

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

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

Version 3.14.1-0-g68371ac4  
2026-03-27

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>	<b>3</b>	<b>Implementation</b>	<b>177</b>
1.1	Requirements . . . . .	2	3.1	Lua Implementation . . .	177
1.2	Feedback . . . . .	6	3.2	Plain T <sub>E</sub> X Implementation	420
1.3	Acknowledgements . . . .	7	3.3	L <sup>A</sup> T <sub>E</sub> X Implementation . .	465
<b>2</b>	<b>Interfaces</b>	<b>7</b>	3.4	ConT <sub>E</sub> Xt Implementation	511
2.1	Lua Interface . . . . .	7			
2.2	Plain T <sub>E</sub> X Interface . . . .	57	<b>References</b>		<b>522</b>
2.3	L <sup>A</sup> T <sub>E</sub> X Interface . . . . .	163	<b>Index</b>		<b>524</b>
2.4	ConT <sub>E</sub> Xt Interface . . . .	172			

## 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 . .	53
3	A sequence diagram of typesetting a document using the Lua CLI . . . . .	54
4	An example directed graph . . . . .	82
5	An example mindmap . . . . .	83
6	An example UML sequence diagram . . . . .	84
7	The banner of the Markdown package . . . . .	85
8	A pushdown automaton that recognizes T <sub>E</sub> X comments . . . . .	296

## 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             "2016-2026 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 3.3.5) 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

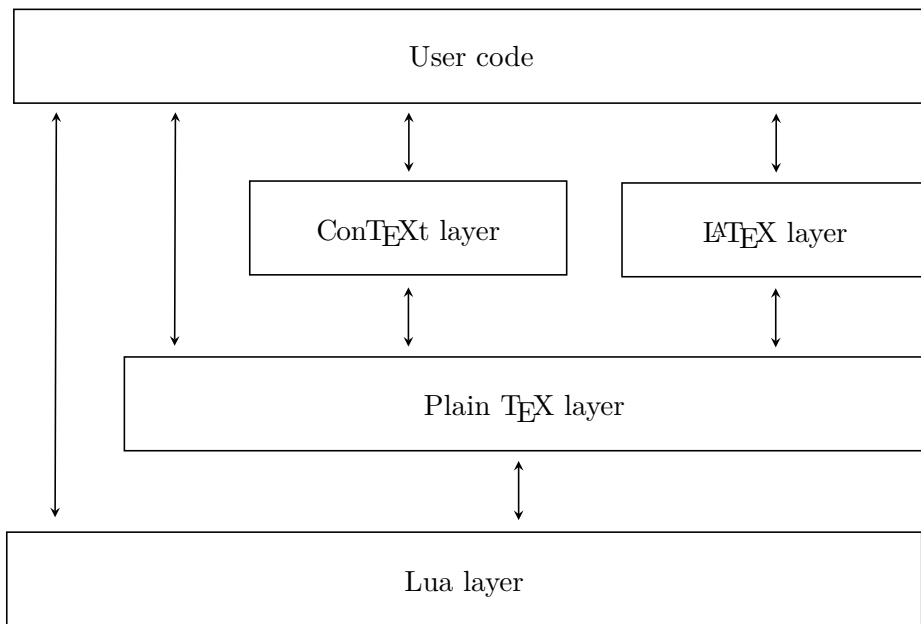
The Lua interface provides the conversion from UTF8 encoded markdown to plain T<sub>E</sub>X. This interface is used by the plain T<sub>E</sub>X implementation (see Section 3.2) and will be of interest to the developers of other packages and Lua modules.

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` and `experimentalOptions` tables.

```
89 local defaultOptions = {}
90 local experimentalOptions = {}
91 setmetatable(experimentalOptions, { __index = function (_, key)
92   return defaultOptions[key] end })
```

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

```
93 \ExplSyntaxOn
94 \seq_new:N \g_@@_lua_options_seq
```

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

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

```

125     { #4 }
126     \@@_typecheck_option:n
127     { #2 }
128 }
129 \cs_generate_variant:Nn
130 \@@_add_option:nnnn
131 { Vnnn }
132 \cs_new:Nn
133 \@@_add_experimental_lua_option:n
134 {
135     \@@_add_experimental_option:Vn
136     \c_@@_option_layer_lua_tl
137     { #1 }
138 }
139 \cs_new:Nn
140 \@@_add_experimental_option:nn
141 {
142     \seq_gput_right:cn
143     { g_@@_ #1 _options_seq }
144     { #2 }
145     \seq_gput_right:cn
146     { g_@@_experimental_ #1 _options_seq }
147     { #2 }
148     \prop_gput:cnn
149     { g_@@_ #1 _option_types_prop }
150     { #2 }
151     { boolean }
152 }
153 \cs_generate_variant:Nn
154 \@@_add_experimental_option:nn
155 { Vn }
156 \tl_const:Nn \c_@@_option_value_true_tl { true }
157 \tl_const:Nn \c_@@_option_value_false_tl { false }
158 \cs_new:Nn \@@_typecheck_option:n
159 {
160     \@@_get_option_type:nN
161     { #1 }
162     \l_tmpa_tl
163     \str_case_e:Vn
164     \l_tmpa_tl
165     {
166         { \c_@@_option_type_boolean_tl }
167         {
168             \@@_get_option_value:nN
169             { #1 }
170             \l_tmpa_tl
171             \bool_if:nF

```

```

172         {
173             \str_if_eq_p:eV
174             { \l_tmpa_tl }
175             \c_@@_option_value_true_tl ||
176             \str_if_eq_p:eV
177             { \l_tmpa_tl }
178             \c_@@_option_value_false_tl
179         }
180     {
181         \msg_error:nnne
182         { markdown }
183         { failed-typecheck-for-boolean-option }
184         { #1 }
185         { \l_tmpa_tl }
186     }
187 }
188 }
189 }
190 \msg_new:nnn
191 { markdown }
192 { failed-typecheck-for-boolean-option }
193 {
194     Option~#1~has~value~#2,~
195     but~a~boolean~(true~or~false)~was~expected.
196 }
197 \cs_generate_variant:Nn
198 \str_case_e:nn
199 { Vn }
200 \cs_generate_variant:Nn
201 \msg_error:nnnn
202 { nnne }
203 \prg_generate_conditional_variant:Nnn
204 \str_if_eq:nn
205 { eV }
206 { p }
207 \seq_new:N
208 \g_@@_option_types_seq
209 \tl_const:Nn
210 \c_@@_option_type_clist_tl
211 { clist }
212 \seq_gput_right:NV
213 \g_@@_option_types_seq
214 \c_@@_option_type_clist_tl
215 \tl_const:Nn
216 \c_@@_option_type_counter_tl
217 { counter }
218 \seq_gput_right:NV

```

```

219 \g_@@_option_types_seq
220 \c_@@_option_type_counter_tl
221 \tl_const:Nn
222 \c_@@_option_type_boolean_tl
223 { boolean }
224 \seq_gput_right:NV
225 \g_@@_option_types_seq
226 \c_@@_option_type_boolean_tl
227 \tl_const:Nn
228 \c_@@_option_type_number_tl
229 { number }
230 \seq_gput_right:NV
231 \g_@@_option_types_seq
232 \c_@@_option_type_number_tl
233 \tl_const:Nn
234 \c_@@_option_type_path_tl
235 { path }
236 \seq_gput_right:NV
237 \g_@@_option_types_seq
238 \c_@@_option_type_path_tl
239 \tl_const:Nn
240 \c_@@_option_type_slice_tl
241 { slice }
242 \seq_gput_right:NV
243 \g_@@_option_types_seq
244 \c_@@_option_type_slice_tl
245 \tl_const:Nn
246 \c_@@_option_type_string_tl
247 { string }
248 \seq_gput_right:NV
249 \g_@@_option_types_seq
250 \c_@@_option_type_string_tl
251 \cs_new:Nn
252 \@@_get_option_type:nN
253 {
254   \bool_set_false:N
255   \l_tmpa_bool
256   \seq_map_inline:Nn
257   \g_@@_option_layers_seq
258   {
259     \prop_get:cnNT
260     { g_@@_ ##1 _option_types_prop }
261     { #1 }
262     \l_tmpa_tl
263     {
264       \bool_set_true:N
265       \l_tmpa_bool

```

```

266         \seq_map_break:
267     }
268 }
269 \bool_if:NF
270   \l_tmpa_bool
271   {
272     \msg_error:nnn
273       { markdown }
274       { undefined-option }
275       { #1 }
276   }
277 \seq_if_in:NVF
278   \g_@@_option_types_seq
279   \l_tmpa_tl
280   {
281     \msg_error:nnnV
282       { markdown }
283       { unknown-option-type }
284       { #1 }
285     \l_tmpa_tl
286   }
287 \tl_set_eq:NN
288   #2
289   \l_tmpa_tl
290 }
291 \msg_new:nnn
292   { markdown }
293   { unknown-option-type }
294   {
295     Option~#1~has~unknown~type~#2.
296   }
297 \msg_new:nnn
298   { markdown }
299   { undefined-option }
300   {
301     Option~#1~is~undefined.
302   }
303 \cs_generate_variant:Nn
304   \msg_error:nnnn
305   { nnnV }
306 \cs_new:Nn
307   \@@_get_default_option_value:nN
308   {
309     \bool_set_false:N
310       \l_tmpa_bool
311     \seq_map_inline:Nn
312       \g_@@_option_layers_seq

```

```

313     {
314         \seq_if_in:cnT
315         { g__markdown_experimental_ ##1 _options_seq }
316         { #1 }
317         {
318             \@@_get_option_value:nN
319             { experimental }
320             #2
321             \bool_set_true:N
322             \l_tmpa_bool
323             \seq_map_break:
324         }
325         \prop_get:cnNT
326         { g_@@_default_ ##1 _options_prop }
327         { #1 }
328         #2
329         {
330             \bool_set_true:N
331             \l_tmpa_bool
332             \seq_map_break:
333         }
334     }
335     \bool_if:NF
336     \l_tmpa_bool
337     {
338         \msg_error:nnn
339         { markdown }
340         { undefined-option }
341         { #1 }
342     }
343 }
344 \cs_new:Nn
345 \@@_get_option_value:nN
346 {
347     \@@_option_tl_to_csname:nN
348     { #1 }
349     \l_tmpa_tl
350     \cs_if_free:cTF
351     { \l_tmpa_tl }
352     {
353         \@@_get_default_option_value:nN
354         { #1 }
355         #2
356     }
357     {
358         \@@_get_option_type:nN
359         { #1 }

```

```

360         \l_tmpa_tl
361     \str_if_eq:NNTF
362         \c_@@_option_type_counter_tl
363         \l_tmpa_tl
364     {
365         \@@_option_tl_to_csname:nN
366         { #1 }
367         \l_tmpa_tl
368         \tl_set:Nx
369             #2
370         { \the \cs:w \l_tmpa_tl \cs_end: }
371     }
372     {
373         \@@_option_tl_to_csname:nN
374         { #1 }
375         \l_tmpa_tl
376         \tl_set:Nv
377             #2
378         { \l_tmpa_tl }
379     }
380 }
381 }
382 \cs_new:Nn \@@_option_tl_to_csname:nN
383 {
384     \tl_set:Nn
385         \l_tmpa_tl
386         { \str_uppercase:n { #1 } }
387     \tl_set:Nx
388         #2
389         {
390             markdownOption
391             \tl_head:f { \l_tmpa_tl }
392             \tl_tail:n { #1 }
393         }
394     }
395 \regex_const:Nn
396     \c_@@_option_regex
397     { ^markdownOption }
398 \prg_new_conditional:Nnn
399     \@@_csname_to_option_str:nN
400     { T }
401     {
402         \tl_if_regex_match:nNTF
403             { #1 }
404             \c_@@_option_regex
405             {
406                 \tl_set:Nn

```



```

407         \l_tmpa_tl
408         { #1 }
409     \tl_regex_replace_once:NNn
410     \l_tmpa_tl
411     \c_@@_option_regex
412     { }
413     \tl_set:Nn
414     \l_tmpb_tl
415     {
416         \str_lowercase:f
417         { \l_tmpa_tl }
418     }
419     \str_set:Nx
420     #2
421     {
422         \tl_head:f { \l_tmpb_tl }
423         \tl_tail:V \l_tmpa_tl
424     }
425     \prg_return_true:
426 }
427 { \prg_return_false: }
428 }

```

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.

```

429 \cs_new:Nn \@@_with_various_cases:nn
430 {
431     \seq_clear:N
432     \l_tmpa_seq
433     \seq_map_inline:Nn
434     \g_@@_cases_seq
435     {
436         \tl_set:Nn
437         \l_tmpa_tl
438         { #1 }
439         \use:c { ##1 }
440         \l_tmpa_tl
441         \seq_put_right:NV
442         \l_tmpa_seq
443         \l_tmpa_tl
444     }
445     \seq_map_inline:Nn
446     \l_tmpa_seq
447     { #2 }
448 }

```

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

```

449 \seq_new:N \g_@@_cases_seq
450 \cs_new:Nn \@@_camel_case:N
451   {
452     \regex_replace_all:nnN
453       { _ ([a-z]) }
454       { \c { str_uppercase:n } \cB\{ \1 \cE\} }
455       #1
456     \tl_set:Nx
457       #1
458       { #1 }
459   }
460 \seq_gput_right:Nn \g_@@_cases_seq { @@_camel_case:N }
461 \cs_new:Nn \@@_snake_case:N
462   {
463     \regex_replace_all:nnN
464       { ([a-z])([A-Z]) }
465       { \1 _ \c { str_lowercase:n } \cB\{ \2 \cE\} }
466       #1
467     \tl_set:Nx
468       #1
469       { #1 }
470   }
471 \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.

```
472 \@@_add_lua_option:nnn
473   { eagerCache }
474   { boolean }
475   { true }
476 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 has the following effects:

1. The version `experimental` of the theme `witiko/markdown/defaults` will be loaded.
2. Warnings for hard-deprecated features will become errors.
3. The experimental option `htmlOverLinks` will be enabled.

In the future, the effects may extend to other areas as well.

**false** Experimental features will be disabled.

```
477 \@@_add_lua_option:nnn
478   { experimental }
479   { boolean }
480   { false }
481 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.

```
482 \@@_add_lua_option:nnn
483 { singletonCache }
484 { boolean }
485 { true }

486 defaultOptions.singletonCache = true

487 local singletonCache = {
488   convert = nil,
489   options = nil,
490 }
```

`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.

```
491 \@@_add_lua_option:nnn
492 { unicodeNormalization }
493 { boolean }
494 { true }

495 defaultOptions.unicodeNormalization = true
```

`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.

---

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

<sup>8</sup>See <https://unicode.org/faq/normalization.html>.

**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.

```

496 \@@_add_lua_option:nnn
497   { unicodeNormalizationForm }
498   { string }
499   { nfc }

500 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.

```

501 \str_new:N
502   \g_@@_unquoted_jobname_str
503 \str_gset:NV
504   \g_@@_unquoted_jobname_str
505   \c_sys_jobname_str
506 \bool_new:N
507   \g_@@_jobname_quoted_bool
508 \regex_replace_all:nnNTF
509   { \A ("|') ( .* ) ("|') \Z }
510   { \2 }
511   \g_@@_unquoted_jobname_str
512   {
513     \bool_gset_true:N
514     \g_@@_jobname_quoted_bool
515   }
516   {
517     \bool_gset_false:N
518     \g_@@_jobname_quoted_bool

```

```

519     }
520     \@@_add_lua_option:nnn
521     { cacheDir }
522     { path }
523     {
524         \markdownOptionOutputDir
525         / _markdown_
526         \str_use:N
527         \g_@@_unquoted_jobname_str
528     }
529     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 2.2.5.11 for more information.

```

530     \@@_add_lua_option:nnn
531     { contentBlocksLanguageMap }
532     { path }
533     { markdown-languages.json }
534     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.

```

535     \@@_add_lua_option:nnn
536     { debugExtensionsFileName }
537     { path }
538     {
539         \markdownOptionOutputDir
540         /
541         \str_use:N
542         \g_@@_unquoted_jobname_str
543         .debug-extensions.json
544     }
545     defaultOptions.debugExtensionsFileName = "debug-extensions.json"

```

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

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

The frozen cache makes it possible to later typeset a plain  $\text{\TeX}$  document that contains markdown documents without invoking Lua using the `frozenCache` plain  $\text{\TeX}$  option. As a result, the plain  $\text{\TeX}$  document becomes more portable, but further changes in the order and the content of markdown documents will not be reflected.

```
546 \@@_add_lua_option:nnn
547   { frozenCacheFileName }
548   { path }
549   { \markdownOptionCacheDir / frozenCache.tex }
550 defaultOptions.frozenCacheFileName = "frozenCache.tex"
```

### 2.1.6 Parser Options

`acronyms`= $\langle acronyms \rangle$

Acronyms, initialisms, and other all-caps sequences that will be recognized in markdown text and formatted accordingly.

```
551 \cs_generate_variant:Nn
552   \@@_add_lua_option:nnn
553   { nnV }
554 \@@_add_lua_option:nnV
555   { acronyms }
556   { clist }
557   \c_empty_clist
558 defaultOptions.acronyms = {}
```

`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.

---

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

See also the option [gfmAutoIdentifiers](#).

```
559 \@@_add_lua_option:nnn
560   { autoIdentifiers }
561   { boolean }
562   { false }

563 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.

```
564 \@@_add_lua_option:nnn
565   { blankBeforeBlockquote }
566   { boolean }
567   { false }

568 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.

```
569 \@@_add_lua_option:nnn
570   { blankBeforeCodeFence }
571   { boolean }
572   { false }

573 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.

```
574 \@@_add_lua_option:nnn
575   { blankBeforeDivFence }
576   { boolean }
577   { false }

578 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.

```
579 \@@_add_lua_option:nnn
580 { blankBeforeHeading }
581 { boolean }
582 { false }

583 defaultOptions.blankBeforeHeading = false
```

`blankBeforeHtmlBlock=true, false` default: false

- `true`      Require a blank line between a paragraph and the following CommonMark HTML block<sup>10</sup>.
- `false`     Do not require a blank line between a paragraph and the following CommonMark HTML block.

```
584 \@@_add_lua_option:nnn
585 { blankBeforeHtmlBlock }
586 { boolean }
587 { false }

588 defaultOptions.blankBeforeHtmlBlock = 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.

```
589 \@@_add_lua_option:nnn
590 { blankBeforeList }
591 { boolean }
592 { false }

593 defaultOptions.blankBeforeList = false
```

---

<sup>10</sup>See <https://spec.commonmark.org/0.31.2/#html-blocks>.

`bracketedSpans=true, false` default: false

**true** Enable the Pandoc bracketed span syntax extension<sup>11</sup>:

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

**false** Disable the Pandoc bracketed span syntax extension.

```
594 \@@_add_lua_option:nnn
595   { bracketedSpans }
596   { boolean }
597   { false }

598 defaultOptions.bracketedSpans = false
```

`breakableBlockquotes=true, false` default: true

**true** A blank line separates block quotes.

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

```
599 \@@_add_lua_option:nnn
600   { breakableBlockquotes }
601   { boolean }
602   { true }

603 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.

```
604 \@@_add_lua_option:nnn
605   { citationNbsps }
606   { boolean }
607   { true }

608 defaultOptions.citationNbsps = true
```

---

<sup>11</sup>See [https://pandoc.org/MANUAL.html#extension-bracketed\\_spans](https://pandoc.org/MANUAL.html#extension-bracketed_spans).

`citations=true, false`

default: false

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

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

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

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

`false` Disable the Pandoc citation syntax extension.

```
609 \@@_add_lua_option:nnn
610   { citations }
611   { boolean }
612   { false }
613 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.``

```
614 \@@_add_lua_option:nnn
615   { codeSpans }
616   { boolean }
617   { true }
618 defaultOptions.codeSpans = true
```

<sup>12</sup>See <https://pandoc.org/MANUAL.html#extension-citations>.

`contentBlocks=true, false`

default: false

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

```
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.

```
619 \@@_add_lua_option:nnn
620 { contentBlocks }
621 { boolean }
622 { false }

623 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
```

```
624 \@@_add_lua_option:nnn
625 { contentLevel }
626 { string }
627 { block }

628 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.

```
629 \@@_add_lua_option:nnn
630 { debugExtensions }
631 { boolean }
632 { false }

633 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.

```
634 \@@_add_lua_option:nnn
635 { definitionLists }
636 { boolean }
637 { false }

638 defaultOptions.definitionLists = false
```

`ensureJekyllData=true, false`

default: false

- 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.
- 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.

See also the plain  $\text{\TeX}$  macros `\yamlBegin`, `\yamlEnd`, and `\yamlInput` in Section 2.2.1, the  $\text{\LaTeX}$  environment `yaml` in Section 2.3.1, and the  $\text{\ConTeXt}$  macros `\startyaml`, `\stopyaml`, and `\inputyaml` in Section 2.4.1 to see how the options `jekyllData`, `expectJekyllData`, and `ensureJekyllData` can be used together to implement high-level interfaces for processing YAML documents.

```
639 \@@_add_lua_option:nnn
640   { ensureJekyllData }
641   { boolean }
642   { false }

643 defaultOptions.ensureJekyllData = false
```

`expectJekyllData=true, false`

default: false

- 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}

```

`false`

When the `jeekyllData` 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[jeekyllData]{markdown}
\begin{document}
\begin{markdown}
---
- this
- is
- YAML
...
- followed
- by
- Markdown
\end{markdown}
\begin{markdown}
- this
- is
- Markdown
\end{markdown}
\end{document}

```

See also the plain  $\text{\TeX}$  macros `\yamlBegin`, `\yamlEnd`, and `\yamlInput` in Section 2.2.1, the  $\text{\LaTeX}$  environment `yaml` in Section 2.3.1, and the  $\text{\ConTeXt}$  macros `\startyaml`, `\stopyaml`, and `\inputyaml` in Section 2.4.1 to see how the options `jeekyllData`, `expectJekyllData`, and `ensureJekyllData` can be used together to implement high-level interfaces for processing YAML documents.

```

644 \@@_add_lua_option:nnn
645   { expectJekyllData }
646   { boolean }
647   { false }

648 defaultOptions.expectJekyllData = false

```

`extensions=` $\langle filenames \rangle$

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

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

```
local strike_through = {
  api_version = 2,
  grammar_version = 4,
  finalize_grammar = function(reader)
    local nonspacechar = lpeg.P(1) - lpeg.S("\t ")
    local doubleslashes = lpeg.P("//")
    local function between(p, starter, ender)
      ender = lpeg.B(nonspacechar) * ender
      return (starter * #nonspacechar
        * lpeg.Ct(p * (p - ender)^0) * ender)
    end

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

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

return strike_through
```

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

```
649 metadata.user_extension_api_version = 2
650 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.

```
651 \@@_add_lua_option:nnV
652   { extensions }
653   { clist }
654   \c_empty_clist
655 defaultOptions.extensions = {}
```

`fancyLists=true, false`

default: false

`true` Enable the Pandoc fancy list syntax extension<sup>13</sup>:

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

`false` Disable the Pandoc fancy list syntax extension.

```
656 \@@_add_lua_option:nnn
657   { fancyLists }
658   { boolean }
659   { false }
660 defaultOptions.fancyLists = false
```

`fencedCode=true, false`

default: true

`true` Enable the commonmark fenced code block extension:

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

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

<sup>13</sup>See <https://pandoc.org/MANUAL.html#org-fancy-lists>.

```
</code>
</pre>
...
```

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

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

665 defaultOptions.fencedCode = true
```

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

**true**      Enable the Pandoc fenced code attribute syntax extension<sup>14</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.

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

670 defaultOptions.fencedCodeAttributes = false
```

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

**true**      Enable the Pandoc fenced div syntax extension<sup>15</sup>:

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

And another.
:::::
```

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

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

<sup>15</sup>See [https://pandoc.org/MANUAL.html#extension-fenced\\_divs](https://pandoc.org/MANUAL.html#extension-fenced_divs).

```

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

675 defaultOptions.fencedDivs = false

```

**frozenCache**=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.

```

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

680 defaultOptions.finalizeCache = false

```

**frozenCacheCounter**=*<number>*

default: 0

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

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

```

681 \@@_add_lua_option:nnn
682   { frozenCacheCounter }
683   { counter }
684   { 0 }

685 defaultOptions.frozenCacheCounter = 0

```

`gfmAutoIdentifiers=true, false` default: false

**true** Enable the Pandoc GitHub-flavored auto identifiers syntax extension<sup>16</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`.

```
686 \@@_add_lua_option:nnn
687 { gfmAutoIdentifiers }
688 { boolean }
689 { false }
690 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.

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

`headerAttributes=true, false` default: false

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

`# My first heading {#foo}`

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

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

---

<sup>16</sup>See [https://pandoc.org/MANUAL.html#extension-gfm\\_auto\\_identifiers](https://pandoc.org/MANUAL.html#extension-gfm_auto_identifiers).

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

```
696 \@@_add_lua_option:nnn
697   { headerAttributes }
698   { boolean }
699   { false }

700 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.

```
701 \@@_add_lua_option:nnn
702   { html }
703   { boolean }
704   { true }

705 defaultOptions.html = true
```

**htmlOverLinks=true, false**      default: false

**true**      When the option **html** is enabled and a text can be understood either as a hyperlink or HTML, interpret it as HTML.  
This is especially relevant when the option **relativeReferences** is enabled, since it makes e.g. `</foo>` a valid hyperlink.

**false**      When the option **html** is enabled and a text can be understood either as a hyperlink or HTML, interpret it as a hyperlink.

This is an experimental option. Whenever the option **experimental** is enabled and this option is unspecified, it will be enabled. Like other experimental options, this option will be enabled by default and soft-deprecated in the next major release of the Markdown package.

```
706 \@@_add_experimental_lua_option:n
707   { htmlOverLinks }

708 defaultOptions.htmlOverLinks = false
709 experimentalOptions.htmlOverLinks = 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 T<sub>E</sub>X commands between dollar signs or backslash-escaped brackets:

```
$H_2 O$ is a liquid.

