\markdownLaTeXCitationsCounter
16533
16534 % Basic implementation
16535 \long\def\@gobblethree#1#2#3{%
16536 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
16537   \advance\markdownLaTeXCitationsCounter by 1\relax
16538   \ifx\relax#4\relax
16539     \ifx\relax#5\relax
16540       \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16541         \relax
16542         \cite{#1#2#6}% No prenotes/postnotes, just accumulate cites
16543         \expandafter\expandafter\expandafter
16544         \expandafter\expandafter\expandafter\expandafter
16545         \@gobblethree
16546       \fi
16547     \else% Before a postnote (#5), dump the accumulator
16548       \ifx\relax#1\relax\else
16549         \cite{#1}%
16550       \fi
16551       \cite[#5]{#6}%
16552       \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16553         \relax
16554       \else
16555         \expandafter\expandafter\expandafter
16556         \expandafter\expandafter\expandafter\expandafter
16557         \expandafter\expandafter\expandafter
16558         \expandafter\expandafter\expandafter\expandafter
16559         \markdownLaTeXBasicCitations
16560       \fi
16561       \expandafter\expandafter\expandafter
16562       \expandafter\expandafter\expandafter\expandafter{%
16563       \expandafter\expandafter\expandafter
16564       \expandafter\expandafter\expandafter\expandafter}%

```

```

16565     \expandafter\expandafter\expandafter
16566     \expandafter\expandafter\expandafter\expandafter{%
16567     \expandafter\expandafter\expandafter
16568     \expandafter\expandafter\expandafter\expandafter}%
16569     \expandafter\expandafter\expandafter
16570     \@gobblethree
16571     \fi
16572 \else% Before a prenote (#4), dump the accumulator
16573     \ifx\relax#1\relax\else
16574         \cite{#1}%
16575     \fi
16576     \ifnum\markdownLaTeXCitationsCounter>1\relax
16577         \space % Insert a space before the prenote in later citations
16578     \fi
16579     #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
16580     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16581     \relax
16582 \else
16583     \expandafter\expandafter\expandafter
16584     \expandafter\expandafter\expandafter\expandafter
16585     \markdownLaTeXBasicCitations
16586     \fi
16587     \expandafter\expandafter\expandafter{%
16588     \expandafter\expandafter\expandafter}%
16589     \expandafter\expandafter\expandafter{%
16590     \expandafter\expandafter\expandafter}%
16591     \expandafter
16592     \@gobblethree
16593     \fi\markdownLaTeXBasicCitations{#1#2#6},}
16594 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
16595
16596 % Natbib implementation
16597 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
16598     \advance\markdownLaTeXCitationsCounter by 1\relax
16599     \ifx\relax#3\relax
16600         \ifx\relax#4\relax
16601             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16602             \relax
16603             \citep{#1,#5}% No prenotes/postnotes, just accumulate cites
16604             \expandafter\expandafter\expandafter
16605             \expandafter\expandafter\expandafter\expandafter
16606             \@gobbletwo
16607         \fi
16608     \else% Before a postnote (#4), dump the accumulator
16609         \ifx\relax#1\relax\else
16610             \citep{#1}%
16611         \fi

```

```

16612 \citep[] [#4]{#5}%
16613 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16614 \relax
16615 \else
16616 \expandafter\expandafter\expandafter
16617 \expandafter\expandafter\expandafter\expandafter
16618 \expandafter\expandafter\expandafter
16619 \expandafter\expandafter\expandafter\expandafter
16620 \markdownLaTeXNatbibCitations
16621 \fi
16622 \expandafter\expandafter\expandafter
16623 \expandafter\expandafter\expandafter\expandafter{%
16624 \expandafter\expandafter\expandafter
16625 \expandafter\expandafter\expandafter\expandafter}%
16626 \expandafter\expandafter\expandafter
16627 \@gobbletwo
16628 \fi
16629 \else% Before a prenote (#3), dump the accumulator
16630 \ifx\relax#1\relax\relax\else
16631 \citep{#1}%
16632 \fi
16633 \citep[#3] [#4]{#5}%
16634 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16635 \relax
16636 \else
16637 \expandafter\expandafter\expandafter
16638 \expandafter\expandafter\expandafter\expandafter
16639 \markdownLaTeXNatbibCitations
16640 \fi
16641 \expandafter\expandafter\expandafter{%
16642 \expandafter\expandafter\expandafter}%
16643 \expandafter
16644 \@gobbletwo
16645 \fi\markdownLaTeXNatbibCitations{#1,#5}}
16646 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
16647 \advance\markdownLaTeXCitationsCounter by 1\relax
16648 \ifx\relax#3\relax
16649 \ifx\relax#4\relax
16650 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16651 \relax
16652 \citet{#1,#5}% No prenotes/postnotes, just accumulate cites
16653 \expandafter\expandafter\expandafter
16654 \expandafter\expandafter\expandafter\expandafter
16655 \@gobbletwo
16656 \fi
16657 \else% After a prenote or a postnote, dump the accumulator
16658 \ifx\relax#1\relax\else

```

```

16659         \citet{#1}%
16660     \fi
16661     , \citet[#3][#4]{#5}%
16662     \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
16663     \relax
16664     ,
16665     \else
16666         \ifnum
16667             \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
16668         \relax
16669         ,
16670         \fi
16671     \fi
16672     \expandafter\expandafter\expandafter
16673     \expandafter\expandafter\expandafter\expandafter
16674     \markdownLaTeXNatbibTextCitations
16675     \expandafter\expandafter\expandafter
16676     \expandafter\expandafter\expandafter\expandafter{%
16677     \expandafter\expandafter\expandafter
16678     \expandafter\expandafter\expandafter\expandafter}%
16679     \expandafter\expandafter\expandafter
16680     \@gobbletwo
16681 \fi
16682 \else% After a prenote or a postnote, dump the accumulator
16683 \ifx\relax#1\relax\relax\else
16684     \citet{#1}%
16685 \fi
16686     , \citet[#3][#4]{#5}%
16687     \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
16688     \relax
16689     ,
16690     \else
16691         \ifnum
16692             \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
16693         \relax
16694         ,
16695         \fi
16696     \fi
16697     \expandafter\expandafter\expandafter
16698     \markdownLaTeXNatbibTextCitations
16699     \expandafter\expandafter\expandafter{%
16700     \expandafter\expandafter\expandafter}%
16701     \expandafter
16702     \@gobbletwo
16703 \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
16704
16705 % BibLaTeX implementation

```

```

16706 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
16707   \advance\markdownLaTeXCitationsCounter by 1\relax
16708   \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16709   \relax
16710     \autocites#1[#3][#4]{#5}%
16711     \expandafter\@gobbletwo
16712   \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}
16713 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
16714   \advance\markdownLaTeXCitationsCounter by 1\relax
16715   \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16716   \relax
16717     \textcites#1[#3][#4]{#5}%
16718     \expandafter\@gobbletwo
16719   \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}
16720
16721 \markdownSetup{rendererPrototypes = {
16722   cite = {%
16723     \markdownLaTeXCitationsCounter=1%
16724     \def\markdownLaTeXCitationsTotal{#1}%
16725     \@ifundefined{autocites}{%
16726       \@ifundefined{citep}{%
16727         \expandafter\expandafter\expandafter
16728         \markdownLaTeXBasicCitations
16729         \expandafter\expandafter\expandafter{%
16730         \expandafter\expandafter\expandafter}%
16731         \expandafter\expandafter\expandafter{%
16732         \expandafter\expandafter\expandafter}%
16733       }{%
16734         \expandafter\expandafter\expandafter
16735         \markdownLaTeXNatbibCitations
16736         \expandafter\expandafter\expandafter{%
16737         \expandafter\expandafter\expandafter}%
16738       }%
16739     }{%
16740       \expandafter\expandafter\expandafter
16741       \markdownLaTeXBibLaTeXCitations
16742       \expandafter{\expandafter}%
16743     }},
16744   textCite = {%
16745     \markdownLaTeXCitationsCounter=1%
16746     \def\markdownLaTeXCitationsTotal{#1}%
16747     \@ifundefined{autocites}{%
16748       \@ifundefined{citep}{%
16749         \expandafter\expandafter\expandafter
16750         \markdownLaTeXBasicTextCitations
16751         \expandafter\expandafter\expandafter{%
16752         \expandafter\expandafter\expandafter}%

```

```

16753         \expandafter\expandafter\expandafter{%
16754         \expandafter\expandafter\expandafter}%
16755     }{%
16756         \expandafter\expandafter\expandafter
16757         \markdownLaTeXNatbibTextCitations
16758         \expandafter\expandafter\expandafter{%
16759         \expandafter\expandafter\expandafter}%
16760     }%
16761 }{%
16762     \expandafter\expandafter\expandafter
16763     \markdownLaTeXBibLaTeXTextCitations
16764     \expandafter{\expandafter}%
16765 }}}}
```

### 3.3.4.7 Links

Here is an implementation for hypertext links and relative references.

```

16766 \RequirePackage{url}
16767 \RequirePackage{expl3}
16768 \ExplSyntaxOn
16769 \cs_gset_protected:Npn
16770   \markdownRendererLinkPrototype
16771   #1#2#3#4
16772   {
16773     \tl_set:Nn \l_tmpa_tl { #1 }
16774     \tl_set:Nn \l_tmpb_tl { #2 }
16775     \bool_set:Nn
16776       \l_tmpa_bool
16777       {
16778         \tl_if_eq_p:NN
16779           \l_tmpa_tl
16780           \l_tmpb_tl
16781       }
16782     \tl_set:Nn \l_tmpa_tl { #4 }
16783     \bool_set:Nn
16784       \l_tmpb_bool
16785       {
16786         \tl_if_empty_p:N
16787           \l_tmpa_tl
16788       }
16789     \bool_if:nTF
```

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.

```

16789     \bool_if:nTF
16790     {
16791       \l_tmpa_bool && \l_tmpb_bool
16792     }
```

```

16793     {
16794         \markdownLaTeXRendererAutolink { #2 } { #3 }
16795     }
16796     {
16797         \markdownLaTeXRendererDirectOrIndirectLink
16798         { #1 } { #2 } { #3 } { #4 }
16799     }
16800 }
16801 \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.

```

16802     \tl_set:Nn
16803     \l_tmpa_tl
16804     { #2 }
16805     \tl_trim_spaces:N
16806     \l_tmpa_tl
16807     \tl_set:Nx
16808     \l_tmpb_tl
16809     {
16810         \tl_range:Nnn
16811         \l_tmpa_tl
16812         { 1 }
16813         { 1 }
16814     }
16815     \str_if_eq:NNTF
16816     \l_tmpb_tl
16817     \c_hash_str
16818     {
16819         \tl_set:Nx
16820         \l_tmpb_tl
16821         {
16822             \tl_range:Nnn
16823             \l_tmpa_tl
16824             { 2 }
16825             { -1 }
16826         }
16827         \exp_args:NV
16828         \ref
16829         \l_tmpb_tl
16830     }
16831     {
16832         \url { #2 }
16833     }
16834 }
16835 \ExplSyntaxOff
16836 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%

```

```
16837 #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.

```
16838 \newcount\markdownLaTeXRowCount
16839 \newcount\markdownLaTeXRowTotal
16840 \newcount\markdownLaTeXColumnCounter
16841 \newcount\markdownLaTeXColumnTotal
16842 \newtoks\markdownLaTeXTable
16843 \newtoks\markdownLaTeXTableAlignment
16844 \newtoks\markdownLaTeXTableEnd
16845 \AtBeginDocument{%
16846   \ifpackageloaded{booktabs}{%
16847     \def\markdownLaTeXTopRule{\toprule}%
16848     \def\markdownLaTeXMidRule{\midrule}%
16849     \def\markdownLaTeXBottomRule{\bottomrule}%
16850   }{%
16851     \def\markdownLaTeXTopRule{\hline}%
16852     \def\markdownLaTeXMidRule{\hline}%
16853     \def\markdownLaTeXBottomRule{\hline}%
16854   }%
16855 }
16856 \markdownSetup{rendererPrototypes={
16857   table = {%
16858     \markdownLaTeXTable={}%
16859     \markdownLaTeXTableAlignment={}%
16860     \markdownLaTeXTableEnd={%
16861       \markdownLaTeXBottomRule
16862     \end{tabular}}}%
16863   \ifx\empty#1\empty\else
16864     \addto@hook\markdownLaTeXTable{%
16865       \begin{table}
16866       \centering}%
16867     \addto@hook\markdownLaTeXTableEnd{%
16868       \caption{#1}}}%
16869   \fi
16870 }
16871 }}
```

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.

```
16872 \ExplSyntaxOn
16873 \seq_new:N
16874   \l_@@_table_identifiers_seq
16875 \markdownSetup {
16876   rendererPrototypes = {
```



```

16877     table += {
16878         \seq_map_inline:Nn
16879         \l_@@_table_identifiers_seq
16880         {
16881             \addto@hook
16882             \markdownLaTeXTableEnd
16883             { \label { ##1 } }
16884         }
16885     },
16886 }
16887 }
16888 \markdownSetup {
16889     rendererPrototypes = {
16890         tableAttributeContextBegin = {
16891             \group_begin:
16892             \markdownSetup {
16893                 rendererPrototypes = {
16894                     attributeIdentifier = {
16895                         \seq_put_right:Nn
16896                         \l_@@_table_identifiers_seq
16897                         { ##1 }
16898                     },
16899                 },
16900             }
16901         },
16902         tableAttributeContextEnd = {
16903             \group_end:
16904         },
16905     },
16906 }
16907 \ExplSyntaxOff
16908 \markdownSetup{rendererPrototypes={
16909     table += {%
16910         \ifx\empty#1\empty\else
16911             \addto@hook\markdownLaTeXTableEnd{%
16912                 \end{table}}}%
16913         \fi
16914         \addto@hook\markdownLaTeXTable{\begin{tabular}}}%
16915         \markdownLaTeXRowCounter=0%
16916         \markdownLaTeXRowTotal=#2%
16917         \markdownLaTeXColumnTotal=#3%
16918         \markdownLaTeXRenderTableRow
16919     }
16920 }}
16921 \def\markdownLaTeXRenderTableRow#1{%
16922     \markdownLaTeXColumnCounter=0%
16923     \ifnum\markdownLaTeXRowCounter=0\relax

```

```

16924 \markdownLaTeXReadAlignments#1%
16925 \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
16926 \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
16927 \the\markdownLaTeXTableAlignment}}%
16928 \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
16929 \else
16930 \markdownLaTeXRenderTableCell#1%
16931 \fi
16932 \ifnum\markdownLaTeXRowCount=1\relax
16933 \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
16934 \fi
16935 \advance\markdownLaTeXRowCount by 1\relax
16936 \ifnum\markdownLaTeXRowCount>\markdownLaTeXRowTotal\relax
16937 \the\markdownLaTeXTable
16938 \the\markdownLaTeXTableEnd
16939 \expandafter\@gobble
16940 \fi\markdownLaTeXRenderTableRow}
16941 \def\markdownLaTeXReadAlignments#1{%
16942 \advance\markdownLaTeXColumnCounter by 1\relax
16943 \if#1d%
16944 \addto@hook\markdownLaTeXTableAlignment{1}%
16945 \else
16946 \addto@hook\markdownLaTeXTableAlignment{#1}%
16947 \fi
16948 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
16949 \expandafter\@gobble
16950 \fi\markdownLaTeXReadAlignments}
16951 \def\markdownLaTeXRenderTableCell#1{%
16952 \advance\markdownLaTeXColumnCounter by 1\relax
16953 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
16954 \addto@hook\markdownLaTeXTable{#1&}%
16955 \else
16956 \addto@hook\markdownLaTeXTable{#1\\}%
16957 \expandafter\@gobble
16958 \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.

```

16959
16960 \markdownIfOption{lineBlocks}{%
16961 \RequirePackage{verse}
16962 \markdownSetup{rendererPrototypes={
16963 lineBlockBegin = {%
16964 \begingroup
16965 \def\markdownRendererHardLineBreak{\\}%