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

```
710 \@@_add_lua_option:nnn
711   { hybrid }
712   { boolean }
713   { false }
714 defaultOptions.hybrid = false
```

`inlineCodeAttributes=true, false` default: false

**true**      Enable the Pandoc inline code span attribute extension<sup>17</sup>:

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

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

```
715 \@@_add_lua_option:nnn
716   { inlineCodeAttributes }
717   { boolean }
718   { false }
719 defaultOptions.inlineCodeAttributes = false
```

`inlineNotes=true, false` default: false

**true**      Enable the Pandoc inline note syntax extension<sup>18</sup>:

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

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

<sup>18</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.

```
720 \@@_add_lua_option:nnn
721   { inlineNotes }
722   { boolean }
723   { false }

724 defaultOptions.inlineNotes = false
```

**jeekyllData=true, false**      default: false

**true**      Enable the Pandoc YAML metadata block syntax extension<sup>19</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.

See also the plain T<sub>E</sub>X macros `\yamlBegin`, `\yamlEnd`, and `\yamlInput` in Section 2.2.1, the L<sup>A</sup>T<sub>E</sub>X environment `yaml` in Section 2.3.1, and the ConT<sub>E</sub>Xt macros `\startyaml`, `\stopyaml`, and `\inputyaml` in Section 2.4.1 to see how the options `jeekyllData`, `expectJeekyllData`, and `ensureJeekyllData` can be used together to implement high-level interfaces for processing YAML documents.

```
725 \@@_add_lua_option:nnn
726   { jeekyllData }
727   { boolean }
728   { false }

729 defaultOptions.jekyllData = false
```

---

<sup>19</sup>See [https://pandoc.org/MANUAL.html#extension-yaml\\_metadata\\_block](https://pandoc.org/MANUAL.html#extension-yaml_metadata_block).



`linkAttributes=true, false`

default: false

**true** Enable the Pandoc link and image attribute syntax extension<sup>20</sup>:

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

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

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

```
730 \@@_add_lua_option:nnn
731   { linkAttributes }
732   { boolean }
733   { false }

734 defaultOptions.linkAttributes = false
```

`lineBlocks=true, false`

default: false

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

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

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

```
735 \@@_add_lua_option:nnn
736   { lineBlocks }
737   { boolean }
738   { false }

739 defaultOptions.lineBlocks = false
```

---

<sup>20</sup>See [https://pandoc.org/MANUAL.html#extension-link\\_attributes](https://pandoc.org/MANUAL.html#extension-link_attributes).

<sup>21</sup>See [https://pandoc.org/MANUAL.html#extension-line\\_blocks](https://pandoc.org/MANUAL.html#extension-line_blocks).

`mark=true, false`

default: false

`true` Enable the Pandoc mark syntax extension<sup>22</sup>:

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

`false` Disable the Pandoc mark syntax extension.

```
740 \@@_add_lua_option:nnn
741   { mark }
742   { boolean }
743   { false }
744 defaultOptions.mark = false
```

`notes=true, false`

default: false

`true` Enable the Pandoc note syntax extension<sup>23</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.

```
745 \@@_add_lua_option:nnn
746   { notes }
747   { boolean }
748   { false }
749 defaultOptions.notes = false
```

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

<sup>23</sup>See <https://pandoc.org/MANUAL.html#extension-footnotes>.

`pipeTables=true, false`

default: false

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

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

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

```
750 \@@_add_lua_option:nnn
751 { pipeTables }
752 { boolean }
753 { false }
754 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.

```
755 \@@_add_lua_option:nnn
756 { preserveTabs }
757 { boolean }
758 { true }
759 defaultOptions.preserveTabs = true
```

`rawAttribute=true, false`

default: false

**true** Enable the Pandoc raw attribute syntax extension<sup>24</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 \\
```

<sup>24</sup>See [https://pandoc.org/MANUAL.html#extension-raw\\_attribute](https://pandoc.org/MANUAL.html#extension-raw_attribute).

```

        c & d
    \end{dcases}
\]
...

```

The `rawAttribute` option is a good alternative to the `hybrid` option. Unlike the `hybrid` option, which affects the entire document, the `rawAttribute` option allows you to isolate the parts of your documents that use T<sub>E</sub>X.

`false`      Disable the Pandoc raw attribute syntax extension.

```

760 \@@_add_lua_option:nnn
761   { rawAttribute }
762   { boolean }
763   { false }
764 defaultOptions.rawAttribute = false

```

`relativeReferences=true, false`

default: `false`

`true`      Enable relative references<sup>25</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.

```

765 \@@_add_lua_option:nnn
766   { relativeReferences }
767   { boolean }
768   { false }
769 defaultOptions.relativeReferences = false

```

<sup>25</sup>See <https://datatracker.ietf.org/doc/html/rfc3986#section-4.2>.

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

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

```
770 \@@_add_lua_option:nnn
771   { shiftHeadings }
772   { number }
773   { 0 }
774 defaultOptions.shiftHeadings = 0
```

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

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

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

```
775 \@@_add_lua_option:nnn
776   { slice }
777   { slice }
778   {  $\sim \$$  }
779 defaultOptions.slice = " $\sim \$$ "
```

`smartEllipses`=true, false default: false

- |                    |  |
|--------------------|--|
| <code>true</code>  | Convert any ellipses in the input to the <code>\markdownRenderEllipsis</code> $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ macro. |
| <code>false</code> | Preserve all ellipses in the input.  |

```

780 \@@_add_lua_option:nnn
781   { smartEllipses }
782   { boolean }
783   { false }
784 defaultOptions.smartEllipses = false

```

`startNumber=true, false`

default: true

- true**      Make the number in the first item of an ordered lists significant. The item numbers will be passed to the `\markdownRendererOliItemWithNumber` T<sub>E</sub>X macro.
- false**     Ignore the numbers in the ordered list items. Each item will only produce a `\markdownRendererOliItem` T<sub>E</sub>X macro.

```

785 \@@_add_lua_option:nnn
786   { startNumber }
787   { boolean }
788   { true }
789 defaultOptions.startNumber = true

```

`strikeThrough=true, false`

default: false

- true**      Enable the Pandoc strike-through syntax extension<sup>26</sup>:

This ~~is deleted text.~~

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

```

790 \@@_add_lua_option:nnn
791   { strikeThrough }
792   { boolean }
793   { false }
794 defaultOptions.strikeThrough = false

```

---

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

`stripIndent=true, false`

default: false

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

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

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

```
795 \@@_add_lua_option:nnn
796   { stripIndent }
797   { boolean }
798   { false }
799 defaultOptions.stripIndent = false
```

`subscripts=true, false`

default: false

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

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

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

```
800 \@@_add_lua_option:nnn
801   { subscripts }
802   { boolean }
803   { false }
804 defaultOptions.subscripts = false
```

---

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

`superscripts=true, false` default: false

**true** Enable the Pandoc superscript syntax extension<sup>28</sup>:

`210` is 1024.

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

```
805 \@@_add_lua_option:nnn
806 { superscripts }
807 { boolean }
808 { false }

809 defaultOptions.superscripts = false
```

`tableAttributes=true, false` default: false

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

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

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

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

```
810 \@@_add_lua_option:nnn
811 { tableAttributes }
812 { boolean }
813 { false }

814 defaultOptions.tableAttributes = false
```

---

<sup>28</sup>See <https://pandoc.org/MANUAL.html#extension-superscript-subscript>.



`tableCaptions=true, false`

default: false

**true** Enable the Pandoc table caption syntax extension<sup>29</sup> for pipe tables (see the `pipeTables` option):

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

: Demonstration of pipe table syntax.

**false** Disable the Pandoc table caption syntax extension.

```
815 \@@_add_lua_option:nnn
816   { tableCaptions }
817   { boolean }
818   { false }
819 defaultOptions.tableCaptions = false
```

`taskLists=true, false`

default: false

**true** Enable the Pandoc task list syntax extension<sup>30</sup>:

- [ ] an unticked task list item
- [X] a ticked task list item

Our implementation also supports half-ticked task list items:

- [/] a half-checked task list item
- [.] another half-checked task list item

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

```
820 \@@_add_lua_option:nnn
821   { taskLists }
822   { boolean }
823   { false }
824 defaultOptions.taskLists = false
```

<sup>29</sup>See [https://pandoc.org/MANUAL.html#extension-table\\_captions](https://pandoc.org/MANUAL.html#extension-table_captions).

<sup>30</sup>See [https://pandoc.org/MANUAL.html#extension-task\\_lists](https://pandoc.org/MANUAL.html#extension-task_lists).

`texComments=true, false`

default: false

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

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

Always enabled when `hybrid` is enabled.

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

```
825 \@@_add_lua_option:nnn
826   { texComments }
827   { boolean }
828   { false }
829 defaultOptions.texComments = false
```

`texMathDollars=true, false`

default: false

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

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

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

```
830 \@@_add_lua_option:nnn
831   { texMathDollars }
832   { boolean }
833   { false }
834 defaultOptions.texMathDollars = false
```

---

<sup>31</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>32</sup>:

<pre>inline math: \\\(E=mc^2\\) display math: \\[E=mc^2\\]</pre>
--

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

```
835 \@@_add_lua_option:nnn
836   { texMathDoubleBackslash }
837   { boolean }
838   { false }
839 defaultOptions.texMathDoubleBackslash = false
```

`texMathSingleBackslash=true, false` default: false

**true** Enable the Pandoc single backslash math syntax extension<sup>33</sup>:

<pre>inline math: \\\(E=mc^2\\) display math: \\[E=mc^2\\]</pre>
--

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

```
840 \@@_add_lua_option:nnn
841   { texMathSingleBackslash }
842   { boolean }
843   { false }
844 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>32</sup>See [https://pandoc.org/MANUAL.html#extension-tex\\_math\\_double\\_backslash](https://pandoc.org/MANUAL.html#extension-tex_math_double_backslash).

<sup>33</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.

```

845 \@@_add_lua_option:nnn
846   { tightLists }
847   { boolean }
848   { true }

849 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.

```

850 \@@_add_lua_option:nnn
851   { underscores }
852   { boolean }
853   { true }
854 \ExplSyntaxOff

855 defaultOptions.underscores = true

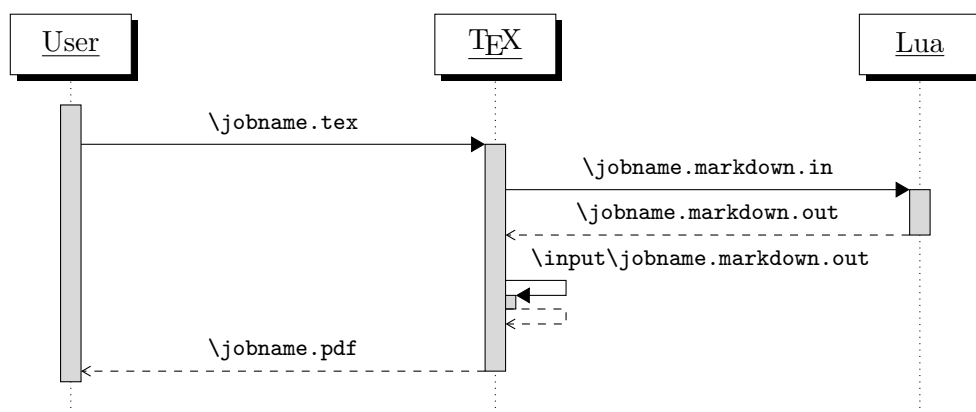
```

### 2.1.7 Command-Line Interface

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

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

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

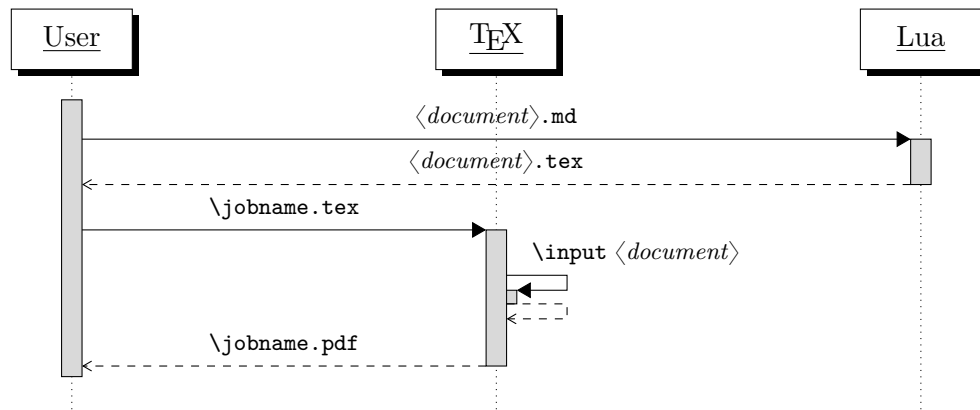


**Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the T<sub>E</sub>X interface**

```

856 .TH MARKDOWN2TEX 1 "(((LASTMODIFIED)))"
857 .SH NAME
858 markdown2tex \- convert .md files to .tex
859 .SH SYNOPSIS
860 local HELP_STRING = "Usage: " .. [[
861 markdown2tex [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
862
863 .SH DESCRIPTION
864 OPTIONS are documented in Section 2.2.1 of the Markdown Package User
865 Manual (https://ctan.org/pkg/markdown).
866

```



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

867 When OUTPUT\_FILE is unspecified, the result of the conversion will be  
 868 written to the standard output. When INPUT\_FILE is also unspecified, the  
 869 result of the conversion will be read from the standard input.

870

871 Report bugs to: `witiko@mail.muni.cz`

872 Markdown package home page: `<https://github.com/witiko/markdown>]]`

873

874 `local VERSION_STRING = [[`

875 `markdown2tex (Markdown) ]] .. metadata.version .. [[`

876

877 `Copyright (C) ]] .. table.concat(metadata.copyright,`

878 `"\nCopyright (C) ") .. [[`

879

880 `License: ]] .. metadata.license`

881

882 `local function warn(s)`

883 `io.stderr:write("Warning: " .. s .. "\n")`

884 `end`

885

886 `local function error(s)`

887 `io.stderr:write("Error: " .. s .. "\n")`

888 `os.exit(1)`

889 `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.

890 `local function camel_case(option_name)`

```

891  local cased_option_name = option_name:gsub("_(%l)", function(match)
892      return match:sub(2, 2):upper()
893  end)
894  return cased_option_name
895 end
896
897 local function snake_case(option_name)
898     local cased_option_name = option_name:gsub("%l%u", function(match)
899         return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()
900     end)
901     return cased_option_name
902 end
903
904 local cases = {camel_case, snake_case}
905 local various_case_options = {}
906 for option_name, _ in pairs(defaultOptions) do
907     for _, case in ipairs(cases) do
908         various_case_options[case(option_name)] = option_name
909     end
910 end
911
912 local process_options = true
913 local options = {}
914 local input_filename
915 local output_filename
916 for i = 1, #arg do
917     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.

```

918         if arg[i] == "--" then
919             process_options = false
920             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.

```

921         elseif arg[i]:match("=") then
922             local key, value = arg[i]:match("(.)=(.*)")
923             if defaultOptions[key] == nil and
924                 various_case_options[key] ~= nil then
925                 key = various_case_options[key]
926             end

```

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

```

927     local default_type = type(defaultOptions[key])
928     if default_type == "boolean" then
929         options[key] = (value == "true")
930     elseif default_type == "number" then
931         options[key] = tonumber(value)
932     elseif default_type == "table" then
933         options[key] = {}
934         for item in value:gmatch("[^,]+") do
935             table.insert(options[key], item)
936         end
937     else
938         if default_type ~= "string" then
939             if default_type == "nil" then
940                 warn('Option "' .. key .. '" not recognized.')
941             else
942                 warn('Option "' .. key .. '" type not recognized, ' ..
943                     'please file a report to the package maintainer.')
944             end
945             warn('Parsing the ' .. 'value "' .. value .. '" of option "' ..
946                 key .. '" as a string.')
947         end
948         options[key] = value
949     end
950     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.

```

951     elseif arg[i] == "--help" or arg[i] == "-h" then
952         print(HELP_STRING)
953         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.

```

954     elseif arg[i] == "--version" or arg[i] == "-v" then
955         print(VERSION_STRING)
956         os.exit()
957     end
958 end

```

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

```

959 if input_filename == nil then
960     input_filename = arg[i]

```



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

```
961 elseif output_filename == nil then
962   output_filename = arg[i]
963 else
964   error('Unexpected argument: "' .. arg[i] .. '".')
965 end
966 ::continue::
967 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  $\text{\TeX}$  document `hello.tex`. After the Markdown package for our  $\text{\TeX}$  format has been loaded, the converted document can be typeset as follows:

```
\input hello
```

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

The plain  $\text{\TeX}$  interface provides macros for the typesetting of markdown input from within plain  $\text{\TeX}$ , for setting the Lua interface options (see Section 2.1.3) used during the conversion from markdown to plain  $\text{\TeX}$  and for changing the way markdown the tokens are rendered.

```
968 \def\markdownLastModified{((LASTMODIFIED))}%
969 \def\markdownVersion{((VERSION))}%
```

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

```
\input markdown
```

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

```
970 \let\markdownBegin\relax
971 \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.

```

972 \let\yamlBegin\relax
973 \def\yamlEnd{\markdownEnd\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
\yamlBegin
title: _Hello_ **world** ...
author: John Doe
\yamlEnd
\bye

```

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

```

\input markdown
\yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}
\markdownBegin
title: _Hello_ **world** ...
author: John Doe
\markdownEnd
\bye

```

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

```

974 \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.

```
975 \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.

```
976 \def\yamlInput#1{%
977   \begingroup
978   \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
979   \markdownInput{#1}%
980   \endgroup
981 }%
```

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.

```
982 \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.

```
983 \ExplSyntaxOn
984 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
985 \cs_generate_variant:Nn
986   \tl_const:Nn
987   { NV }
988 \tl_if_exist:NF
989   \c_@@_top_layer_tl
990   {
991     \tl_const:NV
992       \c_@@_top_layer_tl
993       \c_@@_option_layer_plain_tex_tl
994   }
```

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

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

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

```
996 \prop_new:N \g_@@_plain_tex_option_types_prop
997 \prop_new:N \g_@@_default_plain_tex_options_prop
998 \seq_new:N \g_@@_experimental_plain_tex_options_seq
999 \seq_gput_right:NV
1000   \g_@@_option_layers_seq
1001   \c_@@_option_layer_plain_tex_tl
1002 \cs_new:Nn
1003   \@@_add_plain_tex_option:nnn
1004   {
1005     \@@_add_option:Vnnn
```

```

1006      \c_@@_option_layer_plain_tex_tl
1007      { #1 }
1008      { #2 }
1009      { #3 }
1010  }

```

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.

```

1011 \cs_new:Nn
1012   \@@_setup:n
1013   {
1014     \keys_set:nn
1015       { markdown/options }
1016       { #1 }
1017   }
1018 \cs_gset_eq:NN
1019   \markdownSetup
1020   \@@_setup:n

```

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

```

1021 \cs_gset_eq:NN
1022   \yamlSetup
1023   \markdownSetup

```

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

```

1024 \prg_new_conditional:Nnn
1025   \@@_if_option:n
1026   { TF, T, F }
1027   {
1028     \@@_get_option_type:nN
1029       { #1 }
1030       \l_tmpa_tl
1031     \str_if_eq:NNF
1032       \l_tmpa_tl
1033       \c_@@_option_type_boolean_tl
1034     {
1035       \msg_error:nnxx
1036         { markdown }
1037         { expected-boolean-option }
1038         { #1 }
1039         { \l_tmpa_tl }

```

```

1040     }
1041     \@@_get_option_value:nN
1042     { #1 }
1043     \l_tmpa_tl
1044     \str_if_eq:NNTF
1045     \l_tmpa_tl
1046     \c_@@_option_value_true_tl
1047     { \prg_return_true: }
1048     { \prg_return_false: }
1049   }
1050   \msg_new:nnn
1051   { markdown }
1052   { expected-boolean-option }
1053   {
1054     Option~#1~has~type~#2,~
1055     but~a~boolean~was~expected.
1056   }
1057   \let
1058   \markdownIfOption
1059   \@@_if_option:nTF

```

### 2.2.2.1 Finalizing and Freezing the Cache

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

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

```

1060 \@@_add_plain_tex_option:nnn
1061 { frozenCache }
1062 { boolean }
1063 { 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  $\text{\TeX}$  source. It defaults to `\jobname.markdown.in`.

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

```

1064 \tl_set:Nn
1065   \l_tmpa_tl
1066   {
1067     \str_use:N
1068       \g_@@_unquoted_jobname_str
1069       .markdown.in
1070   }
1071 \bool_if:NT
1072   \g_@@_jobname_quoted_bool
1073   {
1074     \tl_put_left:Nn
1075       \l_tmpa_tl
1076       { " }
1077     \tl_put_right:Nn
1078       \l_tmpa_tl
1079       { " }
1080   }
1081 \cs_generate_variant:Nn
1082   \@@_add_plain_tex_option:nnn
1083   { nnV }
1084 \@@_add_plain_tex_option:nnV
1085   { inputTempFileName }
1086   { path }
1087   \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  $\text{\TeX}$  implementation. The option defaults to `.` or, since  $\text{\TeX}$  Live 2024, to the value of the `-output-directory` option of your  $\text{\TeX}$  engine.

In  $\text{\MikTeX}$ , this automatic detection is currently only supported with  $\text{\LuaTeX}$ <sup>34</sup>. If you need to use  $\text{\MikTeX}$  and cannot use  $\text{\LuaTeX}$ , you can either a) fix the automatic detection by setting the environmental variable `TEXMF_OUTPUT_DIRECTORY` manually or by setting the `\markdownOptionOutputDir` option manually.

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

---

<sup>34</sup>See <https://github.com/MiKTeX/miktex/issues/1630>.



```

1088 \@@_add_plain_tex_option:nnn
1089   { outputDir }
1090   { path }
1091   { . }

```

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

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

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

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

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

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

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

```

1092 \@@_add_plain_tex_option:nnn
1093   { plain }
1094   { boolean }
1095   { false }

```

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

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

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

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

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

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

```
1096 \@@_add_plain_tex_option:nnn
1097   { noDefaults }
1098   { boolean }
1099   { false }
```

#### 2.2.2.4 Miscellaneous Options

The `\markdownOptionStripPercentSigns` macro controls whether a percent sign (%) at the beginning of a line will be discarded when buffering Markdown input (see sections 3.2.5 and 3.2.6) or not. Notably, this enables the use of markdown when writing T<sub>E</sub>X package documentation using the Doc L<sup>A</sup>T<sub>E</sub>X package [10] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```
1100 \seq_gput_right:Nn
1101   \g_@@_plain_tex_options_seq
1102   { stripPercentSigns }
1103 \prop_gput:Nnn
1104   \g_@@_plain_tex_option_types_prop
1105   { stripPercentSigns }
1106   { boolean }
1107 \prop_gput:Nnx
1108   \g_@@_default_plain_tex_options_prop
1109   { stripPercentSigns }
1110   { 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.

```

1111 \cs_new:Nn
1112   \@@_define_option_commands_and_keyvals:
1113   {
1114     \seq_map_inline:Nn
1115       \g_@@_option_layers_seq
1116       {
1117         \seq_map_inline:cn
1118           { g_@@_ ##1 _options_seq }
1119           {
1120             \@@_define_option_command:n
1121             { #####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.

```

1122           \@@_with_various_cases:nn
1123           { #####1 }
1124           {
1125             \@@_define_option_keyval:nnn
1126             { ##1 }
1127             { #####1 }
1128             { #####1 }
1129           }
1130       }
1131   }
1132 }
1133 \cs_new:Nn
1134   \@@_define_option_command:n
1135   {

```

For experimental options, redirect the option command to the option command `\markdownOptionExperimental`.

```

1136   \bool_set_false:N
1137     \l_tmpa_bool
1138   \seq_map_inline:Nn
1139     \g_@@_option_layers_seq
1140     {
1141       \seq_if_in:cnT
1142       { g_@@_experimental_ ##1 _options_seq }
1143       { #1 }
1144       {
1145         \bool_set_true:N
1146         \l_tmpa_bool

```

```

1147         \seq_map_break:
1148     }
1149 }
1150 \bool_if:NTF
1151   \l_tmpa_bool
1152 {
1153   \_@@_option_tl_to_csname:nN
1154   { #1 }
1155   \l_tmpa_tl
1156   \cs_if_exist:cF
1157   { \l_tmpa_tl }
1158   {
1159     \cs_gset:cpn
1160     { \l_tmpa_tl }
1161     { \markdownOptionExperimental }
1162   }
1163 }
1164 {

```

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

```

1165   \str_if_eq:nnTF
1166   { #1 }
1167   { outputDir }
1168   { \@@_define_option_command_output_dir: }
1169   {

```

Do not override options defined before loading the package.

```

1170   \_@@_option_tl_to_csname:nN
1171   { #1 }
1172   \l_tmpa_tl
1173   \cs_if_exist:cF
1174   { \l_tmpa_tl }
1175   {
1176     \@@_get_default_option_value:nN
1177     { #1 }
1178     \l_tmpa_tl
1179     \@@_set_option_value:nV
1180     { #1 }
1181     \l_tmpa_tl
1182   }
1183 }
1184 }
1185 }
1186 \ExplSyntaxOff
1187 \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.

```

1188 \ExplSyntaxOn
1189 \cs_new:Nn
1190   \@@_define_option_command_output_dir:
1191   {
1192     \cs_if_free:NT
1193       \markdownOptionOutputDir
1194       {
1195         \bool_if:nTF
1196         {
1197           \cs_if_exist_p:N
1198             \luabridge_tl_set:Nn &&
1199             (
1200               \int_compare_p:nNn
1201                 { \g_luabridge_method_int }
1202                 =
1203                 { \c_luabridge_method_directlua_int } ||
1204               \sys_if_shell_unrestricted_p:
1205             )
1206         }
1207         {

```

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.