```

```

16966         \begin{verse}%
16967     },
16968     lineBlockEnd = {%
16969         \end{verse}%
16970     \endgroup
16971 },
16972 }}
16973 }{}
16974

```

### 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.

```

16975 \ExplSyntaxOn
16976 \keys_define:nn
16977 { markdown / jekyllData }
16978 {
16979     author .code:n = {
16980         \author
16981         { #1 }
16982     },
16983     date .code:n = {
16984         \date
16985         { #1 }
16986     },
16987     title .code:n = {
16988         \title
16989         { #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.

```

16990     \AddToHook
16991     { begindocument / end }
16992     { \maketitle }
16993 },
16994 }

```

### 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.

```

16995 \@@_if_option:nT
16996 { mark }

```

```

16997 {
16998   \sys_if_engine luatex:TF
16999   {
17000     \RequirePackage
17001       { luacolor }
17002     \RequirePackage
17003       { lua-ul }
17004     \markdownSetup
17005       {
17006         rendererPrototypes = {
17007           mark = {
17008             \highLight
17009               { #1 }
17010           },
17011         }
17012       }
17013   }
17014   {
17015     \RequirePackage
17016       { xcolor }
17017     \RequirePackage
17018       { soul }
17019     \markdownSetup
17020       {
17021         rendererPrototypes = {
17022           mark = {
17023             \hl
17024               { #1 }
17025           },
17026         }
17027       }
17028   }
17029 }

```

### 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.

```

17030 \@@_if_option:nT
17031 { strikeThrough }
17032 {
17033   \sys_if_engine luatex:TF
17034   {
17035     \RequirePackage
17036       { lua-ul }
17037     \markdownSetup
17038       {

```

```

17039         rendererPrototypes = {
17040             strikeThrough = {
17041                 \strikeThrough
17042                 { #1 }
17043             },
17044         }
17045     }
17046 }
17047 {
17048     \RequirePackage
17049     { soul }
17050     \markdownSetup
17051     {
17052         rendererPrototypes = {
17053             strikeThrough = {
17054                 \st
17055                 { #1 }
17056             },
17057         }
17058     }
17059 }
17060 }

```

### 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 `<key>=<value>` 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 L<sup>A</sup>T<sub>E</sub>X labels for referencing figures.

```

17061 \seq_new:N
17062 \l_@@_image_identifiers_seq
17063 \markdownSetup {
17064     rendererPrototypes = {
17065         image = {
17066             \tl_if_empty:nTF
17067             { #4 }
17068             {
17069                 \begin { center }
17070                 \includegraphics
17071                 [ alt = { #1 } ]
17072                 { #3 }
17073                 \end { center }
17074             }
17075         }

```

```

17076         \begin { figure }
17077         \begin { center }
17078             \includegraphics
17079             [ alt = { #1 } ]
17080             { #3 }
17081             \caption { #4 }
17082             \seq_map_inline:Nn
17083             \l_@@_image_identifiers_seq
17084             { \label { ##1 } }
17085         \end { center }
17086     \end { figure }
17087 }
17088 },
17089 }
17090 }
17091 \@@_if_option:nT
17092 { linkAttributes }
17093 {
17094     \RequirePackage { graphicx }
17095 }
17096 \markdownSetup {
17097     rendererPrototypes = {
17098         imageAttributeContextBegin = {
17099             \group_begin:
17100             \markdownSetup {
17101                 rendererPrototypes = {
17102                     attributeIdentifier = {
17103                         \seq_put_right:Nn
17104                         \l_@@_image_identifiers_seq
17105                         { ##1 }
17106                     },
17107                     attributeKeyValue = {
17108                         \setkeys
17109                         { Gin }
17110                         { { ##1 } = { ##2 } }
17111                     },
17112                 },
17113             }
17114         },
17115         imageAttributeContextEnd = {
17116             \group_end:
17117         },
17118     },
17119 }
17120 \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`.

```

17121 \ExplSyntaxOn
17122 \cs_new:Nn
17123   \@@_luaxml_print_html:n
17124   {
17125     \luabridge_now:n
17126     {
17127       local~input_file = assert(io.open(" #1 ", "r"))
17128       local~input = assert(input_file:read("*a"))
17129       assert(input_file:close())
17130       input = "<body>" .. input .. "</body>"
17131       local~dom = require("luaxml-domobject").html_parse(input)
17132       local~output = require("luaxml-htmltemplates"):process_dom(dom)
17133       print(output)
17134     }
17135   }
17136 \cs_gset_protected:Npn
17137   \markdownRendererInputRawInlinePrototype#1#2
17138   {
17139     \str_case:nnF
17140       { #2 }
17141       {
17142         { latex }
17143         {
17144           \@@_plain_tex_default_input_raw_inline:nn
17145             { #1 }
17146             { tex }
17147         }
17148         { html }
17149         {

```

If we are using `TeX4ht`<sup>39</sup>, we will pass HTML elements to the output HTML document unchanged.

```

17150       \cs_if_exist:NTF
17151         \HCode
17152         {
17153           \if_mode_vertical:
17154             \IgnorePar
17155             \EndP
17156           \fi:
17157           \special
17158             { t4ht* < #1 }

```

---

<sup>39</sup>See <https://tug.org/tex4ht/>.

```

17159         }
17160         {
17161             \@@_luaxml_print_html:n
17162             { #1 }
17163         }
17164     }
17165 }
17166 {
17167     \@@_plain_tex_default_input_raw_inline:nn
17168     { #1 }
17169     { #2 }
17170 }
17171 }
17172 \cs_gset_protected:Npn
17173 \markdownRendererInputRawBlockPrototype#1#2
17174 {
17175     \str_case:nnF
17176     { #2 }
17177     {
17178         { latex }
17179         {
17180             \@@_plain_tex_default_input_raw_block:nn
17181             { #1 }
17182             { tex }
17183         }
17184         { html }
17185         {

```

If we are using  $\text{\TeX}4\text{ht}$ <sup>40</sup>, we will pass HTML elements to the output HTML document unchanged.

```

17186         \cs_if_exist:NTF
17187         \HCode
17188         {
17189             \if_mode_vertical:
17190             \IgnorePar
17191             \fi:
17192             \EndP
17193             \special
17194             { t4ht* < #1 }
17195             \par
17196             \ShowPar
17197         }
17198         {
17199             \@@_luaxml_print_html:n
17200             { #1 }
17201         }

```

---

<sup>40</sup>See <https://tug.org/tex4ht/>.



```

17202     }
17203   }
17204   {
17205     \@@_plain_tex_default_input_raw_block:nn
17206     { #1 }
17207     { #2 }
17208   }
17209 }

```

### 3.3.4.15 Bracketed spans

If the `bracketedSpans` option is enabled, we will register any identifiers, so that they can be used as  $\text{\LaTeX}$  labels for referencing the last  $\text{\LaTeX}$  counter that has been incremented in e.g. ordered lists.

```

17210 \seq_new:N
17211   \l_@@_bracketed_span_identifiers_seq
17212 \markdownSetup {
17213   rendererPrototypes = {
17214     bracketedSpanAttributeContextBegin = {
17215       \group_begin:
17216       \markdownSetup {
17217         rendererPrototypes = {
17218           attributeIdentifier = {
17219             \seq_put_right:Nn
17220               \l_@@_bracketed_span_identifiers_seq
17221               { ##1 }
17222           },
17223         },
17224       }
17225     },
17226     bracketedSpanAttributeContextEnd = {
17227       \seq_map_inline:Nn
17228         \l_@@_bracketed_span_identifiers_seq
17229         { \label { ##1 } }
17230       \group_end:
17231     },
17232   },
17233 }
17234 \ExplSyntaxOff
17235 \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.

```
17236 \newcommand\markdownMakeOther{%
17237   \count0=128\relax
17238   \loop
17239     \catcode\count0=11\relax
17240     \advance\count0 by 1\relax
17241   \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.

```
17242 \def\markdownMakeOther{%
17243   \count0=128\relax
17244   \loop
17245     \catcode\count0=11\relax
17246     \advance\count0 by 1\relax
17247   \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.

```
17248   \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).

```
17249 \long\def\inputmarkdown{%
17250   \dosingleempty
17251   \doinputmarkdown}%
17252 \long\def\doinputmarkdown[#1]#2{%
17253   \begingroup
17254     \iffirstargument
17255       \setupmarkdown[#1]%
17256     \fi
17257     \markdownInput{#2}%
17258   \endgroup}%
```

```

17259 \long\def\inputyaml{%
17260   \dosingleempty
17261   \doinputyaml}%
17262 \long\def\doinputyaml[#1]#2{%
17263   \doinputmarkdown
17264   [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. [20, sec. 31]. According to Eijkhout [21, 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.

```

17265 \startluacode
17266   document.markdown_buffering = false
17267   local function preserve_trailing_spaces(line)
17268     if document.markdown_buffering then
17269       line = line:gsub("[ \t][ \t]$", "\t\t")
17270     end
17271     return line
17272   end
17273   resolvers.installinputlinehandler(preserve_trailing_spaces)
17274 \stopluacode
17275 \begingroup
17276   \catcode\|=0%
17277   \catcode\|=12%
17278   |gdef|startmarkdown{%
17279     |ctxlua{document.markdown_buffering = true}%
17280     |markdownReadAndConvert{\stopmarkdown}%
17281     {|stopmarkdown}}%
17282   |gdef|stopmarkdown{%
17283     |ctxlua{document.markdown_buffering = false}%
17284     |markdownEnd}%
17285   |gdef|startyaml{%
17286     |begingroup
17287     |ctxlua{document.markdown_buffering = true}%
17288     |setupyaml[jekyllData, expectJekyllData, ensureJekyllData]%
17289     |markdownReadAndConvert{\stopyaml}%
17290     {|stopyaml}}%
17291   |gdef|stopyaml{%
17292     |ctxlua{document.markdown_buffering = false}%
17293     |yamlEnd}%
17294 |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.

```
17295 \ExplSyntaxOn
17296 \prop_new:N \g_@@_context_loaded_themes_linenos_prop
17297 \prop_new:N \g_@@_context_loaded_themes_versions_prop
17298 \cs_gset:Nn
17299   \@@_load_theme:nnn
17300   {
```

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.

```
17301   \bool_if:nTF
17302   {
17303     \bool_lazy_or_p:nn
17304     {
17305       \prop_if_in_p:Nn
17306         \g_@@_context_built_in_themes_prop
17307         { #1 }
17308     }
17309     {
17310       \file_if_exist_p:n
17311         { t - markdown theme #3.tex }
17312     }
17313   }
17314   {
17315     \prop_get:NnNTF
17316       \g_@@_context_loaded_themes_linenos_prop
17317       { #1 }
17318     \l_tmpa_tl
17319     {
17320       \prop_get:NnN
17321         \g_@@_context_loaded_themes_versions_prop
17322         { #1 }
17323       \l_tmpb_tl
17324       \str_if_eq:nVTF
17325         { #2 }
17326       \l_tmpb_tl
17327       {
17328         \msg_warning:nnnVn
17329           { markdown }
17330           { repeatedly-loaded-context-theme }
17331         { #1 }
```

```

17332         \l_tmpa_tl
17333         { #2 }
17334     }
17335     {
17336         \msg_error:nnnnVV
17337         { markdown }
17338         { different-versions-of-context-theme }
17339         { #1 }
17340         { #2 }
17341         \l_tmpb_tl
17342         \l_tmpa_tl
17343     }
17344 }
17345 {
17346     \prop_gput:Nnx
17347     \g_@@_context_loaded_themes_linenos_prop
17348     { #1 }
17349     { \tex_the:D \tex_inputlineno:D } % noqa: W200
17350     \prop_gput:Nnn
17351     \g_@@_context_loaded_themes_versions_prop
17352     { #1 }
17353     { #2 }

```

Load built-in plain TeX themes from the prop `\g_@@_context_built_in_themes_prop` and from the filesystem otherwise.

```

17354     \prop_if_in:NnTF
17355     \g_@@_context_built_in_themes_prop
17356     { #1 }
17357     {
17358         \msg_info:nnnn
17359         { markdown }
17360         { loading-built-in-context-theme }
17361         { #1 }
17362         { #2 }
17363         \prop_item:Nn
17364         \g_@@_context_built_in_themes_prop
17365         { #1 }
17366     }
17367     {
17368         \msg_info:nnnn
17369         { markdown }
17370         { loading-context-theme }
17371         { #1 }
17372         { #2 }
17373         \usemodule
17374         [ t ]
17375         [ markdown theme #3 ]

```

```

17376         }
17377     }
17378 }
17379 {
17380     \@@_plain_tex_load_theme:nnn
17381     { #1 }
17382     { #2 }
17383     { #3 }
17384 }
17385 }
17386 \msg_new:nnn
17387 { markdown }
17388 { loading-built-in-context-theme }
17389 { Loading~version~#2~of~built-in~ConTeXt~Markdown~theme~#1 }
17390 \msg_new:nnn
17391 { markdown }
17392 { loading-context-theme }
17393 { Loading~version~#2~of~ConTeXt~Markdown~theme~#1 }
17394 \msg_new:nnn
17395 { markdown }
17396 { repeatedly-loaded-context-theme }
17397 {
17398     Version~#3~of~ConTeXt~Markdown~theme~#1~was~previously~
17399     loaded~on~line~#2,~not~loading~it~again
17400 }
17401 \msg_new:nnn
17402 { markdown }
17403 { different-versions-of-context-theme }
17404 {
17405     Tried~to~load~version~#2~of~ConTeXt~Markdown~theme~#1~
17406     but~version~#3~has~already~been~loaded~on~line~#4
17407 }
17408 \ExplSyntaxOff

```

The [witiko/markdown/defaults](#) ConTeXt theme provides default definitions for token renderer prototypes. First, the ConTeXt theme loads the plain T<sub>E</sub>X theme with the default definitions for plain T<sub>E</sub>X:

```

17409 \markdownLoadPlainTeXTheme

```

Next, the ConTeXt theme overrides some of the plain T<sub>E</sub>X definitions. See Section 3.4.3 for the actual definitions.