```

1208       \group_begin:
1209       \cctab_select:N
1210       \c_str_cctab
1211       \luabridge_tl_set:Nn
1212       \l_tmpa_tl
1213       {
1214         print(
1215           (status.output_directory)
1216           or~os.getenv("TEXMF_OUTPUT_DIRECTORY")
1217           or~"."
1218         )
1219       }

```

```

1220         \tl_gset:NV
1221         \markdownOptionOutputDir
1222         \l_tmpa_tl
1223     \group_end:
1224 }
1225 {
1226     \tl_gset:Nn
1227     \markdownOptionOutputDir
1228     { . }
1229 }
1230 }
1231 }
1232 \cs_new:Nn
1233 \@@_set_option_value:nn
1234 {
1235     \@@_define_option:n
1236     { #1 }
1237     \@@_get_option_type:nN
1238     { #1 }
1239     \l_tmpa_tl
1240     \str_if_eq:NNTF
1241     \c_@@_option_type_counter_tl
1242     \l_tmpa_tl
1243     {
1244         \@@_option_tl_to_csname:nN
1245         { #1 }
1246         \l_tmpa_tl
1247         \int_gset:cn
1248         { \l_tmpa_tl }
1249         { #2 }
1250     }
1251     {
1252         \@@_option_tl_to_csname:nN
1253         { #1 }
1254         \l_tmpa_tl
1255         \cs_set:cpn
1256         { \l_tmpa_tl }
1257         { #2 }
1258     }
1259 }
1260 \cs_new:Nn
1261 \@@_prepend_option_value:nn
1262 {
1263     \@@_get_option_value:nN
1264     { #1 }
1265     \l_tmpa_clist
1266     \clist_put_left:Nn

```

```

1267     \l_tmpa_clist
1268     { #2 }
1269     \@@_set_option_value:nV
1270     { #1 }
1271     \l_tmpa_clist
1272 }
1273 \cs_new:Nn
1274 \@@_append_option_value:nn
1275 {
1276     \@@_get_option_value:nN
1277     { #1 }
1278     \l_tmpa_clist
1279     \clist_put_right:Nn
1280     \l_tmpa_clist
1281     { #2 }
1282     \@@_set_option_value:nV
1283     { #1 }
1284     \l_tmpa_clist
1285 }
1286 \cs_generate_variant:Nn
1287 \@@_set_option_value:nn
1288 { nV }
1289 \cs_new:Nn
1290 \@@_define_option:n
1291 {
1292     \@@_option_tl_to_csname:nN
1293     { #1 }
1294     \l_tmpa_tl
1295     \cs_if_free:cT
1296     { \l_tmpa_tl }
1297     {
1298         \@@_get_option_type:nN
1299         { #1 }
1300         \l_tmpb_tl
1301         \str_if_eq:NNT
1302         \c_@@_option_type_counter_tl
1303         \l_tmpb_tl
1304         {
1305             \@@_option_tl_to_csname:nN
1306             { #1 }
1307             \l_tmpa_tl
1308             \int_new:c
1309             { \l_tmpa_tl }
1310         }
1311     }
1312 }
1313 \cs_new:Nn

```

```

1314 \@@_define_option_keyval:nnn
1315 {
1316   \prop_get:cnN
1317   { g_@@_ #1 _option_types_prop }
1318   { #2 }
1319   \l_tmpa_tl
1320   \str_if_eq:VTF
1321   \l_tmpa_tl
1322   \c_@@_option_type_boolean_tl
1323   {
1324     \keys_define:nn
1325     { markdown/options }
1326     {

```

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

```

1327       #3 .code:n = {
1328         \tl_set:Nx
1329         \l_tmpa_tl
1330         {
1331           \str_case:nnF
1332           { ##1 }
1333           {
1334             { yes } { true }
1335             { no } { false }
1336           }
1337           { ##1 }
1338         }
1339         \@@_set_option_value:nV
1340         { #2 }
1341         \l_tmpa_tl
1342       },
1343       #3 .default:n = { true },
1344     }
1345   }
1346   {
1347     \keys_define:nn
1348     { markdown/options }
1349     {
1350       #3 .code:n = {
1351         \@@_set_option_value:nn
1352         { #2 }
1353         { ##1 }
1354       },
1355     }
1356   }

```

For comma-list options, 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.

```

1357     \str_if_eq:VVT
1358     \l_tmpa_tl
1359     \c_@@_option_type_clist_tl
1360     {
1361         \tl_set:Nn
1362             \l_tmpa_tl
1363             { #3 }
1364         \tl_reverse:N
1365             \l_tmpa_tl
1366         \str_if_eq:enF
1367             {
1368                 \tl_head:V
1369                 \l_tmpa_tl
1370             }
1371             { s }
1372             {
1373                 \msg_error:nnn
1374                     { markdown }
1375                     { malformed-name-for-clist-option }
1376                     { #3 }
1377             }
1378         \tl_set:Nx
1379             \l_tmpa_tl
1380             {
1381                 \tl_tail:V
1382                 \l_tmpa_tl
1383             }
1384         \tl_reverse:N
1385             \l_tmpa_tl
1386         \tl_put_right:Nn
1387             \l_tmpa_tl
1388             {
1389                 .code:n = {
1390                     \@@_append_option_value:nn
1391                         { #2 }
1392                         { ##1 }
1393                 }
1394             }
1395         \keys_define:nV
1396             { markdown/options }
1397             \l_tmpa_tl

```

To achieve parity with token renderers and renderer prototypes (see sections [2.2.5.47](#)

and 2.2.6.2, respectively) the syntax  $\langle key \rangle += \langle value \rangle$  and  $\langle key \rangle \$ = \langle value \rangle$  with the same appending semantics as  $\langle singular\ key \rangle = \langle value \rangle$  is also supported.

```

1398     \keys_define:nn
1399     { markdown/options }
1400     {
1401       #3~+ .code:n = {
1402         \@@_append_option_value:nn
1403         { #2 }
1404         { ##1 }
1405       },
1406       #3 + .code:n = {
1407         \@@_append_option_value:nn
1408         { #2 }
1409         { ##1 }
1410       },
1411       #3~$ .code:n = {
1412         \@@_append_option_value:nn
1413         { #2 }
1414         { ##1 }
1415       },
1416       #3 $ .code:n = {
1417         \@@_append_option_value:nn
1418         { #2 }
1419         { ##1 }
1420       },

```

Furthermore, similar to token renderers and renderer prototypes, the syntax  $\langle key \rangle \hat{=} \langle value \rangle$  is also supported. This syntax prepends  $\langle value \rangle$  to the current value as the leftmost item in the list.

```

1421       #3~^ .code:n = {
1422         \@@_prepend_option_value:nn
1423         { #2 }
1424         { ##1 }
1425       },
1426       #3 ^ .code:n = {
1427         \@@_prepend_option_value:nn
1428         { #2 }
1429         { ##1 }
1430       }
1431     }
1432   }
1433 }
1434 \cs_generate_variant:Nn
1435   \clist_set:Nn
1436   { NV }
1437 \cs_generate_variant:Nn
1438   \keys_define:nn

```

```

1439 { nV }
1440 \prg_generate_conditional_variant:Nnn
1441 \str_if_eq:nn
1442 { en }
1443 { p, F }
1444 \msg_new:nnn
1445 { markdown }
1446 { malformed-name-for-clist-option }
1447 {
1448   Clist-option-name~#1~does~not~end~with~-s.
1449 }

```

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

```

1450 \str_if_eq:VVT
1451 \c_@@_top_layer_tl
1452 \c_@@_option_layer_plain_tex_tl
1453 {
1454   \@@_define_option_commands_and_keyvals:
1455 }
1456 \ExplSyntaxOff

```

### 2.2.3 Themes

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

The key-values `theme=<theme name>` and `import=<theme name>`, optionally followed by `@<theme version>`, load a  $\text{\TeX}$  document (further referred to as *a theme*) named `markdowntheme<munged theme name>.tex`, where the *munged theme name* is the *theme name* after the substitution of all forward slashes (/) for an underscore (\_). The theme name must be *qualified* and contain no underscores or at signs (@). Themes are inspired by the Beamer  $\text{\LaTeX}$  package, which provides similar functionality with its `\usetheme` macro [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 `<theme author>/<theme purpose>/<private naming scheme>`, where the *private naming scheme* may contain additional forward slashes. For example, a theme by a user `witiko` for the MU theme of the Beamer document class may have the name `witiko/beamer/MU`.

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

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

```

1457 \ExplSyntaxOn
1458 \keys_define:nn
1459 { markdown/options }
1460 {
1461   theme .code:n = {
1462     \@@_set_theme:n
1463     { #1 }
1464   },
1465   import .code:n = {
1466     \tl_set:Nn
1467     \l_tmpa_tl
1468     { #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.

```

1469     \tl_replace_all:NnV
1470     \l_tmpa_tl
1471     { / }
1472     \c_backslash_str
1473     \keys_set:nV
1474     { markdown/options/import }
1475     \l_tmpa_tl
1476   },
1477 }

```

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.

```

1478 \seq_new:N
1479 \g_@@_theme_names_seq
1480 \seq_new:N
1481 \g_@@_theme_versions_seq
1482 \tl_new:N
1483 \g_@@_current_theme_tl
1484 \tl_gset:Nn
1485 \g_@@_current_theme_tl
1486 { }

```

```

1487 \seq_gput_right:NV
1488   \g_@@_theme_names_seq
1489   \g_@@_current_theme_tl
1490 \cs_new:Npn
1491   \markdownThemeVersion
1492   { }
1493 \seq_gput_right:NV
1494   \g_@@_theme_versions_seq
1495   \g_@@_current_theme_tl
1496 \cs_new:Nn
1497   \@@_set_theme:n
1498   {

```

First, we validate the theme name.

```

1499   \str_if_in:nnF
1500     { #1 }
1501     { / }
1502     {
1503       \msg_error:nnn
1504         { markdown }
1505         { unqualified-theme-name }
1506         { #1 }
1507     }
1508   \str_if_in:nnT
1509     { #1 }
1510     { _ }
1511     {
1512       \msg_error:nnn
1513         { markdown }
1514         { underscores-in-theme-name }
1515         { #1 }
1516     }

```

Next, we extract the theme version.

```

1517   \str_if_in:nnTF
1518     { #1 }
1519     { @ }
1520     {
1521       \regex_extract_once:nnN
1522         { (.*?) @ (.*?) }
1523         { #1 }
1524         \l_tmpa_seq
1525       \seq_gpop_left:NN
1526         \l_tmpa_seq
1527         \l_tmpa_tl
1528       \seq_gpop_left:NN
1529         \l_tmpa_seq
1530         \l_tmpa_tl

```

```

1531     \tl_gset:NV
1532     \g_@@_current_theme_tl
1533     \l_tmpa_tl
1534     \seq_gpop_left:NN
1535     \l_tmpa_seq
1536     \l_tmpa_tl
1537     \cs_gset:Npe
1538     \markdownThemeVersion
1539     {
1540         \tl_use:N
1541         \l_tmpa_tl
1542     }
1543 }
1544 {
1545     \tl_gset:Nn
1546     \g_@@_current_theme_tl
1547     { #1 }
1548     \cs_gset:Npn
1549     \markdownThemeVersion
1550     { latest }
1551 }

```

Next, we strip a leading slash, if present, for consistency with snippets, where a leading slash distinguishes absolute snippet names from relative ones. Theme names, however, are currently always absolute, so stripping the slash is only a syntactic normalization and has no semantic effect.

```

1552     \str_if_eq:enT
1553     {
1554         \tl_head:V
1555         \g_@@_current_theme_tl
1556     }
1557     { / }
1558     {
1559         \tl_gset:Ne
1560         \g_@@_current_theme_tl
1561         {
1562             \tl_tail:V
1563             \g_@@_current_theme_tl
1564         }
1565     }

```

Next, we munge the theme name.

```

1566     \str_set:NV
1567     \l_tmpa_str
1568     \g_@@_current_theme_tl
1569     \str_replace_all:Nnn
1570     \l_tmpa_str
1571     { / }

```

```
1572      { _ }
```

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

```
1573      \tl_set:NV
1574        \l_tmpa_tl
1575        \g_@@_current_theme_tl
1576      \tl_put_right:Nn
1577        \g_@@_current_theme_tl
1578        { / }
1579      \seq_gput_right:NV
1580        \g_@@_theme_names_seq
1581        \g_@@_current_theme_tl
1582      \seq_gput_right:NV
1583        \g_@@_theme_versions_seq
1584        \markdownThemeVersion
1585      \@@_load_theme:VeV
1586        \l_tmpa_tl
1587        { \markdownThemeVersion }
1588        \l_tmpa_str
```

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

```
1589      \seq_gpop_right:NN
1590        \g_@@_theme_names_seq
1591        \l_tmpa_tl
1592      \seq_get_right:NN
1593        \g_@@_theme_names_seq
1594        \l_tmpa_tl
1595      \tl_gset:NV
1596        \g_@@_current_theme_tl
1597        \l_tmpa_tl
1598      \seq_gpop_right:NN
1599        \g_@@_theme_versions_seq
1600        \l_tmpa_tl
1601      \seq_get_right:NN
1602        \g_@@_theme_versions_seq
1603        \l_tmpa_tl
1604      \cs_gset:Npe
1605        \markdownThemeVersion
1606        {
1607          \tl_use:N
1608            \l_tmpa_tl
1609        }
1610    }
1611  \msg_new:nnnn
1612    { markdown }
1613    { unqualified-theme-name }
```

```

1614 { Won't~load~theme~with~unqualified~name~#1 }
1615 { Theme~names~must~contain~at~least~one~forward~slash }
1616 \msg_new:nnnn
1617 { markdown }
1618 { underscores-in-theme-name }
1619 { Won't~load~theme~with~an~underscore~in~its~name~#1 }
1620 { Theme~names~must~not~contain~underscores~in~their~names }
1621 \cs_generate_variant:Nn
1622 \tl_replace_all:Nnn
1623 { NnV }
1624 \cs_generate_variant:Nn
1625 \cs_gset:Npn
1626 { Npe }
1627 \prg_generate_conditional_variant:Nnn
1628 \str_if_eq:nn
1629 { en }
1630 { T }

```

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.

```

1631 \prop_new:N
1632 \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 and for the infostrings `dot` and `plantuml`, the key-value attribute `format` can be used to specify the output image format. The remaining attributes are treated as image attributes.

```

\documentclass{article}
\usepackage[relativeReferences, texComments]{markdown}
\markdownSetup{import=witiko/diagrams@v2}
\begin{document}
\begin{markdown}
``` dot {caption="An example directed graph" %
        format=svg width=12cm #dot}
digraph tree {
  margin = 0;
  rankdir = "LR";

```



```

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

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

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

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

```

```

' Diagram title
title Simple Request-Response Flow

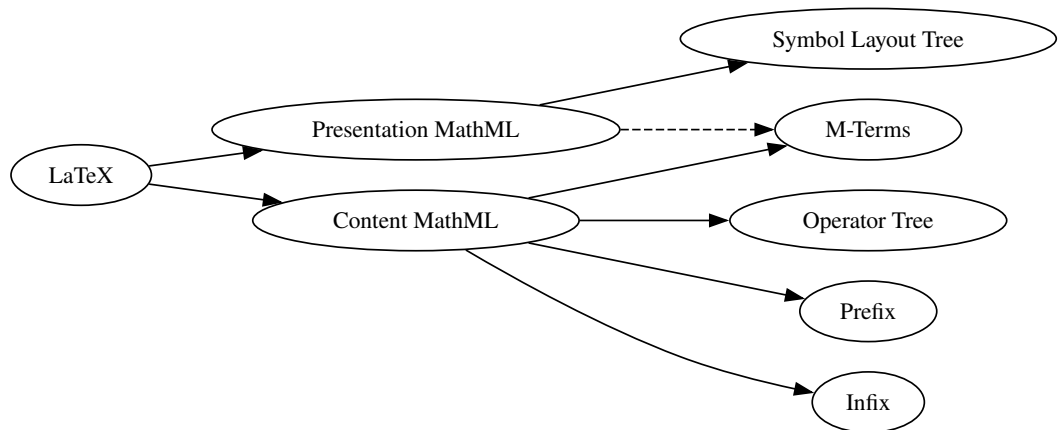
' Messages
C -> S: Send Request
note over S: Process request

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

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

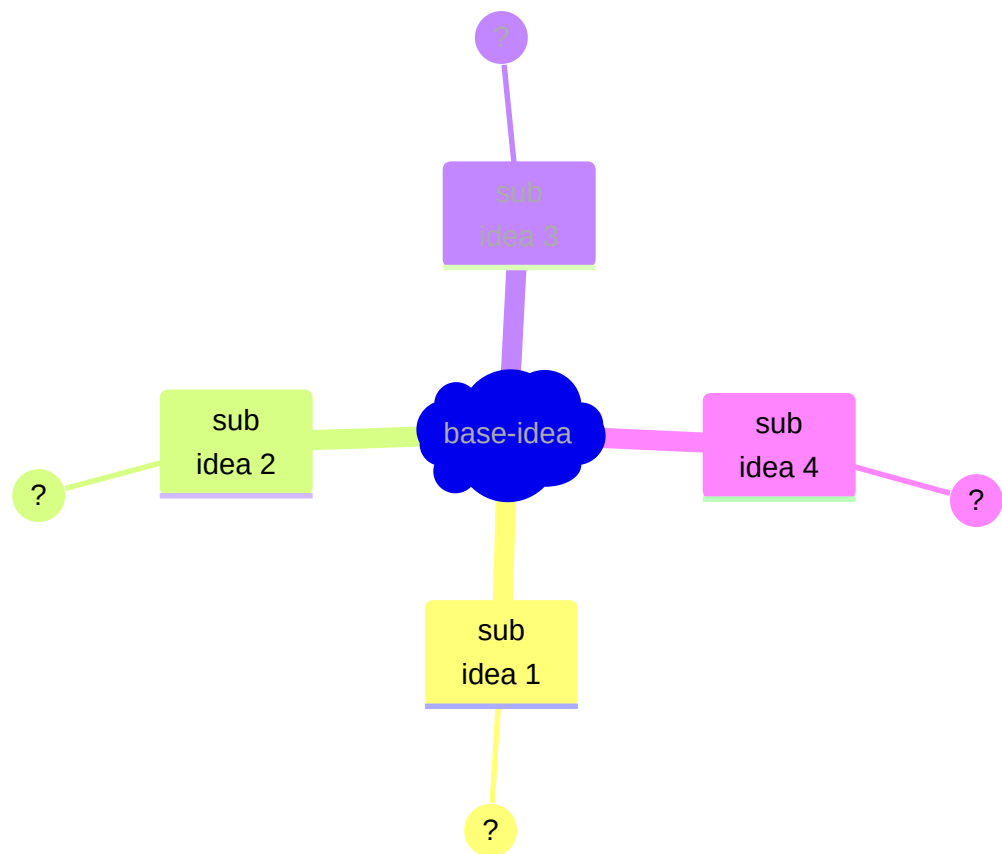
```

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



**Figure 4: An example directed graph**

The theme requires a Unix-like operating system with GNU Diffutils, Graphviz,



**Figure 5: An example mindmap**

## Simple Request-Response Flow

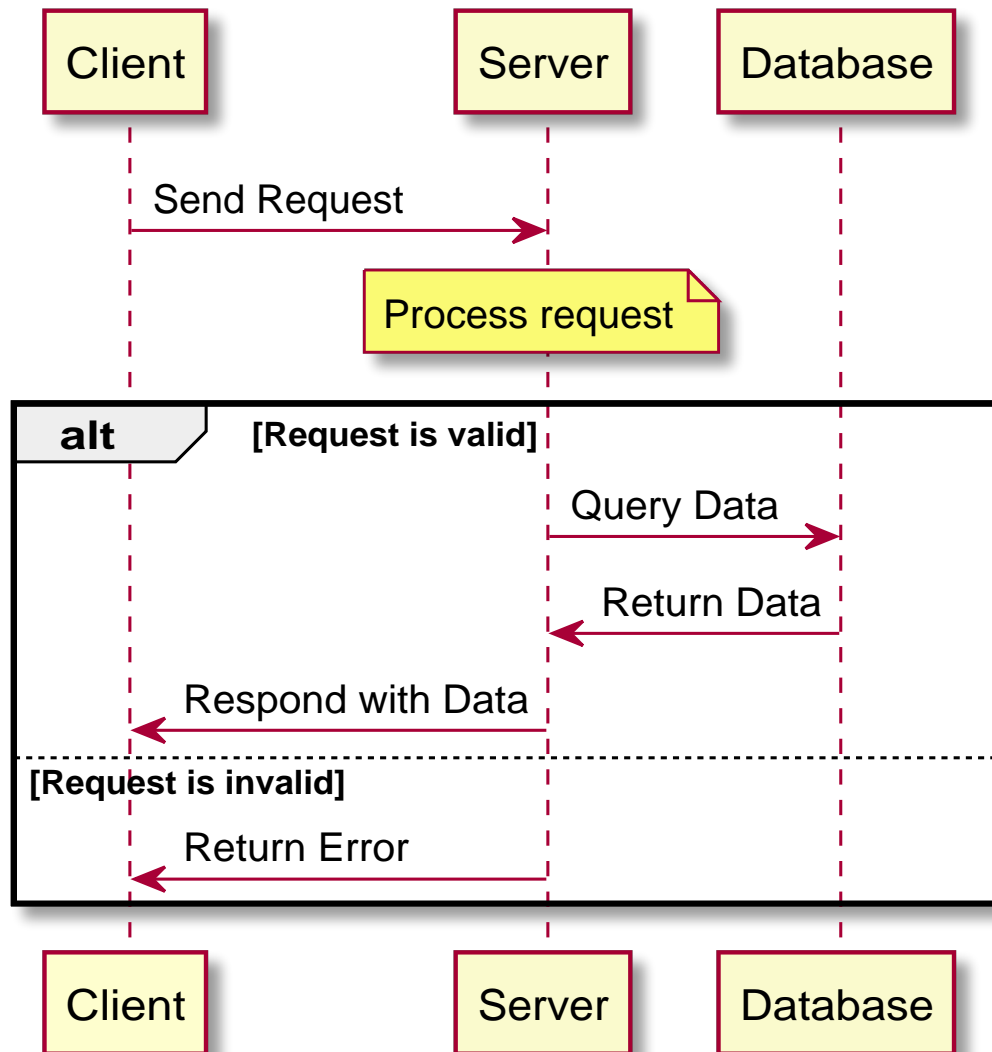


Figure 6: An example UML sequence diagram

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 TeX option is enabled.

**witiko/glossaries** A theme that defines snippets with integrations for the L<sup>A</sup>T<sub>E</sub>X package glossaries. See Section 2.3.5 for more details.

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


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

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

```
\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` L<sup>A</sup>T<sub>E</sub>X package and a Unix-like operating system with GNU Coreutils `md5sum` and either GNU `Wget` or `cURL` installed. The theme also requires shell access unless the `frozenCache` plain T<sub>E</sub>X 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 T<sub>E</sub>X theme with the default definitions of token renderer prototypes for plain T<sub>E</sub>X. This theme is loaded automatically together with the package and explicitly loading it has no effect.

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.

```
1633 \prop_new:N
1634   \g_@@_snippets_prop
1635 \cs_new:Nn
1636   \@@_setup_snippet:nn
1637   {
1638     \tl_if_empty:nT
1639       { #1 }
1640     {
1641       \msg_error:nnn
1642         { markdown }
1643         { empty-snippet-name }
1644         { #1 }
1645     }
1646   \@@_resolve_snippet_name:nN
1647     { #1 }
1648   \l_tmpa_tl
```

```

1649 \@@_if_snippet_exists:nT
1650 { #1 }
1651 {
1652   \msg_warning:nnV
1653   { markdown }
1654   { redefined-snippet }
1655   \l_tmpa_tl
1656 }
1657 \keys_precompile:nnN
1658 { markdown/options }
1659 { #2 }
1660 \l_tmpb_tl
1661 \prop_gput:NVV
1662 \g_@@_snippets_prop
1663 \l_tmpa_tl
1664 \l_tmpb_tl
1665 }
1666 \cs_gset_eq:NN
1667 \markdownSetupSnippet
1668 \@@_setup_snippet:nn
1669 \msg_new:nnnn
1670 { markdown }
1671 { empty-snippet-name }
1672 { Empty-snippet-name~#1 }
1673 { Pick-a-non-empty-name-for-your-snippet }
1674 \msg_new:nnn
1675 { markdown }
1676 { redefined-snippet }
1677 { Redefined~snippet~#1 }

```

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

```

1678 \tl_new:N
1679 \l_@@_current_snippet_tl
1680 \prg_new_conditional:Nnn
1681 \@@_if_snippet_exists:n
1682 { TF, T }
1683 {
1684   \@@_resolve_snippet_name:nN
1685   { #1 }
1686   \l_@@_current_snippet_tl
1687   \prop_if_in:NVTF
1688   \g_@@_snippets_prop
1689   \l_@@_current_snippet_tl
1690   { \prg_return_true: }
1691   { \prg_return_false: }
1692 }
1693 \cs_gset_eq:NN

```

```

1694 \markdownIfSnippetExists
1695 \@@_if_snippet_exists:nTF
1696 \cs_new:Nn
1697 \@@_resolve_snippet_name:nN
1698 {

```

If the provided snippet name starts with a slash (/), consider it as *absolute* and take the rest of the name as the full name of the defined snippet, regardless of the currently processed theme.

```

1699 \str_if_eq:enTF
1700 {
1701     \tl_head:n
1702     { #1 }
1703 }
1704 { / }
1705 {
1706     \tl_set:Ne
1707     #2
1708     {
1709         \tl_tail:n
1710         { #1 }
1711     }
1712 }
1713 {

```

Otherwise, consider the provided snippet name as *relative* and prepend the name of the currently processed theme to it, if any. The result is then the full name of the defined snippet.

```

1714 \tl_set:NV
1715 #2
1716 \g_@@_current_theme_tl
1717 \tl_put_right:Nn
1718 #2
1719 { #1 }
1720 }
1721 }
1722 \prg_generate_conditional_variant:Nnn
1723 \str_if_eq:nn
1724 { en }
1725 { TF }

```

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

```

1726 \keys_define:nn
1727 { markdown/options }
1728 {
1729     snippet .code:n = {
1730         \@@_resolve_snippet_name:nN
1731         { #1 }
1732         \l_tmpa_tl

```



```

1733     \@@_if_snippet_exists:nTF
1734     { #1 }
1735     {
1736         \prop_get:NVN
1737         \g_@@_snippets_prop
1738         \l_tmpa_tl
1739         \l_tmpb_tl
1740         \tl_use:N
1741         \l_tmpb_tl
1742     }
1743     {
1744         \msg_error:nnV
1745         { markdown }
1746         { undefined-snippet }
1747         \l_tmpa_tl
1748     }
1749 }
1750 }
1751 \msg_new:nnn
1752 { markdown }
1753 { undefined-snippet }
1754 { Can't~invoke~undefined~snippet~#1 }
1755 \ExplSyntaxOff

```

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

```

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

```

The following ordered list will be preceded by arabic numerals:

1. wahid
2. aithnayn

```

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

```

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```
\end{markdown}
```

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

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

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```
\end{markdown}
```

Alternatively, we can use the extended variant of the `import` L<sup>A</sup>T<sub>E</sub>X option that allows us to import the `romanNumerals` snippet to the current namespace for easier access:

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

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```
\end{markdown}
```

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

```
\markdownSetup{
  import = {
```

```

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

```

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```

\end{markdown}

```

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

```

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

```

```

1756 \ExplSyntaxOn
1757 \tl_new:N
1758   \l_@@_import_current_theme_tl
1759 \keys_define:nn
1760   { markdown/options/import }
1761   {

```

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

```

1762     unknown .default:n = {},
1763     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.

```

1764     \tl_set_eq:NN
1765     \l_@@_import_current_theme_tl
1766     \l_keys_key_str
1767     \tl_replace_all:NVN
1768     \l_@@_import_current_theme_tl
1769     \c_backslash_str
1770     { / }

```

Here, we import the snippets.

```

1771     \clist_map_inline:nn
1772     { #1 }
1773     {
1774         \regex_extract_once:nnNTF
1775         { ^(.*)\s+as\s+(.*)$ }
1776         { ##1 }
1777         \l_tmpa_seq
1778         {
1779             \seq_pop:NN
1780             \l_tmpa_seq
1781             \l_tmpa_tl
1782             \seq_pop:NN
1783             \l_tmpa_seq
1784             \l_tmpa_tl
1785             \seq_pop:NN
1786             \l_tmpa_seq
1787             \l_tmpb_tl
1788         }
1789         {
1790             \tl_set:Nn
1791             \l_tmpa_tl
1792             { ##1 }
1793             \tl_set:Nn
1794             \l_tmpb_tl
1795             { ##1 }
1796         }
1797         \tl_put_left:Nn
1798         \l_tmpa_tl
1799         { / }
1800         \tl_put_left:NV
1801         \l_tmpa_tl
1802         \l_@@_import_current_theme_tl
1803         \@@_setup_snippet:Vx
1804         \l_tmpb_tl
1805         { snippet = { \l_tmpa_tl } }
1806     }

```

Here, we load the theme.

```

1807     \@@_set_theme:V
1808     \l_@@_import_current_theme_tl
1809   },
1810 }
1811 \cs_generate_variant:Nn
1812   \tl_replace_all:Nnn
1813   { NVn }
1814 \cs_generate_variant:Nn
1815   \@@_set_theme:n
1816   { V }
1817 \cs_generate_variant:Nn
1818   \@@_setup_snippet:nn
1819   { Vx }

```

## 2.2.5 Token Renderers

The following  $\text{\TeX}$  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.

```

1820 \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.

```

1821 \prop_new:N \g_@@_renderer_arities_prop
1822 \ExplSyntaxOff

```

### 2.2.5.1 Acronym Renderers

The following macro is only produced when the option `acronyms` is non-empty.

`\markdownRendererAcronym` represents an acronym, initialism, or another all-caps sequence. The macro receives a single attribute that corresponds to the sequence.

```

1823 \ExplSyntaxOn
1824 \cs_gset_protected:Npn
1825   \markdownRendererAcronym
1826   {
1827     \markdownRendererAcronymPrototype
1828   }
1829 \seq_gput_right:Nn
1830   \g_@@_renderers_seq
1831   { acronym }
1832 \prop_gput:Nnn
1833   \g_@@_renderer_arities_prop

```

```

1834 { acronym }
1835 { 1 }
1836 \ExplSyntaxOff

```

### 2.2.5.2 Attribute Renderers

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

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

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

`\markdownRendererAttributeClassName` represents the  $\langle class name \rangle$  of a markdown element (`class="⟨class name⟩ ..."` in HTML and `.⟨class name⟩` in markdown attributes). The macro receives a single attribute that corresponds to the  $\langle class name \rangle$ .

`\markdownRendererAttributeKeyValue` represents an 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.

```

1837 \ExplSyntaxOn
1838 \cs_gset_protected:Npn
1839   \markdownRendererAttributeIdentifier
1840   {
1841     \markdownRendererAttributeIdentifierPrototype
1842   }
1843 \seq_gput_right:Nn
1844   \g_@@_renderers_seq
1845   { attributeIdentifier }
1846 \prop_gput:Nnn
1847   \g_@@_renderer_arities_prop
1848   { attributeIdentifier }
1849   { 1 }
1850 \cs_gset_protected:Npn
1851   \markdownRendererAttributeClassName
1852   {
1853     \markdownRendererAttributeClassNamePrototype
1854   }
1855 \seq_gput_right:Nn

```

```

1856 \g_@@_renderers_seq
1857 { attributeClassName }
1858 \prop_gput:Nnn
1859 \g_@@_renderer_arities_prop
1860 { attributeClassName }
1861 { 1 }
1862 \cs_gset_protected:Npn
1863 \markdownRendererAttributeKeyValue
1864 {
1865   \markdownRendererAttributeKeyValuePrototype
1866 }
1867 \seq_gput_right:Nn
1868 \g_@@_renderers_seq
1869 { attributeKeyValue }
1870 \prop_gput:Nnn
1871 \g_@@_renderer_arities_prop
1872 { attributeKeyValue }
1873 { 2 }
1874 \ExplSyntaxOff

```

### 2.2.5.3 Block Quote Renderers

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

```

1875 \ExplSyntaxOn
1876 \cs_gset_protected:Npn
1877 \markdownRendererBlockQuoteBegin
1878 {
1879   \markdownRendererBlockQuoteBeginPrototype
1880 }
1881 \seq_gput_right:Nn
1882 \g_@@_renderers_seq
1883 { blockQuoteBegin }
1884 \prop_gput:Nnn
1885 \g_@@_renderer_arities_prop
1886 { blockQuoteBegin }
1887 { 0 }
1888 \ExplSyntaxOff

```

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

```

1889 \ExplSyntaxOn
1890 \cs_gset_protected:Npn
1891 \markdownRendererBlockQuoteEnd
1892 {
1893   \markdownRendererBlockQuoteEndPrototype
1894 }

```

```

1895 \seq_gput_right:Nn
1896   \g_@@_renderers_seq
1897   { blockQuoteEnd }
1898 \prop_gput:Nnn
1899   \g_@@_renderer_arities_prop
1900   { blockQuoteEnd }
1901   { 0 }
1902 \ExplSyntaxOff

```

#### 2.2.5.4 Bracketed Spans

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

The `\markdownRendererBracketedSpan` macro represents an inline bracketed span. It receives a single argument that corresponds to the inline bracketed span.

```

1903 \ExplSyntaxOn
1904 \cs_gset_protected:Npn
1905   \markdownRendererBracketedSpan
1906   {
1907     \markdownRendererBracketedSpanPrototype
1908   }
1909 \seq_gput_right:Nn
1910   \g_@@_renderers_seq
1911   { bracketedSpan }
1912 \prop_gput:Nnn
1913   \g_@@_renderer_arities_prop
1914   { bracketedSpan }
1915   { 1 }
1916 \ExplSyntaxOff

```

#### 2.2.5.5 Bracketed Span 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.

```

1917 \ExplSyntaxOn
1918 \cs_gset_protected:Npn
1919   \markdownRendererBracketedSpanAttributeContextBegin
1920   {
1921     \markdownRendererBracketedSpanAttributeContextBeginPrototype
1922   }
1923 \seq_gput_right:Nn
1924   \g_@@_renderers_seq
1925   { bracketedSpanAttributeContextBegin }

```



```

1926 \prop_gput:Nnn
1927   \g_@@_renderer_arities_prop
1928   { bracketedSpanAttributeContextBegin }
1929   { 0 }
1930 \cs_gset_protected:Npn
1931   \markdownRendererBracketedSpanAttributeContextEnd
1932   {
1933     \markdownRendererBracketedSpanAttributeContextEndPrototype
1934   }
1935 \seq_gput_right:Nn
1936   \g_@@_renderers_seq
1937   { bracketedSpanAttributeContextEnd }
1938 \prop_gput:Nnn
1939   \g_@@_renderer_arities_prop
1940   { bracketedSpanAttributeContextEnd }
1941   { 0 }
1942 \ExplSyntaxOff

```

### 2.2.5.6 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.

```

1943 \ExplSyntaxOn
1944 \cs_gset_protected:Npn
1945   \markdownRendererUlBegin
1946   {
1947     \markdownRendererUlBeginPrototype
1948   }
1949 \seq_gput_right:Nn
1950   \g_@@_renderers_seq
1951   { ulBegin }
1952 \prop_gput:Nnn
1953   \g_@@_renderer_arities_prop
1954   { ulBegin }
1955   { 0 }
1956 \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.

```

1957 \ExplSyntaxOn
1958 \cs_gset_protected:Npn
1959   \markdownRendererUlBeginTight
1960   {

```

```

1961     \markdownRendererUlBeginTightPrototype
1962   }
1963 \seq_gput_right:Nn
1964   \g_@@_renderers_seq
1965   { ulBeginTight }
1966 \prop_gput:Nnn
1967   \g_@@_renderer_arities_prop
1968   { ulBeginTight }
1969   { 0 }
1970 \ExplSyntaxOff

```

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

```

1971 \ExplSyntaxOn
1972 \cs_gset_protected:Npn
1973   \markdownRendererUlItem
1974   {
1975     \markdownRendererUlItemPrototype
1976   }
1977 \seq_gput_right:Nn
1978   \g_@@_renderers_seq
1979   { ulItem }
1980 \prop_gput:Nnn
1981   \g_@@_renderer_arities_prop
1982   { ulItem }
1983   { 0 }
1984 \ExplSyntaxOff

```

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

```

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

```

1999 \ExplSyntaxOn
2000 \cs_gset_protected:Npn
2001   \markdownRendererUEnd
2002   {
2003     \markdownRendererUEndPrototype
2004   }
2005 \seq_gput_right:Nn
2006   \g_@@_renderers_seq
2007   { ulEnd }
2008 \prop_gput:Nnn
2009   \g_@@_renderer_arities_prop
2010   { ulEnd }
2011   { 0 }
2012 \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.

```

2013 \ExplSyntaxOn
2014 \cs_gset_protected:Npn
2015   \markdownRendererUEndTight
2016   {
2017     \markdownRendererUEndTightPrototype
2018   }
2019 \seq_gput_right:Nn
2020   \g_@@_renderers_seq
2021   { ulEndTight }
2022 \prop_gput:Nnn
2023   \g_@@_renderer_arities_prop
2024   { ulEndTight }
2025   { 0 }
2026 \ExplSyntaxOff

```

### 2.2.5.7 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.

```

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

```

2041 \ExplSyntaxOn
2042 \cs_gset_protected:Npn
2043   \markdownRendererTextCite
2044   {
2045     \markdownRendererTextCitePrototype
2046   }
2047 \seq_gput_right:Nn
2048   \g_@@_renderers_seq
2049   { textCite }
2050 \prop_gput:Nnn
2051   \g_@@_renderer_arities_prop
2052   { textCite }
2053   { 1 }
2054 \ExplSyntaxOff

```

#### 2.2.5.8 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.

```

2055 \ExplSyntaxOn
2056 \cs_gset_protected:Npn
2057   \markdownRendererInputVerbatim
2058   {
2059     \markdownRendererInputVerbatimPrototype
2060   }
2061 \seq_gput_right:Nn

```

```

2062 \g_@@_renderers_seq
2063 { inputVerbatim }
2064 \prop_gput:Nnn
2065 \g_@@_renderer_arities_prop
2066 { inputVerbatim }
2067 { 1 }
2068 \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.

```

2069 \ExplSyntaxOn
2070 \cs_gset_protected:Npn
2071 \markdownRendererInputFencedCode
2072 {
2073   \markdownRendererInputFencedCodePrototype
2074 }
2075 \seq_gput_right:Nn
2076 \g_@@_renderers_seq
2077 { inputFencedCode }
2078 \prop_gput:Nnn
2079 \g_@@_renderer_arities_prop
2080 { inputFencedCode }
2081 { 3 }
2082 \ExplSyntaxOff

```

#### 2.2.5.9 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.

```

2083 \ExplSyntaxOn
2084 \cs_gset_protected:Npn
2085 \markdownRendererCodeSpan
2086 {
2087   \markdownRendererCodeSpanPrototype
2088 }
2089 \seq_gput_right:Nn
2090 \g_@@_renderers_seq
2091 { codeSpan }
2092 \prop_gput:Nnn
2093 \g_@@_renderer_arities_prop
2094 { codeSpan }
2095 { 1 }
2096 \ExplSyntaxOff

```

### 2.2.5.10 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.

```
2097 \ExplSyntaxOn
2098 \cs_gset_protected:Npn
2099   \markdownRendererCodeSpanAttributeContextBegin
2100   {
2101     \markdownRendererCodeSpanAttributeContextBeginPrototype
2102   }
2103 \seq_gput_right:Nn
2104   \g_@@_renderers_seq
2105   { codeSpanAttributeContextBegin }
2106 \prop_gput:Nnn
2107   \g_@@_renderer_arities_prop
2108   { codeSpanAttributeContextBegin }
2109   { 0 }
2110 \cs_gset_protected:Npn
2111   \markdownRendererCodeSpanAttributeContextEnd
2112   {
2113     \markdownRendererCodeSpanAttributeContextEndPrototype
2114   }
2115 \seq_gput_right:Nn
2116   \g_@@_renderers_seq
2117   { codeSpanAttributeContextEnd }
2118 \prop_gput:Nnn
2119   \g_@@_renderer_arities_prop
2120   { codeSpanAttributeContextEnd }
2121   { 0 }
2122 \ExplSyntaxOff
```

### 2.2.5.11 Content Block Renderers

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

```
2123 \ExplSyntaxOn
2124 \cs_gset_protected:Npn
2125   \markdownRendererContentBlock
2126   {
2127     \markdownRendererContentBlockPrototype
2128   }
2129 \seq_gput_right:Nn
```

```

2130 \g_@@_renderers_seq
2131 { contentBlock }
2132 \prop_gput:Nnn
2133 \g_@@_renderer_arities_prop
2134 { contentBlock }
2135 { 4 }
2136 \ExplSyntaxOff

```

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

```

2137 \ExplSyntaxOn
2138 \cs_gset_protected:Npn
2139 \markdownRendererContentBlockOnlineImage
2140 {
2141   \markdownRendererContentBlockOnlineImagePrototype
2142 }
2143 \seq_gput_right:Nn
2144 \g_@@_renderers_seq
2145 { contentBlockOnlineImage }
2146 \prop_gput:Nnn
2147 \g_@@_renderer_arities_prop
2148 { contentBlockOnlineImage }
2149 { 4 }
2150 \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>35</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.

```

2151 \ExplSyntaxOn
2152 \cs_gset_protected:Npn

```

---

<sup>35</sup> Filenames other than `markdown-languages.json` may be specified using the `contentBlocksLanguageMap` Lua option.

```

2153 \markdownRendererContentBlockCode
2154 {
2155   \markdownRendererContentBlockCodePrototype
2156 }
2157 \seq_gput_right:Nn
2158 \g_@@_renderers_seq
2159 { contentBlockCode }
2160 \prop_gput:Nnn
2161 \g_@@_renderer_arities_prop
2162 { contentBlockCode }
2163 { 5 }
2164 \ExplSyntaxOff

```

### 2.2.5.12 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.

```

2165 \ExplSyntaxOn
2166 \cs_gset_protected:Npn
2167 \markdownRendererDlBegin
2168 {
2169   \markdownRendererDlBeginPrototype
2170 }
2171 \seq_gput_right:Nn
2172 \g_@@_renderers_seq
2173 { dlBegin }
2174 \prop_gput:Nnn
2175 \g_@@_renderer_arities_prop
2176 { dlBegin }
2177 { 0 }
2178 \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.

```

2179 \ExplSyntaxOn
2180 \cs_gset_protected:Npn
2181 \markdownRendererDlBeginTight
2182 {
2183   \markdownRendererDlBeginTightPrototype
2184 }
2185 \seq_gput_right:Nn

```



```

2186 \g_@@_renderers_seq
2187 { dlBeginTight }
2188 \prop_gput:Nnn
2189 \g_@@_renderer_arities_prop
2190 { dlBeginTight }
2191 { 0 }
2192 \ExplSyntaxOff

```

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

```

2193 \ExplSyntaxOn
2194 \cs_gset_protected:Npn
2195 \markdownRendererDlItem
2196 {
2197   \markdownRendererDlItemPrototype
2198 }
2199 \seq_gput_right:Nn
2200 \g_@@_renderers_seq
2201 { dlItem }
2202 \prop_gput:Nnn
2203 \g_@@_renderer_arities_prop
2204 { dlItem }
2205 { 1 }
2206 \ExplSyntaxOff

```

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

```

2207 \ExplSyntaxOn
2208 \cs_gset_protected:Npn
2209 \markdownRendererDlItemEnd
2210 {
2211   \markdownRendererDlItemEndPrototype
2212 }
2213 \seq_gput_right:Nn
2214 \g_@@_renderers_seq
2215 { dlItemEnd }
2216 \prop_gput:Nnn
2217 \g_@@_renderer_arities_prop
2218 { dlItemEnd }
2219 { 0 }
2220 \ExplSyntaxOff

```

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

```

2221 \ExplSyntaxOn
2222 \cs_gset_protected:Npn
2223 \markdownRendererDlDefinitionBegin

```

```

2224 {
2225     \markdownRendererDlDefinitionBeginPrototype
2226 }
2227 \seq_gput_right:Nn
2228 \g_@@_renderers_seq
2229 { dlDefinitionBegin }
2230 \prop_gput:Nnn
2231 \g_@@_renderer_arities_prop
2232 { dlDefinitionBegin }
2233 { 0 }
2234 \ExplSyntaxOff

```

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

```

2235 \ExplSyntaxOn
2236 \cs_gset_protected:Npn
2237 \markdownRendererDlDefinitionEnd
2238 {
2239     \markdownRendererDlDefinitionEndPrototype
2240 }
2241 \seq_gput_right:Nn
2242 \g_@@_renderers_seq
2243 { dlDefinitionEnd }
2244 \prop_gput:Nnn
2245 \g_@@_renderer_arities_prop
2246 { dlDefinitionEnd }
2247 { 0 }
2248 \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.

```

2249 \ExplSyntaxOn
2250 \cs_gset_protected:Npn
2251 \markdownRendererDlEnd
2252 {
2253     \markdownRendererDlEndPrototype
2254 }
2255 \seq_gput_right:Nn
2256 \g_@@_renderers_seq
2257 { dlEnd }
2258 \prop_gput:Nnn
2259 \g_@@_renderer_arities_prop
2260 { dlEnd }
2261 { 0 }
2262 \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.

```

2263 \ExplSyntaxOn
2264 \cs_gset_protected:Npn
2265   \markdownRendererDlEndTight
2266   {
2267     \markdownRendererDlEndTightPrototype
2268   }
2269 \seq_gput_right:Nn
2270   \g_@@_renderers_seq
2271   { dlEndTight }
2272 \prop_gput:Nnn
2273   \g_@@_renderer_arities_prop
2274   { dlEndTight }
2275   { 0 }
2276 \ExplSyntaxOff

```

#### 2.2.5.13 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.

```

2277 \ExplSyntaxOn
2278 \cs_gset_protected:Npn
2279   \markdownRendererEllipsis
2280   {
2281     \markdownRendererEllipsisPrototype
2282   }
2283 \seq_gput_right:Nn
2284   \g_@@_renderers_seq
2285   { ellipsis }
2286 \prop_gput:Nnn
2287   \g_@@_renderer_arities_prop
2288   { ellipsis }
2289   { 0 }
2290 \ExplSyntaxOff

```

#### 2.2.5.14 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.

```

2291 \ExplSyntaxOn
2292 \cs_gset_protected:Npn

```

```

2293 \markdownRendererEmphasis
2294 {
2295     \markdownRendererEmphasisPrototype
2296 }
2297 \seq_gput_right:Nn
2298 \g_@@_renderers_seq
2299 { emphasis }
2300 \prop_gput:Nnn
2301 \g_@@_renderer_arities_prop
2302 { emphasis }
2303 { 1 }
2304 \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.

```

2305 \ExplSyntaxOn
2306 \cs_gset_protected:Npn
2307 \markdownRendererStrongEmphasis
2308 {
2309     \markdownRendererStrongEmphasisPrototype
2310 }
2311 \seq_gput_right:Nn
2312 \g_@@_renderers_seq
2313 { strongEmphasis }
2314 \prop_gput:Nnn
2315 \g_@@_renderer_arities_prop
2316 { strongEmphasis }
2317 { 1 }
2318 \ExplSyntaxOff

```

#### 2.2.5.15 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.

```

2319 \ExplSyntaxOn
2320 \cs_gset_protected:Npn
2321 \markdownRendererFencedCodeAttributeContextBegin
2322 {
2323     \markdownRendererFencedCodeAttributeContextBeginPrototype
2324 }
2325 \seq_gput_right:Nn
2326 \g_@@_renderers_seq

```

```

2327 { fencedCodeAttributeContextBegin }
2328 \prop_gput:Nnn
2329 \g_@@_renderer_arities_prop
2330 { fencedCodeAttributeContextBegin }
2331 { 0 }
2332 \cs_gset_protected:Npn
2333 \markdownRendererFencedCodeAttributeContextEnd
2334 {
2335   \markdownRendererFencedCodeAttributeContextEndPrototype
2336 }
2337 \seq_gput_right:Nn
2338 \g_@@_renderers_seq
2339 { fencedCodeAttributeContextEnd }
2340 \prop_gput:Nnn
2341 \g_@@_renderer_arities_prop
2342 { fencedCodeAttributeContextEnd }
2343 { 0 }
2344 \ExplSyntaxOff

```

#### 2.2.5.16 Fenced Div Attribute Context Renderers

The following macros are only produced, when the `fencedDivs` 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.

```

2345 \ExplSyntaxOn
2346 \cs_gset_protected:Npn
2347 \markdownRendererFencedDivAttributeContextBegin
2348 {
2349   \markdownRendererFencedDivAttributeContextBeginPrototype
2350 }
2351 \seq_gput_right:Nn
2352 \g_@@_renderers_seq
2353 { fencedDivAttributeContextBegin }
2354 \prop_gput:Nnn
2355 \g_@@_renderer_arities_prop
2356 { fencedDivAttributeContextBegin }
2357 { 0 }
2358 \cs_gset_protected:Npn
2359 \markdownRendererFencedDivAttributeContextEnd
2360 {
2361   \markdownRendererFencedDivAttributeContextEndPrototype
2362 }
2363 \seq_gput_right:Nn
2364 \g_@@_renderers_seq
2365 { fencedDivAttributeContextEnd }
2366 \prop_gput:Nnn

```

```

2367 \g_@@_renderer_arities_prop
2368 { fencedDivAttributeContextEnd }
2369 { 0 }
2370 \ExplSyntaxOff

```

### 2.2.5.17 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.