### 3.4.3 Token Renderer Prototypes

The following configuration should be considered placeholder. If the option `plain` has been enabled (see Section 2.2.2.3), none of the definitions will take effect.

```

17410 \markdownIfOption{plain}{\iffalse}{\iftrue}

```

```

17411 \def\markdownRendererHardLineBreakPrototype{\blank}%
17412 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
17413 \def\markdownRendererRightBracePrototype{\textbraceright}%
17414 \def\markdownRendererDollarSignPrototype{\textdollar}%
17415 \def\markdownRendererPercentSignPrototype{\percent}%
17416 \def\markdownRendererUnderscorePrototype{\textunderscore}%
17417 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
17418 \def\markdownRendererBackslashPrototype{\textbackslash}%
17419 \def\markdownRendererTildePrototype{\textasciitilde}%
17420 \def\markdownRendererPipePrototype{\char`|}%
17421 \def\markdownRendererLinkPrototype#1#2#3#4{%
17422   \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
17423   \fi\texttt<\hyphenatedurl{#3}>}}%
17424 \usemodule[database]
17425 \defineseparatedlist
17426   [MarkdownConTeXtCSV]
17427   [separator={,},
17428    before=\bTABLE,after=\eTABLE,
17429    first=\bTR,last=\eTR,
17430    left=\bTD,right=\eTD]
17431 \def\markdownConTeXtCSV{csv}
17432 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
17433   \def\markdownConTeXtCSV@arg{#1}%
17434   \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
17435     \placetable[] [tab:#1]{#4}{%
17436       \processseparatedfile[MarkdownConTeXtCSV] [#3]}%
17437   \else
17438     \markdownInput{#3}%
17439   \fi}%
17440 \def\markdownRendererImagePrototype#1#2#3#4{%
17441   \placefigure[] []{#4}{\externalfigure[#3]}}%
17442 \def\markdownRendererUlBeginPrototype{\startitemize}%
17443 \def\markdownRendererUlBeginTightPrototype{\startitemize[packed]}%
17444 \def\markdownRendererUlItemPrototype{\item}%
17445 \def\markdownRendererUlEndPrototype{\stopitemize}%
17446 \def\markdownRendererUlEndTightPrototype{\stopitemize}%
17447 \def\markdownRendererOlBeginPrototype{\startitemize[n]}%
17448 \def\markdownRendererOlBeginTightPrototype{\startitemize[packed,n]}%
17449 \def\markdownRendererOlItemPrototype{\item}%
17450 \def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
17451 \def\markdownRendererOlEndPrototype{\stopitemize}%
17452 \def\markdownRendererOlEndTightPrototype{\stopitemize}%
17453 \definedescription
17454   [MarkdownConTeXtDlItemPrototype]
17455   [location=hanging,
17456    margin=standard,
17457    headstyle=bold]%

```

```

17458 \definestartstop
17459   [MarkdownConTeXtDlPrototype]
17460   [before=\blank,
17461    after=\blank]%
17462 \definestartstop
17463   [MarkdownConTeXtDlTightPrototype]
17464   [before=\blank\startpacked,
17465    after=\stoppacked\blank]%
17466 \def\markdownRendererDlBeginPrototype{%
17467   \startMarkdownConTeXtDlPrototype}%
17468 \def\markdownRendererDlBeginTightPrototype{%
17469   \startMarkdownConTeXtDlTightPrototype}%
17470 \def\markdownRendererDlItemPrototype#1{%
17471   \startMarkdownConTeXtDlItemPrototype{#1}}%
17472 \def\markdownRendererDlItemEndPrototype{%
17473   \stopMarkdownConTeXtDlItemPrototype}%
17474 \def\markdownRendererDlEndPrototype{%
17475   \stopMarkdownConTeXtDlPrototype}%
17476 \def\markdownRendererDlEndTightPrototype{%
17477   \stopMarkdownConTeXtDlTightPrototype}%
17478 \def\markdownRendererEmphasisPrototype#1{\em#1}%
17479 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
17480 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
17481 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
17482 \def\markdownRendererLineBlockBeginPrototype{%
17483   \begingroup
17484     \def\markdownRendererHardLineBreak{
17485       }%
17486     \startlines
17487   }%
17488 \def\markdownRendererLineBlockEndPrototype{%
17489   \stoplines
17490   \endgroup
17491 }%
17492 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%

```

### 3.4.3.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

17493 \ExplSyntaxOn
17494 \cs_gset:Npn
17495   \markdownRendererInputFencedCodePrototype#1#2#3
17496   {
17497     \tl_if_empty:nTF
17498       { #2 }
17499       { \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 ConT<sub>E</sub>Xt `\definetyping` macro, which allows the user to set up code highlighting mapping as follows:

```
\definetyping [latex]
\setuptyping [latex] [option=TEX]

\starttext
  \startmarkdown
~~~ latex
\documentclass{article}
\begin{document}
  Hello world!
\end{document}
~~~
  \stopmarkdown
\stoptext
```

```
17500     {
17501         \regex_extract_once:nnN
17502         { \w* }
17503         { #2 }
17504         \l_tmpa_seq
17505         \seq_pop_left:NN
17506         \l_tmpa_seq
17507         \l_tmpa_tl
17508         \typefile[ \l_tmpa_tl ][] {#1}
17509     }
17510 }
17511 \ExplSyntaxOff
17512 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
17513 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
17514 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
17515 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
17516 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
17517 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
17518 \def\markdownRendererThematicBreakPrototype{%
17519     \blackrule[height=1pt, width=\hsize]}%
17520 \def\markdownRendererNotePrototype#1{\footnote{#1}}%
17521 \def\markdownRendererTickedBoxPrototype{$\boxtimes$}
17522 \def\markdownRendererHalfTickedBoxPrototype{$\boxdot$}
17523 \def\markdownRendererUntickedBoxPrototype{$\square$}
17524 \def\markdownRendererStrikeThroughPrototype#1{\overstrides{#1}}
17525 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
```

```

17526 \def\markdownRendererSubscriptPrototype#1{\low{#1}}
17527 \def\markdownRendererDisplayMathPrototype#1{%
17528   \startformula#1\stopformula}%

```

### 3.4.3.2 Tables

There is a basic implementation of tables.

```

17529 \newcount\markdownConTeXtRowCounter
17530 \newcount\markdownConTeXtRowTotal
17531 \newcount\markdownConTeXtColumnCounter
17532 \newcount\markdownConTeXtColumnTotal
17533 \newtoks\markdownConTeXtTable
17534 \newtoks\markdownConTeXtTableFloat
17535 \def\markdownRendererTablePrototype#1#2#3{%
17536   \markdownConTeXtTable={}%
17537   \ifx\empty#1\empty
17538     \markdownConTeXtTableFloat={%
17539       \the\markdownConTeXtTable}%
17540   \else
17541     \markdownConTeXtTableFloat={%
17542       \placetable{#1}{\the\markdownConTeXtTable}}%
17543   \fi
17544   \begingroup
17545   \setupTABLE[r][each][topframe=off, bottomframe=off,
17546     leftframe=off, rightframe=off]
17547   \setupTABLE[c][each][topframe=off, bottomframe=off,
17548     leftframe=off, rightframe=off]
17549   \setupTABLE[r][1][topframe=on, bottomframe=on]
17550   \setupTABLE[r][#1][bottomframe=on]
17551   \markdownConTeXtRowCounter=0%
17552   \markdownConTeXtRowTotal=#2%
17553   \markdownConTeXtColumnTotal=#3%
17554   \markdownConTeXtRenderTableRow}
17555 \def\markdownConTeXtRenderTableRow#1{%
17556   \markdownConTeXtColumnCounter=0%
17557   \ifnum\markdownConTeXtRowCounter=0\relax
17558     \markdownConTeXtReadAlignments#1%
17559     \markdownConTeXtTable={\bTABLE}%
17560   \else
17561     \markdownConTeXtTable=\expandafter{%
17562       \the\markdownConTeXtTable\bTR}%
17563     \markdownConTeXtRenderTableCell#1%
17564     \markdownConTeXtTable=\expandafter{%
17565       \the\markdownConTeXtTable\eTR}%
17566   \fi
17567   \advance\markdownConTeXtRowCounter by 1\relax
17568   \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax

```

```

17569 \markdownConTeXtTable=\expandafter{%
17570 \the\markdownConTeXtTable\eTABLE}%
17571 \the\markdownConTeXtTableFloat
17572 \endgroup
17573 \expandafter\gobbleoneargument
17574 \fi\markdownConTeXtRenderTableRow}
17575 \def\markdownConTeXtReadAlignments#1{%
17576 \advance\markdownConTeXtColumnCounter by 1\relax
17577 \if#1d%
17578 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
17579 \fi\if#1l%
17580 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
17581 \fi\if#1c%
17582 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
17583 \fi\if#1r%
17584 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
17585 \fi
17586 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
17587 \else
17588 \expandafter\gobbleoneargument
17589 \fi\markdownConTeXtReadAlignments}
17590 \def\markdownConTeXtRenderTableCell#1{%
17591 \advance\markdownConTeXtColumnCounter by 1\relax
17592 \markdownConTeXtTable=\expandafter{%
17593 \the\markdownConTeXtTable\bTD#1\eTD}%
17594 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
17595 \else
17596 \expandafter\gobbleoneargument
17597 \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`.