```

2371 \ExplSyntaxOn
2372 \cs_gset_protected:Npn
2373 \markdownRendererHeaderAttributeContextBegin
2374 {
2375   \markdownRendererHeaderAttributeContextBeginPrototype
2376 }
2377 \seq_gput_right:Nn
2378 \g_@@_renderers_seq
2379 { headerAttributeContextBegin }
2380 \prop_gput:Nnn
2381 \g_@@_renderer_arities_prop
2382 { headerAttributeContextBegin }
2383 { 0 }
2384 \cs_gset_protected:Npn
2385 \markdownRendererHeaderAttributeContextEnd
2386 {
2387   \markdownRendererHeaderAttributeContextEndPrototype
2388 }
2389 \seq_gput_right:Nn
2390 \g_@@_renderers_seq
2391 { headerAttributeContextEnd }
2392 \prop_gput:Nnn
2393 \g_@@_renderer_arities_prop
2394 { headerAttributeContextEnd }
2395 { 0 }
2396 \ExplSyntaxOff

```

### 2.2.5.18 Heading Renderers

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

```

2397 \ExplSyntaxOn
2398 \cs_gset_protected:Npn
2399 \markdownRendererHeadingOne

```

```

2400 {
2401   \markdownRendererHeadingOnePrototype
2402 }
2403 \seq_gput_right:Nn
2404   \g_@@_renderers_seq
2405   { headingOne }
2406 \prop_gput:Nnn
2407   \g_@@_renderer_arities_prop
2408   { headingOne }
2409   { 1 }
2410 \ExplSyntaxOff

```

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

```

2411 \ExplSyntaxOn
2412 \cs_gset_protected:Npn
2413   \markdownRendererHeadingTwo
2414   {
2415     \markdownRendererHeadingTwoPrototype
2416   }
2417 \seq_gput_right:Nn
2418   \g_@@_renderers_seq
2419   { headingTwo }
2420 \prop_gput:Nnn
2421   \g_@@_renderer_arities_prop
2422   { headingTwo }
2423   { 1 }
2424 \ExplSyntaxOff

```

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

```

2425 \ExplSyntaxOn
2426 \cs_gset_protected:Npn
2427   \markdownRendererHeadingThree
2428   {
2429     \markdownRendererHeadingThreePrototype
2430   }
2431 \seq_gput_right:Nn
2432   \g_@@_renderers_seq
2433   { headingThree }
2434 \prop_gput:Nnn
2435   \g_@@_renderer_arities_prop
2436   { headingThree }
2437   { 1 }
2438 \ExplSyntaxOff

```

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

```

2439 \ExplSyntaxOn
2440 \cs_gset_protected:Npn
2441   \markdownRendererHeadingFour
2442   {
2443     \markdownRendererHeadingFourPrototype
2444   }
2445 \seq_gput_right:Nn
2446   \g_@@_renderers_seq
2447   { headingFour }
2448 \prop_gput:Nnn
2449   \g_@@_renderer_arities_prop
2450   { headingFour }
2451   { 1 }
2452 \ExplSyntaxOff

```

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

```

2453 \ExplSyntaxOn
2454 \cs_gset_protected:Npn
2455   \markdownRendererHeadingFive
2456   {
2457     \markdownRendererHeadingFivePrototype
2458   }
2459 \seq_gput_right:Nn
2460   \g_@@_renderers_seq
2461   { headingFive }
2462 \prop_gput:Nnn
2463   \g_@@_renderer_arities_prop
2464   { headingFive }
2465   { 1 }
2466 \ExplSyntaxOff

```

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

```

2467 \ExplSyntaxOn
2468 \cs_gset_protected:Npn
2469   \markdownRendererHeadingSix
2470   {
2471     \markdownRendererHeadingSixPrototype
2472   }
2473 \seq_gput_right:Nn
2474   \g_@@_renderers_seq
2475   { headingSix }
2476 \prop_gput:Nnn

```



```

2477 \g_@@_renderer_arities_prop
2478 { headingSix }
2479 { 1 }
2480 \ExplSyntaxOff

```

### 2.2.5.19 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.

```

2481 \ExplSyntaxOn
2482 \cs_gset_protected:Npn
2483 \markdownRendererInlineHtmlComment
2484 {
2485   \markdownRendererInlineHtmlCommentPrototype
2486 }
2487 \seq_gput_right:Nn
2488 \g_@@_renderers_seq
2489 { inlineHtmlComment }
2490 \prop_gput:Nnn
2491 \g_@@_renderer_arities_prop
2492 { inlineHtmlComment }
2493 { 1 }
2494 \ExplSyntaxOff

```

### 2.2.5.20 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.

```

2495 \ExplSyntaxOn
2496 \cs_gset_protected:Npn
2497 \markdownRendererInlineHtmlTag
2498 {
2499   \markdownRendererInlineHtmlTagPrototype
2500 }
2501 \seq_gput_right:Nn
2502 \g_@@_renderers_seq
2503 { inlineHtmlTag }
2504 \prop_gput:Nnn

```

```

2505 \g_@@_renderer_arities_prop
2506 { inlineHtmlTag }
2507 { 1 }
2508 \cs_gset_protected:Npn
2509 \markdownRendererInputBlockHtmlElement
2510 {
2511   \markdownRendererInputBlockHtmlElementPrototype
2512 }
2513 \seq_gput_right:Nn
2514 \g_@@_renderers_seq
2515 { inputBlockHtmlElement }
2516 \prop_gput:Nnn
2517 \g_@@_renderer_arities_prop
2518 { inputBlockHtmlElement }
2519 { 1 }
2520 \ExplSyntaxOff

```

### 2.2.5.21 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.

```

2521 \ExplSyntaxOn
2522 \cs_gset_protected:Npn
2523 \markdownRendererImage
2524 {
2525   \markdownRendererImagePrototype
2526 }
2527 \seq_gput_right:Nn
2528 \g_@@_renderers_seq
2529 { image }
2530 \prop_gput:Nnn
2531 \g_@@_renderer_arities_prop
2532 { image }
2533 { 4 }
2534 \ExplSyntaxOff

```

### 2.2.5.22 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.

```

2535 \ExplSyntaxOn
2536 \cs_gset_protected:Npn

```

```

2537 \markdownRendererImageAttributeContextBegin
2538 {
2539     \markdownRendererImageAttributeContextBeginPrototype
2540 }
2541 \seq_gput_right:Nn
2542 \g_@@_renderers_seq
2543 { imageAttributeContextBegin }
2544 \prop_gput:Nnn
2545 \g_@@_renderer_arities_prop
2546 { imageAttributeContextBegin }
2547 { 0 }
2548 \cs_gset_protected:Npn
2549 \markdownRendererImageAttributeContextEnd
2550 {
2551     \markdownRendererImageAttributeContextEndPrototype
2552 }
2553 \seq_gput_right:Nn
2554 \g_@@_renderers_seq
2555 { imageAttributeContextEnd }
2556 \prop_gput:Nnn
2557 \g_@@_renderer_arities_prop
2558 { imageAttributeContextEnd }
2559 { 0 }
2560 \ExplSyntaxOff

```

### 2.2.5.23 Interblock Separator Renderers

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

```

2561 \ExplSyntaxOn
2562 \cs_gset_protected:Npn
2563 \markdownRendererInterblockSeparator
2564 {
2565     \markdownRendererInterblockSeparatorPrototype
2566 }
2567 \seq_gput_right:Nn
2568 \g_@@_renderers_seq
2569 { interblockSeparator }
2570 \prop_gput:Nnn
2571 \g_@@_renderer_arities_prop
2572 { interblockSeparator }
2573 { 0 }
2574 \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.

```

2575 \ExplSyntaxOn
2576 \cs_gset_protected:Npn
2577   \markdownRendererParagraphSeparator
2578   {
2579     \markdownRendererParagraphSeparatorPrototype
2580   }
2581 \seq_gput_right:Nn
2582   \g_@@_renderers_seq
2583   { paragraphSeparator }
2584 \prop_gput:Nnn
2585   \g_@@_renderer_arities_prop
2586   { paragraphSeparator }
2587   { 0 }
2588 \ExplSyntaxOff

```

#### 2.2.5.24 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.

```

2589 \ExplSyntaxOn
2590 \cs_gset_protected:Npn
2591   \markdownRendererLineBlockBegin
2592   {
2593     \markdownRendererLineBlockBeginPrototype
2594   }
2595 \seq_gput_right:Nn
2596   \g_@@_renderers_seq
2597   { lineBlockBegin }
2598 \prop_gput:Nnn
2599   \g_@@_renderer_arities_prop
2600   { lineBlockBegin }
2601   { 0 }
2602 \cs_gset_protected:Npn
2603   \markdownRendererLineBlockEnd
2604   {
2605     \markdownRendererLineBlockEndPrototype
2606   }
2607 \seq_gput_right:Nn
2608   \g_@@_renderers_seq
2609   { lineBlockEnd }

```

```

2610 \prop_gput:Nnn
2611   \g_@@_renderer_arities_prop
2612   { lineBlockEnd }
2613   { 0 }
2614 \ExplSyntaxOff

```

### 2.2.5.25 Line Break Renderers

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

```

2615 \ExplSyntaxOn
2616 \cs_gset_protected:Npn
2617   \markdownRendererSoftLineBreak
2618   {
2619     \markdownRendererSoftLineBreakPrototype
2620   }
2621 \seq_gput_right:Nn
2622   \g_@@_renderers_seq
2623   { softLineBreak }
2624 \prop_gput:Nnn
2625   \g_@@_renderer_arities_prop
2626   { softLineBreak }
2627   { 0 }
2628 \ExplSyntaxOff

```

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

```

2629 \ExplSyntaxOn
2630 \cs_gset_protected:Npn
2631   \markdownRendererHardLineBreak
2632   {
2633     \markdownRendererHardLineBreakPrototype
2634   }
2635 \seq_gput_right:Nn
2636   \g_@@_renderers_seq
2637   { hardLineBreak }
2638 \prop_gput:Nnn
2639   \g_@@_renderer_arities_prop
2640   { hardLineBreak }
2641   { 0 }
2642 \ExplSyntaxOff

```

### 2.2.5.26 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.

```

2643 \ExplSyntaxOn
2644 \cs_gset_protected:Npn
2645   \markdownRendererLink
2646   {
2647     \markdownRendererLinkPrototype
2648   }
2649 \seq_gput_right:Nn
2650   \g_@@_renderers_seq
2651   { link }
2652 \prop_gput:Nnn
2653   \g_@@_renderer_arities_prop
2654   { link }
2655   { 4 }
2656 \ExplSyntaxOff

```

### 2.2.5.27 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.

```

2657 \ExplSyntaxOn
2658 \cs_gset_protected:Npn
2659   \markdownRendererLinkAttributeContextBegin
2660   {
2661     \markdownRendererLinkAttributeContextBeginPrototype
2662   }
2663 \seq_gput_right:Nn
2664   \g_@@_renderers_seq
2665   { linkAttributeContextBegin }
2666 \prop_gput:Nnn
2667   \g_@@_renderer_arities_prop
2668   { linkAttributeContextBegin }
2669   { 0 }
2670 \cs_gset_protected:Npn
2671   \markdownRendererLinkAttributeContextEnd
2672   {
2673     \markdownRendererLinkAttributeContextEndPrototype
2674   }
2675 \seq_gput_right:Nn
2676   \g_@@_renderers_seq
2677   { linkAttributeContextEnd }
2678 \prop_gput:Nnn
2679   \g_@@_renderer_arities_prop
2680   { linkAttributeContextEnd }
2681   { 0 }

```

```
2682 \ExplSyntaxOff
```

### 2.2.5.28 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.

```
2683 \ExplSyntaxOn
2684 \cs_gset_protected:Npn
2685   \markdownRendererMark
2686   {
2687     \markdownRendererMarkPrototype
2688   }
2689 \seq_gput_right:Nn
2690   \g_@@_renderers_seq
2691   { mark }
2692 \prop_gput:Nnn
2693   \g_@@_renderer_arities_prop
2694   { mark }
2695   { 1 }
2696 \ExplSyntaxOff
```

### 2.2.5.29 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.

```
2697 \ExplSyntaxOn
2698 \cs_gset_protected:Npn
2699   \markdownRendererDocumentBegin
2700   {
2701     \markdownRendererDocumentBeginPrototype
2702   }
2703 \seq_gput_right:Nn
2704   \g_@@_renderers_seq
2705   { documentBegin }
2706 \prop_gput:Nnn
2707   \g_@@_renderer_arities_prop
2708   { documentBegin }
2709   { 0 }
2710 \cs_gset_protected:Npn
2711   \markdownRendererDocumentEnd
```

```

2712 {
2713     \markdownRendererDocumentEndPrototype
2714 }
2715 \seq_gput_right:Nn
2716 \g_@@_renderers_seq
2717 { documentEnd }
2718 \prop_gput:Nnn
2719 \g_@@_renderer_arities_prop
2720 { documentEnd }
2721 { 0 }
2722 \ExplSyntaxOff

```

### 2.2.5.30 Non-Breaking Space Renderer

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

```

2723 \ExplSyntaxOn
2724 \cs_gset_protected:Npn
2725 \markdownRendererNbsp
2726 {
2727     \markdownRendererNbspPrototype
2728 }
2729 \seq_gput_right:Nn
2730 \g_@@_renderers_seq
2731 { nbsp }
2732 \prop_gput:Nnn
2733 \g_@@_renderer_arities_prop
2734 { nbsp }
2735 { 0 }
2736 \ExplSyntaxOff

```

### 2.2.5.31 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.

```

2737 \def\markdownRendererNote{%
2738     \markdownRendererNotePrototype}%
2739 \ExplSyntaxOn
2740 \seq_gput_right:Nn
2741 \g_@@_renderers_seq
2742 { note }
2743 \prop_gput:Nnn
2744 \g_@@_renderer_arities_prop
2745 { note }
2746 { 1 }
2747 \ExplSyntaxOff

```



### 2.2.5.32 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.

```
2748 \ExplSyntaxOn
2749 \cs_gset_protected:Npn
2750   \markdownRendererOlBegin
2751   {
2752     \markdownRendererOlBeginPrototype
2753   }
2754 \seq_gput_right:Nn
2755   \g_@@_renderers_seq
2756   { olBegin }
2757 \prop_gput:Nnn
2758   \g_@@_renderer_arities_prop
2759   { olBegin }
2760   { 0 }
2761 \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.

```
2762 \ExplSyntaxOn
2763 \cs_gset_protected:Npn
2764   \markdownRendererOlBeginTight
2765   {
2766     \markdownRendererOlBeginTightPrototype
2767   }
2768 \seq_gput_right:Nn
2769   \g_@@_renderers_seq
2770   { olBeginTight }
2771 \prop_gput:Nnn
2772   \g_@@_renderer_arities_prop
2773   { olBeginTight }
2774   { 0 }
2775 \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`).

```

2776 \ExplSyntaxOn
2777 \cs_gset_protected:Npn
2778   \markdownRendererFancyOlBegin
2779   {
2780     \markdownRendererFancyOlBeginPrototype
2781   }
2782 \seq_gput_right:Nn
2783   \g_@@_renderers_seq
2784   { fancyOlBegin }
2785 \prop_gput:Nnn
2786   \g_@@_renderer_arities_prop
2787   { fancyOlBegin }
2788   { 2 }
2789 \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.

```

2790 \ExplSyntaxOn
2791 \cs_gset_protected:Npn
2792   \markdownRendererFancyOlBeginTight
2793   {
2794     \markdownRendererFancyOlBeginTightPrototype
2795   }
2796 \seq_gput_right:Nn
2797   \g_@@_renderers_seq
2798   { fancyOlBeginTight }
2799 \prop_gput:Nnn
2800   \g_@@_renderer_arities_prop
2801   { fancyOlBeginTight }
2802   { 2 }
2803 \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.

```

2804 \ExplSyntaxOn
2805 \cs_gset_protected:Npn
2806   \markdownRendererOlItem
2807   {
2808     \markdownRendererOlItemPrototype
2809   }
2810 \seq_gput_right:Nn

```

```

2811 \g_@@_renderers_seq
2812 { olItem }
2813 \prop_gput:Nnn
2814 \g_@@_renderer_arities_prop
2815 { olItem }
2816 { 0 }
2817 \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.

```

2818 \ExplSyntaxOn
2819 \cs_gset_protected:Npn
2820 \markdownRendererOlItemEnd
2821 {
2822   \markdownRendererOlItemEndPrototype
2823 }
2824 \seq_gput_right:Nn
2825 \g_@@_renderers_seq
2826 { olItemEnd }
2827 \prop_gput:Nnn
2828 \g_@@_renderer_arities_prop
2829 { olItemEnd }
2830 { 0 }
2831 \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.

```

2832 \ExplSyntaxOn
2833 \cs_gset_protected:Npn
2834 \markdownRendererOlItemWithNumber
2835 {
2836   \markdownRendererOlItemWithNumberPrototype
2837 }
2838 \seq_gput_right:Nn
2839 \g_@@_renderers_seq
2840 { olItemWithNumber }
2841 \prop_gput:Nnn
2842 \g_@@_renderer_arities_prop
2843 { olItemWithNumber }
2844 { 1 }
2845 \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.

```

2846 \ExplSyntaxOn
2847 \cs_gset_protected:Npn
2848   \markdownRendererFancyO1Item
2849   {
2850     \markdownRendererFancyO1ItemPrototype
2851   }
2852 \seq_gput_right:Nn
2853   \g_@@_renderers_seq
2854   { fancyO1Item }
2855 \prop_gput:Nnn
2856   \g_@@_renderer_arities_prop
2857   { fancyO1Item }
2858   { 0 }
2859 \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.

```

2860 \ExplSyntaxOn
2861 \cs_gset_protected:Npn
2862   \markdownRendererFancyO1ItemEnd
2863   {
2864     \markdownRendererFancyO1ItemEndPrototype
2865   }
2866 \seq_gput_right:Nn
2867   \g_@@_renderers_seq
2868   { fancyO1ItemEnd }
2869 \prop_gput:Nnn
2870   \g_@@_renderer_arities_prop
2871   { fancyO1ItemEnd }
2872   { 0 }
2873 \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.

```

2874 \ExplSyntaxOn
2875 \cs_gset_protected:Npn
2876   \markdownRendererFancyO1ItemWithNumber
2877   {
2878     \markdownRendererFancyO1ItemWithNumberPrototype
2879   }

```

```

2880 \seq_gput_right:Nn
2881   \g_@@_renderers_seq
2882   { fancyO1ItemWithNumber }
2883 \prop_gput:Nnn
2884   \g_@@_renderer_arities_prop
2885   { fancyO1ItemWithNumber }
2886   { 1 }
2887 \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.

```

2888 \ExplSyntaxOn
2889 \cs_gset_protected:Npn
2890   \markdownRendererO1End
2891   {
2892     \markdownRendererO1EndPrototype
2893   }
2894 \seq_gput_right:Nn
2895   \g_@@_renderers_seq
2896   { olEnd }
2897 \prop_gput:Nnn
2898   \g_@@_renderer_arities_prop
2899   { olEnd }
2900   { 0 }
2901 \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.

```

2902 \ExplSyntaxOn
2903 \cs_gset_protected:Npn
2904   \markdownRendererO1EndTight
2905   {
2906     \markdownRendererO1EndTightPrototype
2907   }
2908 \seq_gput_right:Nn
2909   \g_@@_renderers_seq
2910   { olEndTight }
2911 \prop_gput:Nnn
2912   \g_@@_renderer_arities_prop
2913   { olEndTight }
2914   { 0 }
2915 \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.

```

2916 \ExplSyntaxOn
2917 \cs_gset_protected:Npn
2918   \markdownRendererFancy01End
2919   {
2920     \markdownRendererFancy01EndPrototype
2921   }
2922 \seq_gput_right:Nn
2923   \g_@@_renderers_seq
2924   { fancy01End }
2925 \prop_gput:Nnn
2926   \g_@@_renderer_arities_prop
2927   { fancy01End }
2928   { 0 }
2929 \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.

```

2930 \ExplSyntaxOn
2931 \cs_gset_protected:Npn
2932   \markdownRendererFancy01EndTight
2933   {
2934     \markdownRendererFancy01EndTightPrototype
2935   }
2936 \seq_gput_right:Nn
2937   \g_@@_renderers_seq
2938   { fancy01EndTight }
2939 \prop_gput:Nnn
2940   \g_@@_renderer_arities_prop
2941   { fancy01EndTight }
2942   { 0 }
2943 \ExplSyntaxOff

```

### 2.2.5.33 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.

```

2944 \ExplSyntaxOn

```

```

2945 \cs_gset_protected:Npn
2946   \markdownRendererInputRawInline
2947   {
2948     \markdownRendererInputRawInlinePrototype
2949   }
2950 \seq_gput_right:Nn
2951   \g_@@_renderers_seq
2952   { inputRawInline }
2953 \prop_gput:Nnn
2954   \g_@@_renderer_arities_prop
2955   { inputRawInline }
2956   { 2 }
2957 \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.

```

2958 \ExplSyntaxOn
2959 \cs_gset_protected:Npn
2960   \markdownRendererInputRawBlock
2961   {
2962     \markdownRendererInputRawBlockPrototype
2963   }
2964 \seq_gput_right:Nn
2965   \g_@@_renderers_seq
2966   { inputRawBlock }
2967 \prop_gput:Nnn
2968   \g_@@_renderer_arities_prop
2969   { inputRawBlock }
2970   { 2 }
2971 \ExplSyntaxOff

```

#### 2.2.5.34 Section Renderers

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

```

2972 \ExplSyntaxOn
2973 \cs_gset_protected:Npn
2974   \markdownRendererSectionBegin
2975   {
2976     \markdownRendererSectionBeginPrototype
2977   }
2978 \seq_gput_right:Nn
2979   \g_@@_renderers_seq
2980   { sectionBegin }
2981 \prop_gput:Nnn

```

```

2982 \g_@@_renderer_arities_prop
2983 { sectionBegin }
2984 { 0 }
2985 \cs_gset_protected:Npn
2986 \markdownRendererSectionEnd
2987 {
2988   \markdownRendererSectionEndPrototype
2989 }
2990 \seq_gput_right:Nn
2991 \g_@@_renderers_seq
2992 { sectionEnd }
2993 \prop_gput:Nnn
2994 \g_@@_renderer_arities_prop
2995 { sectionEnd }
2996 { 0 }
2997 \ExplSyntaxOff

```

### 2.2.5.35 Replacement Character Renderers

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

```

2998 \ExplSyntaxOn
2999 \cs_gset_protected:Npn
3000 \markdownRendererReplacementCharacter
3001 {
3002   \markdownRendererReplacementCharacterPrototype
3003 }
3004 \seq_gput_right:Nn
3005 \g_@@_renderers_seq
3006 { replacementCharacter }
3007 \prop_gput:Nnn
3008 \g_@@_renderer_arities_prop
3009 { replacementCharacter }
3010 { 0 }
3011 \ExplSyntaxOff

```

### 2.2.5.36 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`.

```

3012 \ExplSyntaxOn
3013 \cs_gset_protected:Npn
3014 \markdownRendererLeftBrace
3015 {
3016   \markdownRendererLeftBracePrototype
3017 }

```



```

3018 \seq_gput_right:Nn
3019   \g_@@_renderers_seq
3020   { leftBrace }
3021 \prop_gput:Nnn
3022   \g_@@_renderer_arities_prop
3023   { leftBrace }
3024   { 0 }
3025 \cs_gset_protected:Npn
3026   \markdownRendererRightBrace
3027   {
3028     \markdownRendererRightBracePrototype
3029   }
3030 \seq_gput_right:Nn
3031   \g_@@_renderers_seq
3032   { rightBrace }
3033 \prop_gput:Nnn
3034   \g_@@_renderer_arities_prop
3035   { rightBrace }
3036   { 0 }
3037 \cs_gset_protected:Npn
3038   \markdownRendererDollarSign
3039   {
3040     \markdownRendererDollarSignPrototype
3041   }
3042 \seq_gput_right:Nn
3043   \g_@@_renderers_seq
3044   { dollarSign }
3045 \prop_gput:Nnn
3046   \g_@@_renderer_arities_prop
3047   { dollarSign }
3048   { 0 }
3049 \cs_gset_protected:Npn
3050   \markdownRendererPercentSign
3051   {
3052     \markdownRendererPercentSignPrototype
3053   }
3054 \seq_gput_right:Nn
3055   \g_@@_renderers_seq
3056   { percentSign }
3057 \prop_gput:Nnn
3058   \g_@@_renderer_arities_prop
3059   { percentSign }
3060   { 0 }
3061 \cs_gset_protected:Npn
3062   \markdownRendererAmpersand
3063   {
3064     \markdownRendererAmpersandPrototype

```

```

3065     }
3066 \seq_gput_right:Nn
3067   \g_@@_renderers_seq
3068   { ampersand }
3069 \prop_gput:Nnn
3070   \g_@@_renderer_arities_prop
3071   { ampersand }
3072   { 0 }
3073 \cs_gset_protected:Npn
3074   \markdownRendererUnderscore
3075   {
3076     \markdownRendererUnderscorePrototype
3077   }
3078 \seq_gput_right:Nn
3079   \g_@@_renderers_seq
3080   { underscore }
3081 \prop_gput:Nnn
3082   \g_@@_renderer_arities_prop
3083   { underscore }
3084   { 0 }
3085 \cs_gset_protected:Npn
3086   \markdownRendererHash
3087   {
3088     \markdownRendererHashPrototype
3089   }
3090 \seq_gput_right:Nn
3091   \g_@@_renderers_seq
3092   { hash }
3093 \prop_gput:Nnn
3094   \g_@@_renderer_arities_prop
3095   { hash }
3096   { 0 }
3097 \cs_gset_protected:Npn
3098   \markdownRendererCircumflex
3099   {
3100     \markdownRendererCircumflexPrototype
3101   }
3102 \seq_gput_right:Nn
3103   \g_@@_renderers_seq
3104   { circumflex }
3105 \prop_gput:Nnn
3106   \g_@@_renderer_arities_prop
3107   { circumflex }
3108   { 0 }
3109 \cs_gset_protected:Npn
3110   \markdownRendererBackslash
3111   {

```

```

3112     \markdownRendererBackslashPrototype
3113   }
3114 \seq_gput_right:Nn
3115   \g_@@_renderers_seq
3116   { backslash }
3117 \prop_gput:Nnn
3118   \g_@@_renderer_arities_prop
3119   { backslash }
3120   { 0 }
3121 \cs_gset_protected:Npn
3122   \markdownRendererTilde
3123   {
3124     \markdownRendererTildePrototype
3125   }
3126 \seq_gput_right:Nn
3127   \g_@@_renderers_seq
3128   { tilde }
3129 \prop_gput:Nnn
3130   \g_@@_renderer_arities_prop
3131   { tilde }
3132   { 0 }
3133 \cs_gset_protected:Npn
3134   \markdownRendererPipe
3135   {
3136     \markdownRendererPipePrototype
3137   }
3138 \seq_gput_right:Nn
3139   \g_@@_renderers_seq
3140   { pipe }
3141 \prop_gput:Nnn
3142   \g_@@_renderer_arities_prop
3143   { pipe }
3144   { 0 }
3145 \ExplSyntaxOff

```

### 2.2.5.37 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.

```

3146 \ExplSyntaxOn
3147 \cs_gset_protected:Npn
3148   \markdownRendererStrikeThrough
3149   {
3150     \markdownRendererStrikeThroughPrototype
3151   }

```

```

3152 \seq_gput_right:Nn
3153   \g_@@_renderers_seq
3154   { strikeThrough }
3155 \prop_gput:Nnn
3156   \g_@@_renderer_arities_prop
3157   { strikeThrough }
3158   { 1 }
3159 \ExplSyntaxOff

```

### 2.2.5.38 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.

```

3160 \ExplSyntaxOn
3161 \cs_gset_protected:Npn
3162   \markdownRendererSubscript
3163   {
3164     \markdownRendererSubscriptPrototype
3165   }
3166 \seq_gput_right:Nn
3167   \g_@@_renderers_seq
3168   { subscript }
3169 \prop_gput:Nnn
3170   \g_@@_renderer_arities_prop
3171   { subscript }
3172   { 1 }

```

### 2.2.5.39 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.

```

3173 \cs_gset_protected:Npn
3174   \markdownRendererSuperscript
3175   {
3176     \markdownRendererSuperscriptPrototype
3177   }
3178 \seq_gput_right:Nn
3179   \g_@@_renderers_seq
3180   { superscript }
3181 \prop_gput:Nnn
3182   \g_@@_renderer_arities_prop
3183   { superscript }
3184   { 1 }
3185 \ExplSyntaxOff

```

#### 2.2.5.40 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.

```
3186 \ExplSyntaxOn
3187 \cs_gset_protected:Npn
3188   \markdownRendererTableAttributeContextBegin
3189   {
3190     \markdownRendererTableAttributeContextBeginPrototype
3191   }
3192 \seq_gput_right:Nn
3193   \g_@@_renderers_seq
3194   { tableAttributeContextBegin }
3195 \prop_gput:Nnn
3196   \g_@@_renderer_arities_prop
3197   { tableAttributeContextBegin }
3198   { 0 }
3199 \cs_gset_protected:Npn
3200   \markdownRendererTableAttributeContextEnd
3201   {
3202     \markdownRendererTableAttributeContextEndPrototype
3203   }
3204 \seq_gput_right:Nn
3205   \g_@@_renderers_seq
3206   { tableAttributeContextEnd }
3207 \prop_gput:Nnn
3208   \g_@@_renderer_arities_prop
3209   { tableAttributeContextEnd }
3210   { 0 }
3211 \ExplSyntaxOff
```

#### 2.2.5.41 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.

```

3212 \ExplSyntaxOn
3213 \cs_gset_protected:Npn
3214   \markdownRendererTable
3215   {
3216     \markdownRendererTablePrototype
3217   }
3218 \seq_gput_right:Nn
3219   \g_@@_renderers_seq
3220   { table }
3221 \prop_gput:Nnn
3222   \g_@@_renderer_arities_prop
3223   { table }
3224   { 3 }
3225 \ExplSyntaxOff

```

#### 2.2.5.42 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.

```

3226 \ExplSyntaxOn
3227 \cs_gset_protected:Npn
3228   \markdownRendererInlineMath
3229   {
3230     \markdownRendererInlineMathPrototype
3231   }
3232 \seq_gput_right:Nn
3233   \g_@@_renderers_seq
3234   { inlineMath }
3235 \prop_gput:Nnn
3236   \g_@@_renderer_arities_prop
3237   { inlineMath }
3238   { 1 }
3239 \cs_gset_protected:Npn
3240   \markdownRendererDisplayMath
3241   {
3242     \markdownRendererDisplayMathPrototype
3243   }
3244 \seq_gput_right:Nn
3245   \g_@@_renderers_seq
3246   { displayMath }
3247 \prop_gput:Nnn
3248   \g_@@_renderer_arities_prop

```

```

3249 { displayMath }
3250 { 1 }
3251 \ExplSyntaxOff

```

### 2.2.5.43 Thematic Break Renderer

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

```

3252 \ExplSyntaxOn
3253 \cs_gset_protected:Npn
3254   \markdownRendererThematicBreak
3255   {
3256     \markdownRendererThematicBreakPrototype
3257   }
3258 \seq_gput_right:Nn
3259   \g_@@_renderers_seq
3260   { thematicBreak }
3261 \prop_gput:Nnn
3262   \g_@@_renderer_arities_prop
3263   { thematicBreak }
3264   { 0 }
3265 \ExplSyntaxOff

```

### 2.2.5.44 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.

```

3266 \ExplSyntaxOn
3267 \cs_gset_protected:Npn
3268   \markdownRendererTickedBox
3269   {
3270     \markdownRendererTickedBoxPrototype
3271   }
3272 \seq_gput_right:Nn
3273   \g_@@_renderers_seq
3274   { tickedBox }
3275 \prop_gput:Nnn
3276   \g_@@_renderer_arities_prop
3277   { tickedBox }
3278   { 0 }
3279 \cs_gset_protected:Npn
3280   \markdownRendererHalfTickedBox
3281   {

```

```

3282     \markdownRendererHalfTickedBoxPrototype
3283   }
3284   \seq_gput_right:Nn
3285     \g_@@_renderers_seq
3286     { halfTickedBox }
3287   \prop_gput:Nnn
3288     \g_@@_renderer_arities_prop
3289     { halfTickedBox }
3290     { 0 }
3291   \cs_gset_protected:Npn
3292     \markdownRendererUntickedBox
3293     {
3294       \markdownRendererUntickedBoxPrototype
3295     }
3296   \seq_gput_right:Nn
3297     \g_@@_renderers_seq
3298     { untickedBox }
3299   \prop_gput:Nnn
3300     \g_@@_renderer_arities_prop
3301     { untickedBox }
3302     { 0 }
3303   \ExplSyntaxOff

```

#### 2.2.5.45 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.

```

3304   \ExplSyntaxOn
3305   \cs_gset_protected:Npn
3306     \markdownRendererWarning
3307     {
3308       \markdownRendererWarningPrototype
3309     }
3310   \cs_gset_protected:Npn
3311     \markdownRendererError
3312     {

```



```

3313     \markdownRendererErrorPrototype
3314   }
3315   \seq_gput_right:Nn
3316     \g_@@_renderers_seq
3317     { warning }
3318   \prop_gput:Nnn
3319     \g_@@_renderer_arities_prop
3320     { warning }
3321     { 4 }
3322   \seq_gput_right:Nn
3323     \g_@@_renderers_seq
3324     { error }
3325   \prop_gput:Nnn
3326     \g_@@_renderer_arities_prop
3327     { error }
3328     { 4 }
3329   \ExplSyntaxOff

```

#### 2.2.5.46 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.

```

3330   \ExplSyntaxOn
3331   \cs_gset_protected:Npn
3332     \markdownRendererJekyllDataBegin
3333     {
3334       \markdownRendererJekyllDataBeginPrototype
3335     }
3336   \seq_gput_right:Nn
3337     \g_@@_renderers_seq
3338     { jekyllDataBegin }
3339   \prop_gput:Nnn
3340     \g_@@_renderer_arities_prop
3341     { jekyllDataBegin }
3342     { 0 }
3343   \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.

```

3344   \ExplSyntaxOn
3345   \cs_gset_protected:Npn
3346     \markdownRendererJekyllDataEnd
3347     {
3348       \markdownRendererJekyllDataEndPrototype
3349     }

```

```

3350 \seq_gput_right:Nn
3351   \g_@@_renderers_seq
3352   { jekyllDataEnd }
3353 \prop_gput:Nnn
3354   \g_@@_renderer_arities_prop
3355   { jekyllDataEnd }
3356   { 0 }
3357 \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.

```

3358 \ExplSyntaxOn
3359 \cs_gset_protected:Npn
3360   \markdownRendererJekyllDataMappingBegin
3361   {
3362     \markdownRendererJekyllDataMappingBeginPrototype
3363   }
3364 \seq_gput_right:Nn
3365   \g_@@_renderers_seq
3366   { jekyllDataMappingBegin }
3367 \prop_gput:Nnn
3368   \g_@@_renderer_arities_prop
3369   { jekyllDataMappingBegin }
3370   { 2 }
3371 \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.

```

3372 \ExplSyntaxOn
3373 \cs_gset_protected:Npn
3374   \markdownRendererJekyllDataMappingEnd
3375   {
3376     \markdownRendererJekyllDataMappingEndPrototype
3377   }
3378 \seq_gput_right:Nn
3379   \g_@@_renderers_seq
3380   { jekyllDataMappingEnd }
3381 \prop_gput:Nnn
3382   \g_@@_renderer_arities_prop
3383   { jekyllDataMappingEnd }
3384   { 0 }
3385 \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.

```

3386 \ExplSyntaxOn
3387 \cs_gset_protected:Npn
3388   \markdownRendererJekyllDataSequenceBegin
3389   {
3390     \markdownRendererJekyllDataSequenceBeginPrototype
3391   }
3392 \seq_gput_right:Nn
3393   \g_@@_renderers_seq
3394   { jekyllDataSequenceBegin }
3395 \prop_gput:Nnn
3396   \g_@@_renderer_arities_prop
3397   { jekyllDataSequenceBegin }
3398   { 2 }
3399 \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.

```

3400 \ExplSyntaxOn
3401 \cs_gset_protected:Npn
3402   \markdownRendererJekyllDataSequenceEnd
3403   {
3404     \markdownRendererJekyllDataSequenceEndPrototype
3405   }
3406 \seq_gput_right:Nn
3407   \g_@@_renderers_seq
3408   { jekyllDataSequenceEnd }
3409 \prop_gput:Nnn
3410   \g_@@_renderer_arities_prop
3411   { jekyllDataSequenceEnd }
3412   { 0 }
3413 \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.

```

3414 \ExplSyntaxOn
3415 \cs_gset_protected:Npn

```

```

3416 \markdownRendererJekyllDataBoolean
3417 {
3418   \markdownRendererJekyllDataBooleanPrototype
3419 }
3420 \seq_gput_right:Nn
3421 \g_@@_renderers_seq
3422 { jekyllDataBoolean }
3423 \prop_gput:Nnn
3424 \g_@@_renderer_arities_prop
3425 { jekyllDataBoolean }
3426 { 2 }
3427 \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.

```

3428 \ExplSyntaxOn
3429 \cs_gset_protected:Npn
3430 \markdownRendererJekyllDataNumber
3431 {
3432   \markdownRendererJekyllDataNumberPrototype
3433 }
3434 \seq_gput_right:Nn
3435 \g_@@_renderers_seq
3436 { jekyllDataNumber }
3437 \prop_gput:Nnn
3438 \g_@@_renderer_arities_prop
3439 { jekyllDataNumber }
3440 { 2 }
3441 \ExplSyntaxOff

```

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

For each string scalar value, both macros are produced. Whereas `\markdownRendererJekyllDataTypographicString` receives the scalar value after all markdown markup and special  $\text{\TeX}$  characters in the string have been replaced by  $\text{\TeX}$  macros, `\markdownRendererJekyllDataProgrammaticString` receives the raw scalar value. Therefore, whereas the `\markdownRendererJekyllDataTypographicString` macro is more appropriate for texts that are supposed to be typeset with  $\text{\TeX}$ , such as document titles, author names, or exam questions, the

`\markdownRendererJekyllDataProgrammaticString` macro is more appropriate for identifiers and other programmatic text that won't be typeset by T<sub>E</sub>X.

```

3442 \ExplSyntaxOn
3443 \cs_gset_protected:Npn
3444   \markdownRendererJekyllDataTypographicString
3445   {
3446     \markdownRendererJekyllDataTypographicStringPrototype
3447   }
3448 \cs_gset_protected:Npn
3449   \markdownRendererJekyllDataProgrammaticString
3450   {
3451     \markdownRendererJekyllDataProgrammaticStringPrototype
3452   }
3453 \seq_gput_right:Nn
3454   \g_@@_renderers_seq
3455   { jekyllDataTypographicString }
3456 \prop_gput:Nnn
3457   \g_@@_renderer_arities_prop
3458   { jekyllDataTypographicString }
3459   { 2 }
3460 \seq_gput_right:Nn
3461   \g_@@_renderers_seq
3462   { jekyllDataProgrammaticString }
3463 \prop_gput:Nnn
3464   \g_@@_renderer_arities_prop
3465   { jekyllDataProgrammaticString }
3466   { 2 }
3467 \ExplSyntaxOff

```

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

```

3468 \ExplSyntaxOn
3469 \cs_gset:Npn
3470   \markdownRendererJekyllDataTypographicString
3471   {
3472     \cs_if_exist:NTF
3473       \markdownRendererJekyllDataString
3474       {
3475         \@@_if_option:nTF
3476           { experimental }
3477           {
3478             \markdownError
3479             {
3480               The~jekyllDataString~renderer~has~been~deprecated,~

```

```

3481         to-be-removed-in-Markdown-4.0.0
3482     }
3483 }
3484 {
3485     \markdownWarning
3486     {
3487         The~jekyllDataString~renderer~has~been~deprecated,~
3488         to-be-removed-in-Markdown-4.0.0
3489     }
3490     \markdownRendererJekyllDataString
3491 }
3492 }
3493 {
3494     \cs_if_exist:NTF
3495     \markdownRendererJekyllDataStringPrototype
3496     {
3497         \@@_if_option:nTF
3498         { experimental }
3499         {
3500             \markdownError
3501             {
3502                 The~jekyllDataString~renderer~prototype~
3503                 has~been~deprecated,~
3504                 to-be-removed-in-Markdown-4.0.0
3505             }
3506         }
3507         {
3508             \markdownWarning
3509             {
3510                 The~jekyllDataString~renderer~prototype~
3511                 has~been~deprecated,~
3512                 to-be-removed-in-Markdown-4.0.0
3513             }
3514             \markdownRendererJekyllDataStringPrototype
3515         }
3516     }
3517     {
3518         \markdownRendererJekyllDataTypographicStringPrototype
3519     }
3520 }
3521 }
3522 \seq_gput_right:Nn
3523 \g_@@_renderers_seq
3524 { jekyllDataString }
3525 \prop_gput:Nnn
3526 \g_@@_renderer_arities_prop
3527 { jekyllDataString }

```

```

3528 { 2 }
3529 \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.

```

3530 \ExplSyntaxOn
3531 \cs_gset_protected:Npn
3532   \markdownRendererJekyllDataEmpty
3533   {
3534     \markdownRendererJekyllDataEmptyPrototype
3535   }
3536 \seq_gput_right:Nn
3537   \g_@@_renderers_seq
3538   { jekyllDataEmpty }
3539 \prop_gput:Nnn
3540   \g_@@_renderer_arities_prop
3541   { jekyllDataEmpty }
3542   { 1 }
3543 \ExplSyntaxOff

```

#### 2.2.5.47 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.

```

3544 \ExplSyntaxOn
3545 \cs_new:Nn \@@_define_renderers:
3546   {
3547     \seq_map_inline:Nn
3548       \g_@@_renderers_seq
3549       {
3550         \@@_define_renderer:n
3551         { ##1 }
3552       }
3553   }
3554 \cs_new:Nn \@@_define_renderer:n
3555   {

```

```

3556 \@@_renderer_tl_to_csname:nN
3557 { #1 }
3558 \l_tmpa_tl
3559 \prop_get:NnN
3560 \g_@@_renderer_arities_prop
3561 { #1 }
3562 \l_tmpb_tl
3563 \@@_define_renderer:ncV
3564 { #1 }
3565 { \l_tmpa_tl }
3566 \l_tmpb_tl
3567 }
3568 \cs_new:Nn \@@_renderer_tl_to_csname:nN
3569 {
3570 \tl_set:Nn
3571 \l_tmpa_tl
3572 { \str_uppercase:n { #1 } }
3573 \tl_set:Nx
3574 #2
3575 {
3576 markdownRenderer
3577 \tl_head:f { \l_tmpa_tl }
3578 \tl_tail:n { #1 }
3579 }
3580 }
3581 \tl_new:N
3582 \l_@@_renderer_definition_tl
3583 \bool_new:N
3584 \g_@@_prepending_renderer_bool
3585 \bool_new:N
3586 \g_@@_appending_renderer_bool
3587 \bool_new:N
3588 \g_@@_unprotected_renderer_bool
3589 \cs_new:Nn \@@_define_renderer:nNn
3590 {
3591 \keys_define:nn
3592 { markdown/options/renderers }
3593 {
3594 #1 .code:n = {
3595 \tl_set:Nn
3596 \l_@@_renderer_definition_tl
3597 { ##1 }
3598 \regex_replace_all:nnN
3599 { \cP\#0 }
3600 { #1 }
3601 \l_@@_renderer_definition_tl
3602 \bool_if:nT

```



```

3603         {
3604             \g_@@_prepending_renderer_bool ||
3605             \g_@@_appending_renderer_bool
3606         }
3607         {
3608             \@@_tl_set_from_cs:NNn
3609             \l_tmpa_tl
3610             #2
3611             { #3 }
3612             \bool_if:NTF
3613             \g_@@_prepending_renderer_bool
3614             {
3615                 \tl_put_right:NV
3616                 \l_@@_renderer_definition_tl
3617                 \l_tmpa_tl
3618             }
3619             {
3620                 \tl_put_left:NV
3621                 \l_@@_renderer_definition_tl
3622                 \l_tmpa_tl
3623             }
3624         }
3625         \bool_if:NTF
3626         \g_@@_unprotected_renderer_bool
3627         {
3628             \tl_set:Nn
3629             \l_tmpa_tl
3630             { \cs_set:Npn }
3631         }
3632         {
3633             \tl_set:Nn
3634             \l_tmpa_tl
3635             { \cs_set_protected:Npn }
3636         }
3637         \exp_last_unbraced:NNV
3638         \cs_generate_from_arg_count:NNnV
3639         #2
3640         \l_tmpa_tl
3641         { #3 }
3642         \l_@@_renderer_definition_tl
3643     },
3644 }

```

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.

```

3645     \str_if_eq:nnT

```

```

3646     { #1 }
3647     { jekyllDataString }
3648     {
3649         \cs_undefine:N
3650         #2
3651     }
3652 }

```

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.

```

3653 \cs_new_protected:Nn
3654   \@@_tl_set_from_cs:NNn
3655   {
3656     \tl_set:Nn
3657       \l_tmpa_tl
3658       { #2 }
3659     \int_step_inline:nn
3660       { #3 }
3661       {
3662         \exp_args:Nnc
3663         \tl_put_right:Nn
3664         \l_tmpa_tl
3665         { @@_tl_set_from_cs_parameter_ ##1 }
3666       }
3667     \exp_args:NNV
3668     \tl_set:No
3669     \l_tmpb_tl
3670     \l_tmpa_tl
3671     \regex_replace_all:nnN
3672     { \cP. }
3673     { \0\0 }
3674     \l_tmpb_tl
3675     \int_step_inline:nn
3676     { #3 }
3677     {
3678       \regex_replace_all:nnN
3679       { \c { @@_tl_set_from_cs_parameter_ ##1 } }
3680       { \cP\# ##1 }
3681       \l_tmpb_tl
3682     }
3683     \tl_set:NV
3684     #1
3685     \l_tmpb_tl
3686   }
3687 \cs_generate_variant:Nn

```

```

3688 \@@_define_renderer:nNn
3689 { ncV }
3690 \cs_generate_variant:Nn
3691 \cs_generate_from_arg_count:NNnn
3692 { NNnV }
3693 \cs_generate_variant:Nn
3694 \tl_put_left:Nn
3695 { Nv }
3696 \keys_define:nn
3697 { markdown/options }
3698 {
3699     renderers .code:n = {
3700         \bool_gset_false:N
3701         \g_@@_unprotected_renderer_bool
3702         \keys_set:nn
3703         { markdown/options/renderers }
3704         { #1 }
3705     },
3706     unprotectedRenderers .code:n = {
3707         \bool_gset_true:N
3708         \g_@@_unprotected_renderer_bool
3709         \keys_set:nn
3710         { markdown/options/renderers }
3711         { #1 }
3712     },
3713 }

```

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.
  }
}

```

```

3714 \tl_new:N
3715 \l_@@_renderer_glob_definition_tl
3716 \seq_new:N
3717 \l_@@_renderer_glob_results_seq
3718 \regex_const:Nn
3719 \c_@@_prepending_key_regex
3720 { \^$ }
3721 \regex_const:Nn
3722 \c_@@_appending_key_regex
3723 { [\$+]$ }

```

```

3724 \regex_const:Nn
3725   \c_@@_prepending_or_appending_key_regex
3726   { [\^\$+]\$ }
3727 \keys_define:nn
3728   { markdown/options/renderers }
3729   {
3730     unknown .code:n = {

```

Besides defining renderers at once, we can also define them incrementally using the prepending ( $\text{\textasciitilde{=}}$ ) and appending ( $\text{\$=}$  or  $\text{+=}$ ) operators. This can be especially useful in defining rules for processing different HTML class names and identifiers:

```

\markdownSetup{
  renderers = {
    % Start with empty renderers.
    headerAttributeContextBegin = \begingroup,
    headerAttributeContextEnd = \endgroup,
    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 += {...},
        },
      },
    },
  },
}

```

```

3731   % TODO: Use `regex_if_match` in TeX Live 2025.
3732 \regex_match:NVT
3733   \c_@@_prepending_key_regex
3734   \l_keys_key_str
3735   {
3736     \bool_gset_true:N

```

```

3737         \g_@@_prepending_renderer_bool
3738     }
3739     % TODO: Use `\\regex_if_match` in TeX Live 2025.
3740     \regex_match:NVT
3741         \c_@@_appending_key_regex
3742         \l_keys_key_str
3743     {
3744         \bool_gset_true:N
3745         \g_@@_appending_renderer_bool
3746     }
3747     % TODO: Use `\\regex_if_match` in TeX Live 2025.
3748     \regex_match:NVTF
3749         \c_@@_prepending_or_appending_key_regex
3750         \l_keys_key_str
3751     {
3752         \tl_set:NV
3753             \l_tmpa_tl
3754             \l_keys_key_str
3755         \regex_replace_once:NnN
3756             \c_@@_prepending_or_appending_key_regex
3757             { }
3758             \l_tmpa_tl
3759         \tl_set:Nx
3760             \l_tmpb_tl
3761             { { \l_tmpa_tl } = }
3762         \tl_put_right:Nn
3763             \l_tmpb_tl
3764             { { #1 } }
3765         \keys_set:nV
3766             { markdown/options/renderers }
3767             \l_tmpb_tl
3768         \bool_gset_false:N
3769             \g_@@_prepending_renderer_bool
3770         \bool_gset_false:N
3771             \g_@@_appending_renderer_bool
3772     }

```

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

```

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

```

```
}
```

Wildcards and enumerations can be combined:

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

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

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

```
3773 {
3774   \@@_glob_seq:VnN
3775   \l_keys_key_str
3776   { g_@@_renderers_seq }
3777   \l_@@_renderer_glob_results_seq
3778   \seq_if_empty:NTF
3779   \l_@@_renderer_glob_results_seq
3780   {
3781     \msg_error:nnV
3782     { markdown }
3783     { undefined-renderer }
3784     \l_keys_key_str
3785   }
3786   {
3787     \tl_set:Nn
3788     \l_@@_renderer_glob_definition_tl
3789     { \exp_not:n { #1 } }
3790     \seq_map_inline:Nn
3791     \l_@@_renderer_glob_results_seq
3792     {
3793       \tl_set:Nn
3794       \l_tmpa_tl
3795       { { ##1 } = }
3796       \tl_put_right:Nx
3797       \l_tmpa_tl
3798       { { \l_@@_renderer_glob_definition_tl } }
```

```

3799             \keys_set:nV
3800             { markdown/options/renderers }
3801             \l_tmpa_tl
3802         }
3803     }
3804 }
3805 },
3806 }
3807 \msg_new:nnn
3808 { markdown }
3809 { undefined-renderer }
3810 {
3811     Renderer~#1~is~undefined.
3812 }
3813 \cs_generate_variant:Nn
3814 \@@_glob_seq:nnN
3815 { VnN }
3816 \cs_generate_variant:Nn
3817 \cs_generate_from_arg_count:NNnn
3818 { cNvV }
3819 \cs_generate_variant:Nn
3820 \msg_error:nnn
3821 { nnV }
3822 \prg_generate_conditional_variant:Nnn
3823 % TODO: Use `regex_if_match` in TeX Live 2025.
3824 \regex_match:Nn % noqa: w202
3825 { NV }
3826 { T, TF }
3827 \prop_new:N
3828 \g_@@_glob_cache_prop
3829 \tl_new:N
3830 \l_@@_current_glob_tl
3831 \cs_new:Nn
3832 \@@_glob_seq:nnN
3833 {
3834     \tl_set:Nn
3835     \l_@@_current_glob_tl
3836     { ^ #1 $ }
3837     \prop_get:NeNTF
3838     \g_@@_glob_cache_prop
3839     { #2 / \l_@@_current_glob_tl }
3840     \l_tmpa_clist
3841     {
3842         \seq_set_from_clist:NN
3843         #3
3844         \l_tmpa_clist
3845     }

```

```

3846     {
3847         \seq_clear:N
3848         #3
3849         \regex_replace_all:nnN
3850         { \* }
3851         { .* }
3852         \l_@@_current_glob_tl
3853         \regex_set:NV
3854         \l_tmpa_regex
3855         \l_@@_current_glob_tl
3856         \seq_map_inline:cn
3857         { #2 }
3858         {
3859             % TODO: Use \regex_if_match in TeX Live 2025.
3860             \regex_match:NnT % noqa: w202
3861             \l_tmpa_regex
3862             { ##1 }
3863             {
3864                 \seq_put_right:Nn
3865                 #3
3866                 { ##1 }
3867             }
3868         }
3869         \clist_set_from_seq:NN
3870         \l_tmpa_clist
3871         #3
3872         \prop_gput:NeV
3873         \g_@@_glob_cache_prop
3874         { #2 / \l_@@_current_glob_tl }
3875         \l_tmpa_clist
3876     }
3877 }
3878 \cs_generate_variant:Nn
3879 \regex_set:Nn
3880 { NV }
3881 \cs_generate_variant:Nn
3882 \prop_gput:Nnn
3883 { NeV }

```

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

```

3884 \str_if_eq:VVT
3885 \c_@@_top_layer_tl
3886 \c_@@_option_layer_plain_tex_tl
3887 {
3888     \@@_define_renderers:

```



```

3889   }
3890 \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`.

```

3891 \ExplSyntaxOn
3892 \keys_define:nn
3893   { markdown/jekyllData }
3894   { }
3895 \ExplSyntaxOff

```

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

```

3896 \ExplSyntaxOn
3897 \@@_with_various_cases:nn
3898   { jekyllDataRenderers }
3899   {
3900     \keys_define:nn
3901       { markdown/options }
3902       {
3903         #1 .code:n = {
3904           \tl_set:Nn
3905             \l_tmpa_tl
3906             { ##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.