```

17598 \ExplSyntaxOn
17599 \cs_gset:Npn
17600 \markdownRendererInputRawInlinePrototype#1#2
17601 {
17602 \str_case:nnF
17603 { #2 }
17604 {
17605 { latex }
17606 {
17607 \@@_plain_tex_default_input_raw_inline:nn
17608 { #1 }
17609 { context }
17610 }

```

```

17611     }
17612     {
17613         \@@_plain_tex_default_input_raw_inline:nn
17614         { #1 }
17615         { #2 }
17616     }
17617 }
17618 \cs_gset:Npn
17619   \markdownRendererInputRawBlockPrototype#1#2
17620 {
17621   \str_case:nnF
17622     { #2 }
17623     {
17624       { context }
17625       {
17626         \@@_plain_tex_default_input_raw_block:nn
17627         { #1 }
17628         { tex }
17629       }
17630     }
17631   {
17632     \@@_plain_tex_default_input_raw_block:nn
17633     { #1 }
17634     { #2 }
17635   }
17636 }
17637 \cs_gset_eq:NN
17638   \markdownRendererInputRawBlockPrototype
17639   \markdownRendererInputRawInlinePrototype
17640 \fi % Closes ~\markdownIfOption{plain}{\iffalse}{\iftrue}~
17641 \ExplSyntaxOff
17642 \stopmodule
17643 \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).

```

17644 \ExplSyntaxOn
17645 \str_if_eq:VVT
17646   \c_@@_top_layer_tl
17647   \c_@@_option_layer_context_tl
17648   {
17649     \use:c
17650     { ExplSyntaxOff }
17651     \@@_if_option:nF
17652     { noDefaults }
17653     {

```

```

17654     \@@_if_option:nTF
17655     { experimental }
17656     {
17657         \@@_setup:n
17658         { theme = witiko/markdown/defaults@experimental }
17659     }
17660     {
17661         \@@_setup:n
17662         { theme = witiko/markdown/defaults }
17663     }
17664 }
17665 \use:c
17666 { ExplSyntaxOn }
17667 }
17668 \ExplSyntaxOff
17669 \stopmodule
17670 \protect

```

## References

- [1] Hans Hagen. *ConT<sub>E</sub>Xt Lua Documents*. July 8, 2023. URL: <https://www.pragma-ade.nl/general/manuals/cld-mkiv.pdf> (visited on 09/22/2025).
- [2] LuaT<sub>E</sub>X development team. *LuaT<sub>E</sub>X reference manual*. Version 1.21. Feb. 1, 2025. URL: <http://mirrors.ctan.org/systems/doc/luatex/luatex.pdf> (visited on 05/12/2025).
- [3] 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).
- [4] 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).
- [5] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: <https://www.muni.cz/en/research/projects/32984> (visited on 02/19/2018).
- [6] Anton Sotkov. *File transclusion syntax for Markdown*. Jan. 19, 2017. URL: <https://github.com/iainc/Markdown-Content-Blocks> (visited on 01/08/2018).
- [7] John MacFarlane. *Pandoc. a universal document converter*. 2022. URL: <https://pandoc.org/> (visited on 10/05/2022).

- [8] 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).
- [9] 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.
- [10] Frank Mittelbach. *The doc and shortverb Packages*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/doc.pdf> (visited on 02/19/2018).
- [11] 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).
- [12] Vít Starý Novotný. *Versioned Themes*. Markdown Enhancement Proposal. Oct. 13, 2024. URL: <https://github.com/Witiko/markdown/discussions/514> (visited on 10/21/2024).
- [13] 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).
- [14] 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).
- [15] 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).
- [16] 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).
- [17] Unicode Consortium. *The Unicode Standard. Version 16.0 – Core Specification*. Sept. 10, 2024. URL: <https://www.unicode.org/versions/Unicode16.0.0/UnicodeStandard-16.0.pdf> (visited on 05/07/2025).
- [18] Roberto Ierusalimsky. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.
- [19] 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).
- [20] 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.
- [21] 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, 102
blankBeforeBlockquote	22
blankBeforeCodeFence	22
blankBeforeDivFence	22
blankBeforeHeading	23
blankBeforeList	23
bracketedSpans	23, 88, 489
breakableBlockquotes	24
cacheDir	4, 16, 19, 59, 60, 160, 191, 400, 421, 441
citationNbsps	24
citations	24, 91, 92
codeSpans	25
contentBlocks	20, 25, 35
contentBlocksLanguageMap	20
contentLevel	26
debugExtensions	9, 20, 26, 344
debugExtensionsFileName	20, 26
defaultOptions	10, 52, 398, 400
definitionLists	27, 96
depth_first_search	168
\DocumentMetadata	459
eagerCache	16, 398
ensureJekyllData	27
entities.char_entity	246
entities.dec_entity	245
entities.hex_entity	245
entities.hex_entity_with_x_char	245
escape_minimal	250
escape_programmatic_text	250
escape_typographic_text	250
expandtabs	305
expectJekyllData	27, 28
experimental	5, 17, 459
extensions	29, 168, 349
extensions.bracketed_spans	349
extensions.citations	350
extensions.content_blocks	355

<code>extensions.definition_lists</code>	358
<code>extensions.fancy_lists</code>	360
<code>extensions.fenced_code</code>	366
<code>extensions.fenced_divs</code>	371
<code>extensions.header_attributes</code>	375
<code>extensions.inline_code_attributes</code>	377
<code>extensions.jekyll_data</code>	394
<code>extensions.line_blocks</code>	378
<code>extensions.link_attributes</code>	379
<code>extensions.mark</code>	379
<code>extensions.notes</code>	381
<code>extensions.pipe_table</code>	383
<code>extensions.raw_inline</code>	388
<code>extensions.strike_through</code>	388
<code>extensions.subscripts</code>	389
<code>extensions.superscripts</code>	390
<code>extensions.tex_math</code>	390
<code>fancyLists</code>	30, 113–118, 459
<code>fencedCode</code>	31, 40, 93, 100, 119, 412, 415
<code>fencedCodeAttributes</code>	31, 86, 100, 415
<code>fencedDiv</code>	101
<code>fencedDivs</code>	32, 42
<code>finalizeCache</code>	17, 21, 32, 33, 59, 60, 159, 398, 399
<code>frozenCache</code>	21, 32, 60, 76, 79, 159, 412, 421
<code>frozenCacheCounter</code>	33, 399, 449
<code>frozenCacheFileName</code>	21, 32, 59, 399
<code>\g_markdown_diagrams_infostrings_prop</code>	417
<code>gfmAutoIdentifiers</code>	21, 33, 86, 102
<code>hashEnumerators</code>	33
<code>headerAttributes</code>	34, 42, 86, 102
<code>html</code>	34, 105, 472
<code>hybrid</code>	35, 35, 40, 46, 48, 63, 79, 120, 160, 250, 306, 449
<code>inlineCodeAttributes</code>	36, 86, 94
<code>inlineNotes</code>	36
<code>\input</code>	56, 400
<code>\inputmarkdown</code>	163, 164, 165, 490
<code>inputTempFileName</code>	61, 63, 442–444, 446
<code>\inputyaml</code>	163, 165, 490
<code>iterlines</code>	305



jeekyllData	3, 27, 28, 37, 129–132, 135
\l_file_search_path_seq	448
languages_json	355, 355
lineBlocks	38, 108
linkAttributes	37, 86, 106, 110, 325, 485
mark	38, 111, 483
\markdown	155, 156, 452
markdown	154, 154, 155, 451
markdown*	154, 154, 155, 159, 451
\markdownBegin	54, 54, 55, 56, 152–154, 156, 163
\markdownCleanup	442
\markdownConvert	441
\markdownEnd	54, 54, 55, 56, 153–156, 163
\markdownError	152, 152
\markdownEscape	54, 57, 450
\markdownIfOption	58
\markdownIfSnippetExists	81
\markdownInfo	152, 152
\markdownInput	54, 56, 154, 157, 159, 164, 447, 451
\markdownInputFilename	441
\markdownInputFileStream	442
\markdownInputPlainTeX	451
\markdownLoadPlainTeXTheme	160, 167, 411
\markdownLuaExecute	444, 447
\markdownLuaOptions	438, 441
\markdownMakeOther	152, 490
\markdownOptionFinalizeCache	59
\markdownOptionFrozenCache	59
\markdownOptionHybrid	63
\markdownOptionInputTempFileName	60
\markdownOptionNoDefaults	62
\markdownOptionOutputDir	60, 61, 63, 64
\markdownOptionPlain	61
\markdownOptionStripPercentSigns	62
\markdownOutputFileStream	442
\markdownPrepare	441
\markdownPrepareInputFilename	440
\markdownPrepareLuaOptions	438
\markdownReadAndConvert	152, 442, 451, 452, 491
\markdownReadAndConvertProcessLine	443, 444

<code>\markdownReadAndConvertStripPercentSigns</code>	443
<code>\markdownReadAndConvertTab</code>	442
<code>\markdownRendererAttributeClassName</code>	86
<code>\markdownRendererAttributeIdentifier</code>	86
<code>\markdownRendererAttributeKeyValue</code>	86
<code>\markdownRendererBlockQuoteBegin</code>	87
<code>\markdownRendererBlockQuoteEnd</code>	88
<code>\markdownRendererBracketedSpanAttributeContextBegin</code>	88
<code>\markdownRendererBracketedSpanAttributeContextEnd</code>	88
<code>\markdownRendererCite</code>	91, 92
<code>\markdownRendererCodeSpan</code>	93
<code>\markdownRendererCodeSpanAttributeContextBegin</code>	94
<code>\markdownRendererCodeSpanAttributeContextEnd</code>	94
<code>\markdownRendererContentBlock</code>	94, 95
<code>\markdownRendererContentBlockCode</code>	95
<code>\markdownRendererContentBlockOnlineImage</code>	95
<code>\markdownRendererDisplayMath</code>	126
<code>\markdownRendererDlBegin</code>	96
<code>\markdownRendererDlBeginTight</code>	96
<code>\markdownRendererDlDefinitionBegin</code>	97
<code>\markdownRendererDlDefinitionEnd</code>	98
<code>\markdownRendererDlEnd</code>	98
<code>\markdownRendererDlEndTight</code>	99
<code>\markdownRendererDlItem</code>	97
<code>\markdownRendererDlItemEnd</code>	97
<code>\markdownRendererDocumentBegin</code>	111
<code>\markdownRendererDocumentEnd</code>	111
<code>\markdownRendererEllipsis</code>	42, 99
<code>\markdownRendererEmphasis</code>	99, 139
<code>\markdownRendererError</code>	128
<code>\markdownRendererFancyOlBegin</code>	113, 114
<code>\markdownRendererFancyOlBeginTight</code>	114
<code>\markdownRendererFancyOlEnd</code>	118
<code>\markdownRendererFancyOlEndTight</code>	118
<code>\markdownRendererFancyOlItem</code>	116
<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>	102
<code>\markdownRendererHeaderAttributeContextEnd</code>	102
<code>\markdownRendererHeadingFive</code>	104
<code>\markdownRendererHeadingFour</code>	104
<code>\markdownRendererHeadingOne</code>	102
<code>\markdownRendererHeadingSix</code>	104
<code>\markdownRendererHeadingThree</code>	103
<code>\markdownRendererHeadingTwo</code>	103
<code>\markdownRendererImage</code>	106
<code>\markdownRendererImageAttributeContextBegin</code>	106
<code>\markdownRendererImageAttributeContextEnd</code>	106
<code>\markdownRendererInlineHtmlComment</code>	105
<code>\markdownRendererInlineHtmlTag</code>	105
<code>\markdownRendererInlineMath</code>	126
<code>\markdownRendererInputBlockHtmlElement</code>	105
<code>\markdownRendererInputFencedCode</code>	93
<code>\markdownRendererInputRawBlock</code>	119
<code>\markdownRendererInputRawInline</code>	118
<code>\markdownRendererInputVerbatim</code>	92
<code>\markdownRendererInterblockSeparator</code>	107
<code>\markdownRendererJekyllDataBegin</code>	129
<code>\markdownRendererJekyllDataBoolean</code>	131
<code>\markdownRendererJekyllDataEmpty</code>	135
<code>\markdownRendererJekyllDataEnd</code>	129
<code>\markdownRendererJekyllDataMappingBegin</code>	130
<code>\markdownRendererJekyllDataMappingEnd</code>	130
<code>\markdownRendererJekyllDataNumber</code>	132
<code>\markdownRendererJekyllDataProgrammaticString</code>	132, 132, 133
<code>\markdownRendererJekyllDataSequenceBegin</code>	131
<code>\markdownRendererJekyllDataSequenceEnd</code>	131
<code>\markdownRendererJekyllDataString</code>	133, 137
<code>\markdownRendererJekyllDataStringPrototype</code>	148
<code>\markdownRendererJekyllDataTypographicString</code>	132, 132, 133, 395
<code>\markdownRendererLineBlockBegin</code>	108
<code>\markdownRendererLineBlockEnd</code>	108
<code>\markdownRendererLink</code>	109, 139
<code>\markdownRendererLinkAttributeContextBegin</code>	110
<code>\markdownRendererLinkAttributeContextEnd</code>	110
<code>\markdownRendererMark</code>	111
<code>\markdownRendererNbsp</code>	112
<code>\markdownRendererNote</code>	112

<code>\markdownRendererOlBegin</code>	113
<code>\markdownRendererOlBeginTight</code>	113
<code>\markdownRendererOlEnd</code>	117
<code>\markdownRendererOlEndTight</code>	117
<code>\markdownRendererOlItem</code>	43, 114
<code>\markdownRendererOlItemEnd</code>	115
<code>\markdownRendererOlItemWithNumber</code>	42, 115
<code>\markdownRendererParagraphSeparator</code>	108
<code>\markdownRendererReplacementCharacter</code>	120
<code>\markdownRendererSectionBegin</code>	119
<code>\markdownRendererSectionEnd</code>	119
<code>\markdownRendererSoftLineBreak</code>	109
<code>\markdownRendererStrikeThrough</code>	123
<code>\markdownRendererStrongEmphasis</code>	100
<code>\markdownRendererSubscript</code>	124
<code>\markdownRendererSuperscript</code>	124
<code>\markdownRendererTable</code>	125
<code>\markdownRendererTableAttributeContextBegin</code>	125
<code>\markdownRendererTableAttributeContextEnd</code>	125
<code>\markdownRendererTextCite</code>	92
<code>\markdownRendererThematicBreak</code>	127
<code>\markdownRendererTickedBox</code>	127
<code>\markdownRendererUlBegin</code>	89
<code>\markdownRendererUlBeginTight</code>	89
<code>\markdownRendererUlEnd</code>	91
<code>\markdownRendererUlEndTight</code>	91
<code>\markdownRendererUlItem</code>	90
<code>\markdownRendererUlItemEnd</code>	90
<code>\markdownRendererUntickedBox</code>	127
<code>\markdownRendererWarning</code>	128
<code>\markdownSetup</code>	58, 58, 63, 158, 159, 165, 452, 459
<code>\markdownSetupSnippet</code>	80, 80
<code>\markdownThemeVersion</code>	70, 70
<code>\markdownWarning</code>	152, 152
<code>\markinline</code>	54, 55, 56, 154, 156, 157, 446, 450
<code>\markinlinePlainTeX</code>	450
<code>new</code>	7, 18, 398, 400
<code>notes</code>	39, 112
<code>parsers</code>	267, 304
<code>parsers.commented_line</code>	285