```

3907         \tl_replace_all:NnV
3908           \l_tmpa_tl
3909           { / }
3910         \c_backslash_str
3911         \keys_set:nV
3912           { markdown/options/jekyll-data-renderers }
3913           \l_tmpa_tl
3914       },
3915   }
3916 }
3917 \keys_define:nn
3918   { markdown/options/jekyll-data-renderers }
3919   {
3920     unknown .code:n = {

```

```

3921     \tl_set_eq:NN
3922     \l_tmpa_tl
3923     \l_keys_key_str
3924     \tl_replace_all:Nvn
3925     \l_tmpa_tl
3926     \c_backslash_str
3927     { / }
3928     \tl_put_right:Nn
3929     \l_tmpa_tl
3930     {
3931         .code:n = { #1 }
3932     }
3933     \keys_define:nV
3934     { markdown/jekyllData }
3935     \l_tmpa_tl
3936 }
3937 }
3938 \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 `<module>`.

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

We define the command `\@@_define_renderer_prototypes:` that defines plain  $\text{\TeX}$  macros for token renderer prototypes. Furthermore, the `\markdownSetup` macro also accepts the `rendererPrototypes` and `unprotectedRendererPrototypes` keys. The value for these keys must be a list of key-values, where the keys correspond to the markdown token renderer prototype macros and the values are new definitions of these token renderer prototypes.

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

```

3939 \ExplSyntaxOn
3940 \cs_new:Nn \@@_define_renderer_prototypes:
3941 {
3942     \seq_map_inline:Nn
3943     \g_@@_renderers_seq
3944     {
3945         \@@_define_renderer_prototype:n
3946         { ##1 }
3947     }
3948 }

```

```

3949 \cs_new:Nn \@@_define_renderer_prototype:n
3950 {
3951   \@@_renderer_prototype_tl_to_csname:nN
3952   { #1 }
3953   \l_tmpa_tl
3954   \prop_get:NnN
3955   \g_@@_renderer_arities_prop
3956   { #1 }
3957   \l_tmpb_tl
3958   \@@_define_renderer_prototype:ncV
3959   { #1 }
3960   { \l_tmpa_tl }
3961   \l_tmpb_tl
3962 }
3963 \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
3964 {
3965   \tl_set:Nn
3966   \l_tmpa_tl
3967   { \str_uppercase:n { #1 } }
3968   \tl_set:Nx
3969   #2
3970   {
3971     markdownRenderer
3972     \tl_head:f { \l_tmpa_tl }
3973     \tl_tail:n { #1 }
3974     Prototype
3975   }
3976 }
3977 \tl_new:N
3978 \l_@@_renderer_prototype_definition_tl
3979 \bool_new:N
3980 \g_@@_prepending_renderer_prototype_bool
3981 \bool_new:N
3982 \g_@@_appending_renderer_prototype_bool
3983 \bool_new:N
3984 \g_@@_unprotected_renderer_prototype_bool
3985 \cs_new:Nn \@@_define_renderer_prototype:nNn
3986 {
3987   \keys_define:nn
3988   { markdown/options/renderer-prototypes }
3989   {
3990     #1 .code:n = {
3991       \tl_set:Nn
3992       \l_@@_renderer_prototype_definition_tl
3993       { ##1 }
3994       \regex_replace_all:nnN
3995       { \cP\#0 }

```

```

3996         { #1 }
3997         \l_@@_renderer_prototype_definition_tl
3998     \bool_if:nT
3999     {
4000         \g_@@_prepending_renderer_prototype_bool ||
4001         \g_@@_appending_renderer_prototype_bool
4002     }
4003     {
4004         \@@_tl_set_from_cs:NNn
4005         \l_tmpa_tl
4006         #2
4007         { #3 }
4008     \bool_if:NTF
4009     \g_@@_prepending_renderer_prototype_bool
4010     {
4011         \tl_put_right:NV
4012         \l_@@_renderer_prototype_definition_tl
4013         \l_tmpa_tl
4014     }
4015     {
4016         \tl_put_left:NV
4017         \l_@@_renderer_prototype_definition_tl
4018         \l_tmpa_tl
4019     }
4020     }
4021 \bool_if:NTF
4022 \g_@@_unprotected_renderer_prototype_bool
4023 {
4024     \tl_set:Nn
4025     \l_tmpa_tl
4026     { \cs_set:Npn }
4027 }
4028 {
4029     \tl_set:Nn
4030     \l_tmpa_tl
4031     { \cs_set_protected:Npn }
4032 }
4033 \exp_last_unbraced:NNV
4034 \cs_generate_from_arg_count:NNnV
4035 #2
4036 \l_tmpa_tl
4037 { #3 }
4038 \l_@@_renderer_prototype_definition_tl
4039 },
4040 }

```

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.

```

4041 \str_if_eq:nnF
4042   { #1 }
4043   { jekyllDataString }
4044   {
4045     \cs_if_free:NT
4046       #2
4047       {
4048         \cs_generate_from_arg_count:NNnn
4049           #2
4050           \cs_gset_protected:Npn
4051             { #3 }
4052             { }
4053       }
4054   }
4055 }
4056 \cs_generate_variant:Nn
4057   \@@_define_renderer_prototype:nNn
4058   { 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.
  }
}

```

```

4059 \keys_define:nn
4060   { markdown/options/renderer-prototypes }
4061   {
4062     unknown .code:n = {

```

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

```

\markdownSetup{
  rendererPrototypes = {
    % Start with empty renderer prototypes.
    headerAttributeContextBegin = \begingroup,
    headerAttributeContextEnd = \endgroup,

```

```

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 += {...},
    },
  }
},
}
}

```

```

4063      % TODO: Use `\\regex_if_match` in TeX Live 2025.
4064      \\regex_match:NVT
4065      \\c_@@_prepending_key_regex
4066      \\l_keys_key_str
4067      {
4068        \\bool_gset_true:N
4069        \\g_@@_prepending_renderer_prototype_bool
4070      }
4071      % TODO: Use `\\regex_if_match` in TeX Live 2025.
4072      \\regex_match:NVT
4073      \\c_@@_appending_key_regex
4074      \\l_keys_key_str
4075      {
4076        \\bool_gset_true:N
4077        \\g_@@_appending_renderer_prototype_bool
4078      }
4079      % TODO: Use `\\regex_if_match` in TeX Live 2025.
4080      \\regex_match:NVTF
4081      \\c_@@_prepending_or_appending_key_regex
4082      \\l_keys_key_str
4083      {
4084        \\tl_set:NV
4085        \\l_tmpa_tl
4086        \\l_keys_key_str

```

```

4087         \regex_replace_once:NnN
4088         \c_@@_prepending_or_appending_key_regex
4089         { }
4090         \l_tmpa_tl
4091         \tl_set:Nx
4092         \l_tmpb_tl
4093         { { \l_tmpa_tl } = }
4094         \tl_put_right:Nn
4095         \l_tmpb_tl
4096         { { #1 } }
4097         \keys_set:nV
4098         { markdown/options/renderer-prototypes }
4099         \l_tmpb_tl
4100         \bool_gset_false:N
4101         \g_@@_prepending_renderer_prototype_bool
4102         \bool_gset_false:N
4103         \g_@@_appending_renderer_prototype_bool
4104     }

```

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

```

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

```

Wildcards and enumerations can be combined:

```

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

```

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

```

\markdownSetup{
  rendererPrototypes = {

```

```

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

```

```

4105     {
4106         \@@_glob_seq:VnN
4107         \l_keys_key_str
4108         { g_@@_renderers_seq }
4109         \l_@@_renderer_glob_results_seq
4110         \seq_if_empty:NTF
4111         \l_@@_renderer_glob_results_seq
4112         {
4113             \msg_error:nnV
4114             { markdown }
4115             { undefined-renderer-prototype }
4116             \l_keys_key_str
4117         }
4118         {
4119             \tl_set:Nn
4120             \l_@@_renderer_glob_definition_tl
4121             { \exp_not:n { #1 } }
4122             \seq_map_inline:Nn
4123             \l_@@_renderer_glob_results_seq
4124             {
4125                 \tl_set:Nn
4126                 \l_tmpa_tl
4127                 { { ##1 } = }
4128                 \tl_put_right:Nx
4129                 \l_tmpa_tl
4130                 { { \l_@@_renderer_glob_definition_tl } }
4131                 \keys_set:nV
4132                 { markdown/options/renderer-prototypes }
4133                 \l_tmpa_tl
4134             }
4135         }
4136     }
4137 },
4138 }
4139 \msg_new:nnn
4140 { markdown }
4141 { undefined-renderer-prototype }
4142 {
4143     Renderer~prototype~#1~is~undefined.
4144 }
4145 \@@_with_various_cases:nn
4146 { rendererPrototypes }
4147 {

```



```

4148     \keys_define:nn
4149     { markdown/options }
4150     {
4151       #1 .code:n = {
4152         \bool_gset_false:N
4153         \g_@@_unprotected_renderer_prototype_bool
4154         \keys_set:nn
4155         { markdown/options/renderer-prototypes }
4156         { ##1 }
4157       },
4158     }
4159   }
4160 \@@_with_various_cases:nn
4161 { unprotectedRendererPrototypes }
4162 {
4163   \keys_define:nn
4164   { markdown/options }
4165   {
4166     #1 .code:n = {
4167       \bool_gset_true:N
4168       \g_@@_unprotected_renderer_prototype_bool
4169       \keys_set:nn
4170       { markdown/options/renderer-prototypes }
4171       { ##1 }
4172     },
4173   }
4174 }

```

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.

```

4175 \str_if_eq:VVT
4176 \c_@@_top_layer_tl
4177 \c_@@_option_layer_plain_tex_tl
4178 {
4179   \@@_define_renderer_prototypes:
4180 }
4181 \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*.

```
4182 \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.

```
4183 \let\markdownReadAndConvert\relax
```

```
4184 \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*.

```
4185 \catcode`\|=0\catcode`\=12%
4186 |gdef|markdownBegin{%
4187   |markdownReadAndConvert{\markdownEnd}%
4188                               {|markdownEnd}}%
4189 |gdef|yamlBegin{%
4190   |begingroup
4191   |yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
4192   |markdownReadAndConvert{\yamlEnd}%
4193                               {|yamlEnd}}%
4194 |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.

```
4195 \ExplSyntaxOn
4196 \keys_define:nn
4197   { markdown/options }
4198   {
4199     code .code:n = { #1 },
4200   }
```

```
4201 \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.

```
4202 \ExplSyntaxOn
4203 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
4204 \cs_generate_variant:Nn
4205   \tl_const:Nn
4206   { NV }
4207 \tl_if_exist:NF
4208   \c_@@_top_layer_tl
4209   {
4210     \tl_const:NV
4211       \c_@@_top_layer_tl
4212       \c_@@_option_layer_latex_tl
4213   }
4214 \ExplSyntaxOff
4215 \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 2.2.5.47) 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 and YAML

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.

```
4216 \newenvironment{markdown}\relax\relax
4217 \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:

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

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

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

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

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

Specifically, the `\markdown` macro must appear at the end of the replacement 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:

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

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

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

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

```
4218 \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 $\text{\LaTeX}$ hooks with the Markdown package

$\text{\LaTeX}$  provides an intricate hook management system that allows users to insert extra material before and after certain  $\text{\TeX}$  macros and  $\text{\LaTeX}$  environments, among other things. [15, Section 3.1.2]

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

```

\documentclass{article}
\usepackage{markdown}
\begin{document}
\AddToHook{cmd/markdownRendererEmphasis/before}{emphasis: }
\AddToHook{env/markdown/before}{<markdown>}
\AddToHook{env/markdown/after}{</markdown>}
\begin{markdown}
foo _bar_ baz!
\end{markdown}
\end{document}

```

Processing the above example with  $\text{\LaTeX}$  will produce the text “foo emphasis: \_bar\_ baz!”, as expected.

However, using hooks to insert extra material after  $\text{\TeX}$  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  $\text{\LaTeX}$  will produce the text “foo \_bar\_ baz!”, as expected.

However, the default renderer for emphasis uses currying and calls the renderer prototype in a way that prevents the use of hooks to insert extra material after the renderer, see Section 2.2.5.14. 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 2.2.5.41.

### 2.3.3 Options

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

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

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



### 2.3.3.1 Finalizing and Freezing the Cache

To ensure compatibility with the `minted` package [16, Section 5.1], which supports the `finalizcache` and `frozenscache` package options with similar semantics to the `finalizeCache` and `frozenCache` plain  $\text{\TeX}$  options, the Markdown package also recognizes these as aliases and accepts them as document class options. By passing `finalizcache` and `frozenscache` as document class options, you may conveniently control the behavior of both packages at once:

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

We hope that other packages will support the `finalizcache` and `frozenscache` package options in the future, so that they can become a standard interface for preparing  $\text{\LaTeX}$  document sources for distribution.

```
4219 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
4220 \DeclareOption{frozenscache}{\markdownSetup{frozenCache}}
```

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

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

```
4221 \ExplSyntaxOn
4222 \str_if_eq:VVT
4223   \c_@@_top_layer_tl
4224   \c_@@_option_layer_latex_tl
4225   {
4226     \@@define_option_commands_and_keyvals:
4227     \@@define_renderers:
4228     \@@define_renderer_prototypes:
4229   }
4230 \ExplSyntaxOff
```

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

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

```

    cacheDir = /tmp,
}

```

### 2.3.4 Themes

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

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

If the L<sup>A</sup>T<sub>E</sub>X option with keys `theme` or `import` is (repeatedly) specified in the `\usepackage` macro, the loading of the theme(s) will be postponed in first-in-first-out order until after the Markdown L<sup>A</sup>T<sub>E</sub>X package has been loaded. Otherwise, the theme(s) will be loaded immediately. For example, 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}

```

```

4231 \newif\ifmarkdownLaTeXLoaded
4232 \markdownLaTeXLoadedfalse

```

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

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.

```

4233 \ExplSyntaxOn
4234 \prop_new:N
4235   \g_@@_latex_built_in_themes_prop
4236 \ExplSyntaxOff

```

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

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

```
4237 \AtEndOfPackage{\markdownLaTeXLoadedtrue}
```

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

```
4238 \ExplSyntaxOn
4239 \str_if_eq:VVT
4240   \c_@@_top_layer_tl
4241   \c_@@_option_layer_latex_tl
4242   {
4243     \use:c
4244       { ExplSyntaxOff }
4245     \AtEndOfPackage
4246       {
4247         \@@_if_option:nF
4248           { noDefaults }
4249           {
4250             \@@_if_option:nTF
4251               { experimental }
4252               {
4253                 \@@_setup:n
4254                   { theme = witiko/markdown/defaults@experimental }
4255               }
4256               {
4257                 \@@_setup:n
4258                   { theme = witiko/markdown/defaults }
4259               }
4260           }
4261       }
4262     \use:c
4263       { ExplSyntaxOn }
4264   }
4265 \ExplSyntaxOff
```

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

### 2.3.5 Snippets

In Section 2.2.4, we described the concept of snippets.

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

**witiko/glossaries/import-acronyms** Imports all acronyms from the  $\text{\LaTeX}$  package **glossaries**, appends them to the **acronyms** option and updates the corresponding

renderers to recognize these acronyms in running text and link them to the corresponding glossary entries.

This snippet is provided by the theme `witiko/glossaries@v1`.

See Section 3.3.3 for implementation details of the built-in L<sup>A</sup>T<sub>E</sub>X snippets.

## 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.

```

4266 \ExplSyntaxOn
4267 \tl_const:Nn \c_@@_option_layer_context_tl { context }
4268 \cs_generate_variant:Nn
4269   \tl_const:Nn
4270   { NV }
4271 \tl_if_exist:NF
4272   \c_@@_top_layer_tl
4273   {
4274     \tl_const:NV
4275       \c_@@_top_layer_tl
4276       \c_@@_option_layer_context_tl
4277   }
4278 \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).

```

4279 \writestatus{loading}{ConTEXt User Module / markdown}%
4280 \startmodule[markdown]
4281 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
4282   \do#\do\^\do\_do\%do\~}%
4283 \input markdown/markdown

```

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

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

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

### 2.4.1 Typesetting Markdown 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  $\TeX$  interface.

```
4284 \let\startmarkdown\relax
```

```
4285 \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 Con $\TeX$ t code showcases the usage of the `\startmarkdown` and `\stopmarkdown` macros:

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

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

```
4286 \let\startyaml\relax
```

```
4287 \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 Con $\TeX$ t 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  $\TeX$  interface.

```
4288 \let\inputmarkdown\relax
```

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

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

```

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

```

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

```

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

```

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

```
4289 \let\inputyaml\relax
```

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

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

```

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

```

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

```

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

```

## 2.4.2 Options

The ConT<sub>E</sub>Xt options are represented by a comma-delimited list of  $\langle key \rangle = \langle value \rangle$  pairs. For boolean options, the  $= \langle value \rangle$  part is optional, and  $\langle key \rangle$  will be interpreted as  $\langle key \rangle = \text{true}$  (or, equivalently,  $\langle key \rangle = \text{yes}$ ) if the  $= \langle value \rangle$  part has been omitted.

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

The ConT<sub>E</sub>Xt options may be specified when using the `\inputmarkdown` macro (see Section 2.4), via the `\markdownSetup` macro, or via the `\setupmarkdown[#1]` macro, which is an alias for `\markdownSetup{#1}`.

```

4290 \ExplSyntaxOn
4291 \cs_new:Npn
4292   \setupmarkdown
4293   [ #1 ]
4294   {
4295     \@@_setup:n
4296     { #1 }
4297   }

```

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

```

4298 \cs_gset_eq:NN
4299   \setupyaml
4300   \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.

```

4301 \cs_new:Nn \@@_caseless:N

```

```

4302 {
4303   \regex_replace_all:nnN
4304     { ([a-z])([A-Z]) }
4305     { \1 \c { str_lowercase:n } \cB\{ \2 \cE\} }
4306     #1
4307   \tl_set:Nx
4308     #1
4309     { #1 }
4310 }
4311 \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.

```

4312 \str_if_eq:VVT
4313   \c_@@_top_layer_tl
4314   \c_@@_option_layer_context_tl
4315   {
4316     \@@_define_option_commands_and_keyvals:
4317     \@@_define_renderers:
4318     \@@_define_renderer_prototypes:
4319   }

```

### 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.

```
4320 \prop_new:N
4321   \g_@@_context_built_in_themes_prop
4322 \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.

```
4323 \startmodule[markdownthemewitiko_markdown_defaults]
4324 \unprotect
```

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.

```
4325 local upper, format, length =
4326   string.upper, string.format, string.len
4327 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, Cp, any =
4328   lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
4329   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.

```
4330 local early_warnings = {}
4331 local unicode_data = require("markdown-unicode-data")
4332 if metadata.version ~= unicode_data.metadata.version then
4333     table.insert(
4334         early_warnings,
4335         "markdown.lua " .. metadata.version .. " used with " ..
4336         "markdown-unicode-data.lua " .. unicode_data.metadata.version .. ".")
4337     )
4338 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).

```
4339 local function depth_first_search(node, path, visit, leave)
4340     assert(node._type == "intermediate")
4341     visit(node, path)
4342     for label, child in pairs(node) do
4343         if label == "_type" then
4344             goto continue
4345         end
4346         if child._type ~= "intermediate" then
4347             goto continue
4348         end
4349         depth_first_search(child, path .. label, visit, leave)
4350         ::continue::
4351     end
4352     for label, child in pairs(node) do
4353         if label == "_type" then
4354             goto continue
4355         end
4356         if child._type ~= "leaf" then
4357             goto continue
4358         end
4359         visit(child, path)
4360         ::continue::
4361     end
4362 end
```

```

4361     end
4362     leave(node, path)
4363 end

```

This function is defined in the file [markdown-parser.lua](#) as well, so that it can be used to define the [extensions.acronyms](#) built-in syntax extension. The function [serialize\\_byte\\_parser](#) produces the Lua code for a PEG parser that matches a single byte.

```

4364 local function serialize_byte_parser(byte)
4365     if byte == '"' then
4366         return "P('\" .. byte .. \"')"
4367     elseif byte == "\\" then
4368         return "P(\"\\\\\\\")"
4369     else
4370         return "P('\" .. byte .. \"')"
4371     end
4372 end

```

The function [serialize\\_byte\\_range\\_parser](#) produces the Lua code for a PEG parser that matches a contiguous range of bytes.

```

4373 local function serialize_byte_range_parser(start_byte, end_byte)
4374     assert(start_byte <= end_byte)
4375     if start_byte == "\\" then
4376         start_byte = "\\\\"
4377     end
4378     if end_byte == "\\" then
4379         end_byte = "\\\\"
4380     end
4381     if start_byte == '"' or end_byte == '"' then
4382         return "R('\" .. start_byte .. end_byte .. \"')"
4383     else
4384         return "R('\" .. start_byte .. end_byte .. \"')"
4385     end
4386 end

```

The function [serialize\\_replacement](#) produces the Lua code for a string literal with one or more UTF-8-encoded Unicode characters.

```

4387 local function serialize_replacement(codepoints)
4388     assert(#codepoints > 0)
4389     local buffer = {}
4390     for _, codepoint in ipairs(codepoints) do
4391         local code = utf8.char(codepoint)
4392         for i = 1, #code do
4393             local byte = code:sub(i, i)
4394             if byte == '"' then
4395                 table.insert(buffer, '\\\"')
4396             elseif byte == "\\" then
4397                 table.insert(buffer, "\\\\")

```

```

4398     else
4399         table.insert(buffer, byte)
4400     end
4401 end
4402 end
4403 table.insert(buffer, '')
4404 return table.concat(buffer)
4405 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.

```

4406 local function read_decompositions(file)
4407     return function()
4408         local from_codepoint, mapping
4409         for line in file:lines() do
4410             from_codepoint, mapping
4411                 = line:match("^(%x+);[~;]*;%a*;%d+;%a*;( [<%a>%x ]*)")
4412             assert(from_codepoint ~= nil)
4413             if #mapping > 0 then
4414                 break
4415             end
4416         end
4417         if #mapping == 0 then
4418             return nil
4419         end
4420         from_codepoint = tonumber(from_codepoint, 16)
4421         local to_codepoints, decomposition_type = {}, "canonical"
4422         for raw_codepoint in mapping:gmatch("%S+") do
4423             assert(#raw_codepoint > 0)
4424             if raw_codepoint:sub(1, 1) == "<" then
4425                 decomposition_type = "compatibility"
4426             else
4427                 local codepoint = tonumber(raw_codepoint, 16)
4428                 table.insert(to_codepoints, codepoint)
4429             end
4430         end
4431         assert(#to_codepoints > 0)
4432         return decomposition_type, from_codepoint, to_codepoints
4433     end
4434 end

```

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

```

4435 print("local P, R, Cc, C = lpeg.P, lpeg.R, lpeg.Cc, lpeg.C")
4436 print("local fail = P(false)")
4437 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`.

```
4438 ;(function()  
4439   local file = assert(io.open("UnicodeData.txt", "r"),  
4440     [[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.

```
4441   local decomposition_types = {"canonical", "compatibility"}  
4442   local prefix_trees = {}  
4443   for _, decomposition_type in ipairs(decomposition_types) do  
4444     prefix_trees[decomposition_type] = {}  
4445     for char_length = 1, 4 do  
4446       prefix_trees[decomposition_type][char_length]  
4447         = {_type = "intermediate"}  
4448     end  
4449   end  
4450   for decomposition_type, from_codepoint, to_codepoints  
4451     in read_decompositions(file) do  
4452     local from_code = utf8.char(from_codepoint)  
4453     local node = prefix_trees[decomposition_type][#from_code]  
4454     for i = 1, #from_code do  
4455       local from_byte = from_code:sub(i, i)  
4456       if i < #from_code then  
4457         if node[from_byte] == nil then  
4458           node[from_byte] = {_type = "intermediate"}  
4459         end  
4460         node = node[from_byte]  
4461       else  
4462         table.insert(node, {from_byte, to_codepoints, _type = "leaf"})  
4463       end  
4464     end  
4465   end  
4466   assert(file:close())
```

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

```
4467   print("M.decomposition_mapping = {}")  
4468   for _, decomposition_type in ipairs(decomposition_types) do  
4469     print("M.decomposition_mapping." .. decomposition_type .. " = {}")  
4470     for length, prefix_tree in pairs(prefix_trees[decomposition_type])  
4471     do  
4472       local subparsers = {}
```

```

4473     depth_first_search(prefix_tree, "", function(node, path)
4474         if node._type == "leaf" then
4475             local from_byte, to_codepoints = table.unpack(node)
4476             local suffix = serialize_byte_parser(from_byte)
4477             .. " / " .. serialize_replacement(to_codepoints)
4478             if subparsers[path] ~= nil then
4479                 subparsers[path] = subparsers[path] .. " + " .. suffix
4480             else
4481                 subparsers[path] = suffix
4482             end
4483         end
4484     end, function(_, path)
4485         if #path > 0 then
4486             local byte = path:sub(#path, #path)
4487             local parent_path = path:sub(1, #path-1)
4488             local prefix = serialize_byte_parser(byte)
4489             local suffix
4490             if subparsers[path]:find(" %+ ") then
4491                 suffix = prefix .. " * (" .. subparsers[path] .. ")"
4492             else
4493                 suffix = prefix .. " * " .. subparsers[path]
4494             end
4495             if subparsers[parent_path] ~= nil then
4496                 subparsers[parent_path] = subparsers[parent_path]
4497                     .. " + " .. suffix
4498             else
4499                 subparsers[parent_path] = suffix
4500             end
4501         else
4502             print(
4503                 "M.decomposition_mapping." .. decomposition_type
4504                 .. "[" .. length .. "]" = " .. (subparsers[path] or "fail")
4505             )
4506         end
4507     end)
4508 end
4509 end
4510 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](#).