<code>parsers.unicode</code>	269
<code>pipeTables</code>	7, 39, 45, 125
<code>preserveTabs</code>	40, 43, 305
<code>rawAttribute</code>	35, 40, 40, 118, 119
<code>read_decompositions</code>	170
<code>reader</code>	8, 30, 168, 267, 304, 349
<code>reader-&gt;add_special_character</code>	8, 9, 30, 343
<code>reader-&gt;auto_link_email</code>	333
<code>reader-&gt;auto_link_url</code>	333
<code>reader-&gt;create_parser</code>	305
<code>reader-&gt;finalize_grammar</code>	339, 405
<code>reader-&gt;initialize_named_group</code>	344
<code>reader-&gt;insert_pattern</code>	8, 9, 30, 339, 340, 345
<code>reader-&gt;lookup_note_reference</code>	318
<code>reader-&gt;lookup_reference</code>	318
<code>reader-&gt;normalize_tag</code>	305
<code>reader-&gt;options</code>	304
<code>reader-&gt;parser_functions</code>	305
<code>reader-&gt;parser_functions.name</code>	305
<code>reader-&gt;parsers</code>	304, 304
<code>reader-&gt;register_link</code>	317
<code>reader-&gt;update_rule</code>	339, 342, 346
<code>reader-&gt;writer</code>	304
<code>reader.new</code>	304, 304, 405
<code>relativeReferences</code>	41
<code>serialize_byte_parser</code>	169
<code>serialize_byte_range_parser</code>	169
<code>serialize_replacement</code>	169
<code>\setupmarkdown</code>	165, 166
<code>\setupyaml</code>	166
<code>shiftHeadings</code>	7, 41
<code>singletonCache</code>	18
<code>slice</code>	7, 42, 247, 259, 260
<code>smartEllipses</code>	42, 99, 160
<code>\startmarkdown</code>	163, 163, 491
<code>startNumber</code>	42, 114–116
<code>\startyaml</code>	163, 163, 164, 491
<code>\stopmarkdown</code>	163, 163, 491
<code>\stopyaml</code>	163, 163, 164, 491
<code>strikeThrough</code>	43, 123, 484

stripIndent	43, 306
stripPercentSigns	442, 443
subscripts	44, 124
superscripts	44, 124
syntax	341, 345
tableAttributes	44, 125, 480
tableCaptions	7, 44, 45, 125
taskLists	45, 127, 471
texComments	46, 306
texMathDollars	36, 46, 126
texMathDoubleBackslash	36, 47, 126
texMathSingleBackslash	36, 47, 126
tightLists	47, 89, 91, 96, 99, 113, 114, 117, 118, 459
underscores	48
unicode_data.casefold_mapping	181, 192
unicode_data.categories	183, 269
unicode_data.ccc	184, 192
unicode_data.composition_mapping	177, 196
unicode_data.decomposition_mapping	171, 173, 195
unicodeNormalization	18, 19
unicodeNormalizationForm	18, 19
util.cache	187, 187
util.cache_verbatim	187
util.canonically_order	192
util.casefold	192
util.compose	196
util.decompose	195
util.encode_json_string	187
util.err	186
util.escaper	190
util.expand_tabs_in_line	187
util.find_file	199
util.find_files	199
util.flatten	188
util.intersperse	189
util.map	190
util.normalize	199
util.pathname	191
util.rope_last	189
util.rope_to_string	189

<code>util.salt</code>	191
<code>util.table_copy</code>	187
<code>util.walk</code>	188, 189
<code>util.warning</code>	191
<code>walkable_syntax</code>	8, 20, 26, 339, 340, 342–345
<code>writer</code>	168, 168, 246, 349
<code>writer-&gt;active_attributes</code>	258, 258, 259, 260
<code>writer-&gt;attribute_type_levels</code>	259
<code>writer-&gt;attributes</code>	256
<code>writer-&gt;block_html_element</code>	254
<code>writer-&gt;blockquote</code>	255
<code>writer-&gt;bulletitem</code>	253
<code>writer-&gt;bulletlist</code>	252
<code>writer-&gt;citations</code>	351
<code>writer-&gt;code</code>	251
<code>writer-&gt;contentblock</code>	355
<code>writer-&gt;defer_call</code>	267, 267
<code>writer-&gt;definitionlist</code>	358
<code>writer-&gt;display_math</code>	391
<code>writer-&gt;div_begin</code>	371
<code>writer-&gt;div_end</code>	372
<code>writer-&gt;document</code>	256
<code>writer-&gt;ellipsis</code>	249
<code>writer-&gt;emphasis</code>	254
<code>writer-&gt;error</code>	251
<code>writer-&gt;escape</code>	250
<code>writer-&gt;escaped_chars</code>	249, 250
<code>writer-&gt;escaped_minimal_strings</code>	249, 250
<code>writer-&gt;escaped_strings</code>	249
<code>writer-&gt;escaped_uri_chars</code>	249, 250
<code>writer-&gt;fancyitem</code>	361
<code>writer-&gt;fancylist</code>	360
<code>writer-&gt;fencedCode</code>	366
<code>writer-&gt;flatten_inlines</code>	247, 247
<code>writer-&gt;get_state</code>	266
<code>writer-&gt;hard_line_break</code>	248
<code>writer-&gt;heading</code>	264
<code>writer-&gt;identifier</code>	250
<code>writer-&gt;image</code>	252
<code>writer-&gt;infostring</code>	250
<code>writer-&gt;inline_html_comment</code>	254

writer->inline_html_tag	254
writer->inline_math	391
writer->interblocksep	248
writer->is_writing	247, 247
writer->jekyllData	394
writer->lineblock	378
writer->link	251
writer->mark	379
writer->math	250
writer->nbsp	247
writer->note	381
writer->options	246
writer->ordereditem	254
writer->orderedlist	253
writer->paragraph	248
writer->paragraphsep	248
writer->plain	248
writer->pop_attributes	259, 259, 260
writer->push_attributes	259, 259, 260
writer->rawBlock	367
writer->rawInline	388
writer->set_state	266
writer->slice_begin	247
writer->slice_end	247
writer->soft_line_break	248
writer->space	247
writer->span	349
writer->strike_through	389
writer->string	250
writer->strong	255
writer->subscript	389
writer->superscript	390
writer->table	385
writer->thematic_break	249
writer->checkbox	255
writer->undosep	248, 348
writer->uri	250
writer->verbatim	255
writer->warning	191, 251
writer.new	246, 246, 405
\yaml	156



yaml	154, 156, 451
\yamlBegin	54, 55, 152, 153, 156, 163
\yamlEnd	54, 55, 153, 156, 163
\yamlInput	54, 56, 154, 157, 165, 451
\yamlSetup	58