```

4511 ;(function()

```

```

4512 local file = assert(io.open("HangulSyllableType.txt", "r"),
4513     [[Could not open file "HangulSyllableType.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 syllable type and code length.

```

4514 local syllable_types = {"LV", "LVT"}
4515 local prefix_trees = {}
4516 for _, syllable_type in ipairs(syllable_types) do
4517     prefix_trees[syllable_type] = {}
4518     for char_length = 1, 4 do
4519         prefix_trees[syllable_type][char_length]
4520             = {_type = "intermediate"}
4521     end
4522 end
4523 for line in file:lines() do
4524     if #line == 0 or line:sub(1, 1) == "#" then
4525         goto continue
4526     end
4527     local codepoint, syllable_type = line:match("^([%x.]+)%s*;%s*(%a*)")
4528     assert(codepoint ~= nil)
4529     if prefix_trees[syllable_type] == nil then
4530         goto continue
4531     end
4532     local codepoint_start, codepoint_end
4533     if codepoint:find("%.%.") then
4534         codepoint_start, codepoint_end
4535             = codepoint:match("^(%x+)%.%.(%x+)$")
4536     else
4537         codepoint_start, codepoint_end = codepoint, codepoint
4538     end
4539     codepoint_start = tonumber(codepoint_start, 16)
4540     codepoint_end = tonumber(codepoint_end, 16)
4541     local prev_code, prev_leaf_node
4542     for codepoint = codepoint_start, codepoint_end do
4543         local code = utf8.char(codepoint)
4544         local node = prefix_trees[syllable_type][#code]
4545         for i = 1, #code do
4546             local byte = code:sub(i, i)
4547             if i < #code then
4548                 if node[byte] == nil then
4549                     node[byte] = {_type = "intermediate"}
4550                 end
4551                 node = node[byte]
4552             else
4553                 local leaf_node
4554                 if (
4555                     prev_code ~= nil

```

```

4556         and #prev_code == #code
4557         and code:sub(1, #code - 1)
4558         == prev_code:sub(1, #code - 1)
4559     ) then
4560         assert(prev_leaf_node ~= nil)
4561         leaf_node = prev_leaf_node
4562         leaf_node[2] = byte
4563     else
4564         leaf_node = {byte, byte, _type = "leaf"}
4565         table.insert(node, leaf_node)
4566     end
4567     prev_code, prev_leaf_node = code, leaf_node
4568 end
4569 end
4570 end
4571 ::continue::
4572 end
4573 assert(file:close())

```

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

```

4574 print(string.format("local s_base = %d", tonumber("AC00", 16)))
4575 print(string.format("local l_base = %d", tonumber("1100", 16)))
4576 print(string.format("local v_base = %d", tonumber("1161", 16)))
4577 print(string.format("local t_base = %d", tonumber("11A7", 16)))
4578 print(string.format("local l_count = %d", 19))
4579 print(string.format("local v_count = %d", 21))
4580 print(string.format("local t_count = %d", 28))
4581 print("local n_count = v_count * t_count")
4582 print("local s_count = l_count * n_count")
4583
4584 print("local function hangul_decompose_LV(byte)")
4585 print("  local s = utf8.codepoint(byte)")
4586 print("  local s_index = s - s_base")
4587 print("  local l_index = s_index // n_count")
4588 print("  local v_index = (s_index % n_count) // t_count")
4589 print("  local l_part = l_base + l_index")
4590 print("  local v_part = v_base + v_index")
4591 print("  return utf8.char(l_part) .. utf8.char(v_part)")
4592 print("end")
4593
4594 print("local function hangul_decompose_LVT(byte)")
4595 print("  local s = utf8.codepoint(byte)")
4596 print("  local s_index = s - s_base")
4597 print("  local lv_index = (s_index // t_count) * t_count")
4598 print("  local t_index = s_index % t_count")
4599 print("  local lv_part = s_base + lv_index")

```



```

4600 print(" local t_part = t_base + t_index")
4601 print(" return utf8.char(lv_part) .. utf8.char(t_part)")
4602 print("end")
4603
4604 print("function M.hangul_compose(first_byte, second_byte)")
4605 print(" local last = utf8.codepoint(first_byte)")
4606 print(" local ch = utf8.codepoint(second_byte)")
4607 print(" local l_index = last - l_base")
4608 print(" if 0 <= l_index and l_index < l_count then")
4609 print("     local v_index = ch - v_base")
4610 print("     if 0 <= v_index and v_index < v_count then")
4611 print("         return utf8.char("
4612             s_base + (l_index * v_count + v_index) * t_count")
4613         ")")
4614 print("     end")
4615 print(" end")
4616 print(" local s_index = last - s_base")
4617 print(" if 0 <= s_index and s_index < s_count")
4618 print("     and s_index % t_count == 0 then")
4619 print("     local t_index = ch - t_base")
4620 print("     if 0 < t_index and t_index < t_count then")
4621 print("         return utf8.char(last + t_index)")
4622 print("     end")
4623 print(" end")
4624 print(" return nil")
4625 print("end")

```

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

```

4626 print("M.decomposition_mapping.hangul = {}")
4627 for _, syllable_type in ipairs(syllable_types) do
4628     print("M.decomposition_mapping.hangul." .. syllable_type .. " = {}")
4629     for length, prefix_tree in pairs(prefix_trees[syllable_type]) do
4630         local subparsers = {}
4631         depth_first_search(prefix_tree, "", function(node, path)
4632             if node._type == "leaf" then
4633                 local start_byte, end_byte = table.unpack(node)
4634                 local suffix
4635                 if start_byte == end_byte then
4636                     suffix = serialize_byte_parser(start_byte)
4637                 else
4638                     assert(start_byte < end_byte)
4639                     suffix = serialize_byte_range_parser(start_byte, end_byte)
4640                 end
4641                 if subparsers[path] ~= nil then
4642                     subparsers[path] = subparsers[path] .. " + " .. suffix
4643                 else
4644                     subparsers[path] = suffix
4645                 end

```

```

4646         end
4647     end, function(_, path)
4648         if #path > 0 then
4649             local byte = path:sub(#path, #path)
4650             local parent_path = path:sub(1, #path-1)
4651             local prefix = serialize_byte_parser(byte)
4652             local suffix
4653             if subparsers[path]:find(" %+ ") then
4654                 suffix = prefix .. " * (" .. subparsers[path] .. ")"
4655             else
4656                 suffix = prefix .. " * " .. subparsers[path]
4657             end
4658             if subparsers[parent_path] ~= nil then
4659                 subparsers[parent_path] = subparsers[parent_path]
4660   .. " + " .. suffix
4661             else
4662                 subparsers[parent_path] = suffix
4663             end
4664         else
4665             if subparsers[path] then
4666                 print(
4667                     "M.decomposition_mapping.hangul." .. syllable_type
4668                     .. "[" .. length .. "] = C(" .. subparsers[path] .. ")"
4669                     .. " / hangul_decompose_" .. syllable_type
4670                 )
4671             else
4672                 print(
4673                     "M.decomposition_mapping.hangul." .. syllable_type
4674                     .. "[" .. length .. "] = fail"
4675                 )
4676             end
4677         end
4678     end)
4679 end
4680 end
4681 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.

```

4682 ;(function()
4683     local file = assert(io.open("DerivedNormalizationProps.txt", "r"),

```

```

4684     [[Could not open file "DerivedNormalizationProps.txt"]])
4685     local full_composition_exclusions = {}
4686     for line in file:lines() do
4687         if #line == 0 or line:sub(1, 1) == "#" then
4688             goto continue
4689         end
4690         local codepoint, property = line:match("^([%x.]+)%s*;%s*([%a_]+)")
4691         assert(codepoint ~= nil)
4692         if property ~= "Full_Composition_Exclusion" then
4693             goto continue
4694         end
4695         local codepoint_start, codepoint_end
4696         if codepoint:find("%.%.") then
4697             codepoint_start, codepoint_end
4698             = codepoint:match("^(%x+)%.%.(%x+)$")
4699         else
4700             codepoint_start, codepoint_end = codepoint, codepoint
4701         end
4702         codepoint_start = tonumber(codepoint_start, 16)
4703         codepoint_end = tonumber(codepoint_end, 16)
4704         for codepoint = codepoint_start, codepoint_end do
4705             full_composition_exclusions[codepoint] = true
4706         end
4707         ::continue::
4708     end
4709     assert(file:close())

```

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

```

4710     file = assert(io.open("UnicodeData.txt", "r"),
4711         [[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.

```

4712     local prefix_trees = {starters = {}, both = {}}
4713     for first_char_length = 1, 4 do
4714         prefix_trees.starters[first_char_length]
4715             = {_type = "intermediate"}
4716         prefix_trees.both[first_char_length] = {}
4717         for second_char_length = 1, 4 do
4718             prefix_trees.both[first_char_length][second_char_length]
4719                 = {_type = "intermediate"}
4720         end
4721     end
4722     local seen_starter_codes = {}
4723     for decomposition_type, to_codepoint, from_codepoints
4724         in read_decompositions(file) do
4725         if (
4726             decomposition_type ~= "canonical"

```

```

4727         or #from_codepoints ~= 2
4728         or full_composition_exclusions[to_codepoint]
4729     ) then
4730         goto continue
4731     end
4732     local starter_code = utf8.char(from_codepoints[1])
4733     local combining_character_code = utf8.char(from_codepoints[2])
4734     local starter_node = prefix_trees.starters[#starter_code]
4735     local both_node
4736         = prefix_trees.both[#starter_code][#combining_character_code]
4737     for i = 1, #starter_code do
4738         local from_byte = starter_code:sub(i, i)
4739         if both_node[from_byte] == nil then
4740             both_node[from_byte] = {_type = "intermediate"}
4741         end
4742         both_node = both_node[from_byte]
4743         if i < #starter_code then
4744             if starter_node[from_byte] == nil then
4745                 starter_node[from_byte] = {_type = "intermediate"}
4746             end
4747             starter_node = starter_node[from_byte]
4748         elseif seen_starter_codes[starter_code] == nil then
4749             seen_starter_codes[starter_code] = true
4750             table.insert(starter_node, {from_byte, _type = "leaf"})
4751         end
4752     end
4753     for i = 1, #combining_character_code do
4754         local from_byte = combining_character_code:sub(i, i)
4755         if i < #combining_character_code then
4756             if both_node[from_byte] == nil then
4757                 both_node[from_byte] = {_type = "intermediate"}
4758             end
4759             both_node = both_node[from_byte]
4760         else
4761             table.insert(
4762                 both_node,
4763                 {from_byte, to_codepoint, _type = "leaf"}
4764             )
4765         end
4766     end
4767     ::continue::
4768 end
4769 assert(file:close())

```

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

```

4770 print("M.composition_mapping = {starters = {}, both = {}}")
4771 for first_char_length = 1, 4 do
4772     local prefix_tree = prefix_trees.starters[first_char_length]

```

```

4773 local subparsers = {}
4774 depth_first_search(prefix_tree, "", function(node, path)
4775     if node._type == "leaf" then
4776         local from_byte = table.unpack(node)
4777         local suffix = serialize_byte_parser(from_byte)
4778         if subparsers[path] ~= nil then
4779             subparsers[path] = subparsers[path] .. " + " .. suffix
4780         else
4781             subparsers[path] = suffix
4782         end
4783     end
4784 end, function(_, path)
4785     if #path > 0 then
4786         local byte = path:sub(#path, #path)
4787         local parent_path = path:sub(1, #path-1)
4788         local prefix = serialize_byte_parser(byte)
4789         local suffix
4790         if subparsers[path]:find(" %+ ") then
4791             suffix = prefix .. " * (" .. subparsers[path] .. ")"
4792         else
4793             suffix = prefix .. " * " .. subparsers[path]
4794         end
4795         if subparsers[parent_path] ~= nil then
4796             subparsers[parent_path] = subparsers[parent_path]
4797                 .. " + " .. suffix
4798         else
4799             subparsers[parent_path] = suffix
4800         end
4801     else
4802         print(
4803             "M.composition_mapping.starters["
4804             .. first_char_length .. "] = " .. (subparsers[path] or "fail")
4805         )
4806     end
4807 end)
4808 print(
4809     string.format(
4810         "M.composition_mapping.both[%d] = {}",
4811         first_char_length
4812     )
4813 )
4814 for second_char_length = 1, 4 do
4815     prefix_tree
4816     = prefix_trees.both[first_char_length][second_char_length]
4817     subparsers = {}
4818     depth_first_search(prefix_tree, "", function(node, path)
4819         if node._type == "leaf" then

```

```

4820         local from_byte, to_codepoint = table.unpack(node)
4821         local suffix = serialize_byte_parser(from_byte)
4822         .. " / " .. serialize_replacement({to_codepoint})
4823         if subparsers[path] ~= nil then
4824             subparsers[path] = subparsers[path] .. " + " .. suffix
4825         else
4826             subparsers[path] = suffix
4827         end
4828     end
4829 end, function(_, path)
4830     if #path > 0 then
4831         local byte = path:sub(#path, #path)
4832         local parent_path = path:sub(1, #path-1)
4833         local prefix = serialize_byte_parser(byte)
4834         local suffix
4835         if subparsers[path]:find(" %+ ") then
4836             suffix = prefix .. " * (" .. subparsers[path] .. ")"
4837         else
4838             suffix = prefix .. " * " .. subparsers[path]
4839         end
4840         if subparsers[parent_path] ~= nil then
4841             subparsers[parent_path] = subparsers[parent_path]
4842                 .. " + " .. suffix
4843         else
4844             subparsers[parent_path] = suffix
4845         end
4846     else
4847         print(
4848             "M.composition_mapping.both[" .. first_char_length .. "]["
4849                 .. second_char_length .. "] = "
4850                 .. (subparsers[path] or "fail")
4851         )
4852     end
4853 end)
4854 end
4855 end
4856 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`.

```

4857 ;(function()
4858     local file = assert(io.open("CaseFolding.txt", "r"),

```

```
4859      [[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.

```
4860     local prefix_trees = {}
4861     for char_length = 1, 4 do
4862         prefix_trees[char_length] = {_type = "intermediate"}
4863     end
4864     for line in file:lines() do
4865         if #line == 0 or line:sub(1, 1) == "#" then
4866             goto continue
4867         end
4868         local raw_from_codepoint, status, raw_to_codepoints
4869             = line:match("^(%x+); ([CFST]); ([%x ]+);")
4870         assert(raw_from_codepoint ~= nil)
4871         assert(status ~= nil)
4872         assert(raw_to_codepoints ~= nil)
4873         if status ~= "C" and status ~= "F" then
4874             goto continue
4875         end
4876         local from_codepoint = tonumber(raw_from_codepoint, 16)
4877         local to_codepoints = {}
4878         for raw_codepoint in raw_to_codepoints:gmatch('%x+') do
4879             local codepoint = tonumber(raw_codepoint, 16)
4880             table.insert(to_codepoints, codepoint)
4881         end
4882         local from_code = utf8.char(from_codepoint)
4883         local node = prefix_trees[#from_code]
4884         for i = 1, #from_code do
4885             local from_byte = from_code:sub(i, i)
4886             if i < #from_code then
4887                 if node[from_byte] == nil then
4888                     node[from_byte] = {_type = "intermediate"}
4889                 end
4890                 node = node[from_byte]
4891             else
4892                 table.insert(node, {from_byte, to_codepoints, _type = "leaf"})
4893             end
4894         end
4895         ::continue::
4896     end
4897     assert(file:close())
```

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

```
4898     print("M.casefold_mapping = {}")
4899     for length, prefix_tree in pairs(prefix_trees) do
4900         local subparsers = {}
4901         depth_first_search(prefix_tree, "", function(node, path)
```

```

4902     if node._type == "leaf" then
4903         local from_byte, to_codepoints = table.unpack(node)
4904         local suffix = serialize_byte_parser(from_byte)
4905         .. " / " .. serialize_replacement(to_codepoints)
4906         if subparsers[path] ~= nil then
4907             subparsers[path] = subparsers[path] .. " + " .. suffix
4908         else
4909             subparsers[path] = suffix
4910         end
4911     end
4912 end, function(_, path)
4913     if #path > 0 then
4914         local byte = path:sub(#path, #path)
4915         local parent_path = path:sub(1, #path-1)
4916         local prefix = serialize_byte_parser(byte)
4917         local suffix
4918         if subparsers[path]:find(" %+ ") then
4919             suffix = prefix .. " * (" .. subparsers[path] .. ")"
4920         else
4921             suffix = prefix .. " * " .. subparsers[path]
4922         end
4923         if subparsers[parent_path] ~= nil then
4924             subparsers[parent_path] = subparsers[parent_path]
4925                 .. " + " .. suffix
4926         else
4927             subparsers[parent_path] = suffix
4928         end
4929     else
4930         print(
4931             "M.casefold_mapping[" .. length .. "] = "
4932             .. (subparsers[path] or "fail")
4933         )
4934     end
4935 end)
4936 end
4937 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`.

```

4938 ;(function()
4939     local file = assert(io.open("UnicodeData.txt", "r"),
4940         [[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.

```

4941 local categories = {"L", "N", "P", "Pc", "S", "Z"}
4942 local prefix_trees = {}
4943 for _, category in ipairs(categories) do
4944     prefix_trees[category] = {}
4945     for char_length = 1, 4 do
4946         prefix_trees[category][char_length] = {_type = "intermediate"}
4947     end
4948 end
4949 for line in file:lines() do
4950     local codepoint, full_category = line:match("^(%x+);[~;]*;(%a*)")
4951     assert(#full_category >= 1)
4952     local major_category = full_category:sub(1, 1)
4953     for _, category in ipairs({full_category, major_category}) do
4954         if prefix_trees[category] == nil then
4955             goto continue
4956         end
4957         local code = utf8.char(tonumber(codepoint, 16))
4958         local node = prefix_trees[category][#code]
4959         for i = 1, #code do
4960             local byte = code:sub(i, i)
4961             if i < #code then
4962                 if node[byte] == nil then
4963                     node[byte] = {_type = "intermediate"}
4964                 end
4965                 node = node[byte]
4966             else
4967                 table.insert(node, {byte, _type = "leaf"})
4968             end
4969         end
4970         ::continue::
4971     end
4972 end
4973 assert(file:close())

```

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

```

4974 print("M.categories = {}")
4975 for _, category in ipairs(categories) do
4976     print("M.categories." .. category .. " = {}")
4977     for length, prefix_tree in pairs(prefix_trees[category]) do
4978         local subparsers = {}
4979         depth_first_search(prefix_tree, "", function(node, path)
4980             if node._type == "leaf" then
4981                 local byte = node[1]
4982                 local suffix = serialize_byte_parser(byte)

```

```

4983         if subparsers[path] ~= nil then
4984             subparsers[path] = subparsers[path] .. " + " .. suffix
4985         else
4986             subparsers[path] = suffix
4987         end
4988     end
4989 end, function(_, path)
4990     if #path > 0 then
4991         local byte = path:sub(#path, #path)
4992         local parent_path = path:sub(1, #path-1)
4993         local prefix = serialize_byte_parser(byte)
4994         local suffix
4995         if subparsers[path]:find(" %+ ") then
4996             suffix = prefix .. " * (" .. subparsers[path] .. ")"
4997         else
4998             suffix = prefix .. " * " .. subparsers[path]
4999         end
5000         if subparsers[parent_path] ~= nil then
5001             subparsers[parent_path] = subparsers[parent_path]
5002                 .. " + " .. suffix
5003         else
5004             subparsers[parent_path] = suffix
5005         end
5006     else
5007         print(
5008             "M.categories." .. category .. "[" .. length .. "] = "
5009             .. (subparsers[path] or "fail")
5010         )
5011     end
5012 end)
5013 end
5014 end
5015 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`.

```

5016 ;(function()
5017     local file = assert(io.open("UnicodeData.txt", "r"),
5018         [[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.

```

5019 local prefix_trees = {}
5020 for char_length = 1, 4 do
5021     prefix_trees[char_length] = {_type = "intermediate"}
5022 end
5023 for line in file:lines() do
5024     local codepoint, combining_class
5025     = line:match("^(%x+);[~;]*;%a*;%d+")
5026     combining_class = tonumber(combining_class)
5027     if combining_class == 0 then
5028         goto continue
5029     end
5030     local code = utf8.char(tonumber(codepoint, 16))
5031     local node = prefix_trees[#code]
5032     for i = 1, #code do
5033         local byte = code:sub(i, i)
5034         if i < #code then
5035             if node[byte] == nil then
5036                 node[byte] = {_type = "intermediate"}
5037             end
5038             node = node[byte]
5039         else
5040             table.insert(node, {byte, combining_class, _type = "leaf"})
5041         end
5042     end
5043     ::continue::
5044 end
5045 assert(file:close())

```

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

```

5046 print("M.ccc = {}")
5047 for length, prefix_tree in pairs(prefix_trees) do
5048     local subparsers = {}
5049     depth_first_search(prefix_tree, "", function(node, path)
5050         if node._type == "leaf" then
5051             local byte, combining_class = table.unpack(node)
5052             local suffix = serialize_byte_parser(byte)
5053             .. " * Cc(" .. tostring(combining_class) .. ")"
5054             if subparsers[path] ~= nil then
5055                 subparsers[path] = subparsers[path] .. " + " .. suffix
5056             else
5057                 subparsers[path] = suffix
5058             end
5059         end
5060     end, function(_, path)
5061         if #path > 0 then
5062             local byte = path:sub(#path, #path)
5063             local parent_path = path:sub(1, #path-1)
5064             local prefix = serialize_byte_parser(byte)

```

```

5065     local suffix
5066     if subparsers[path]:find(" %+ ") then
5067         suffix = prefix .. " * (" .. subparsers[path] .. ")"
5068     else
5069         suffix = prefix .. " * " .. subparsers[path]
5070     end
5071     if subparsers[parent_path] ~= nil then
5072         subparsers[parent_path] = subparsers[parent_path]
5073             .. " + " .. suffix
5074     else
5075         subparsers[parent_path] = suffix
5076     end
5077 else
5078     print(
5079         "M.ccc[" .. length .. "] = " .. (subparsers[path] or "fail")
5080     )
5081 end
5082 end)
5083 end
5084 end)()
5085 print("-- luacheck: pop")
5086 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.

```

5087 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.

```

5088 function util.err(msg, exit_code)
5089     io.stderr:write("markdown.lua: " .. msg .. "\n")
5090     os.exit(exit_code or 1)
5091 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.

```

5092 function util.cache(dir, string, salt, transform, suffix)
5093     local digest = md5.sumhexa(string .. (salt or ""))
5094     local name = util.pathname(dir, digest .. suffix)

```

```

5095 local file = io.open(name, "r")
5096 local result = nil
5097 if file == nil then -- If no cache entry exists, create a new one.
5098     file = assert(io.open(name, "w"),
5099         [[Could not open file "]] .. name .. [[ for writing]])
5100     result = string
5101     if transform ~= nil then
5102         result = transform(result)
5103     end
5104     assert(file:write(result))
5105     assert(file:close())
5106 end
5107 return name, result
5108 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.

```

5109 function util.cache_verbatim(dir, string)
5110     local name = util.cache(dir, string, nil, nil, ".verbatim")
5111     return name
5112 end

```

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

```

5113 function util.table_copy(t)
5114     local u = { }
5115     for k, v in pairs(t) do u[k] = v end
5116     return setmetatable(u, getmetatable(t))
5117 end

```

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

```

5118 function util.encode_json_string(s)
5119     s = s:gsub([[\\]], [[\\]])
5120     s = s:gsub([[""]], [[\"]])
5121     return [["]] .. s .. [["]]
5122 end

```

The `util.expand_tabs_in_line` expands tabs in string `s`. If `tabstop` is specified, it is used as the tab stop width. Otherwise, the tab stop width of 4 characters is used. The method is a copy of the tab expansion algorithm from Ierusalimsky [18, Chapter 21].

```

5123 function util.expand_tabs_in_line(s, tabstop)
5124     local tab = tabstop or 4
5125     local corr = 0
5126     return (s:gsub(")\t", function(p)
5127         local sp = tab - (p - 1 + corr) % tab
5128         corr = corr - 1 + sp
5129         return string.rep(" ", sp)

```

```

5130         end))
5131 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.

```

5132 function util.walk(t, f)
5133   local typ = type(t)
5134   if typ == "string" then
5135     f(t)
5136   elseif typ == "table" then
5137     local i = 1
5138     local n
5139     n = t[i]
5140     while n do
5141       util.walk(n, f)
5142       i = i + 1
5143       n = t[i]
5144     end
5145   elseif typ == "function" then
5146     local ok, val = pcall(t)
5147     if ok then
5148       util.walk(val, f)
5149     end
5150   else
5151     f(tostring(t))
5152   end
5153 end

```

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

```

5154 function util.flatten(ary)
5155   local new = {}
5156   for _,v in ipairs(ary) do
5157     if type(v) == "table" then
5158       for _,w in ipairs(util.flatten(v)) do
5159         new[#new + 1] = w
5160       end
5161     else
5162       new[#new + 1] = v
5163     end
5164   end
5165   return new
5166 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.

```

5167 function util.rope_to_string(rope)
5168   local buffer = {}
5169   util.walk(rope, function(x) buffer[#buffer + 1] = x end)
5170   return table.concat(buffer)
5171 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.

```

5172 function util.rope_last(rope)
5173   if #rope == 0 then
5174     return nil
5175   else
5176     local l = rope[#rope]
5177     if type(l) == "table" then
5178       return util.rope_last(l)
5179     else
5180       return l
5181     end
5182   end
5183 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  $1 \leq i \leq \#ary$ .

```

5184 function util.intersperse(ary, x)
5185   local new = {}
5186   local l = #ary
5187   for i,v in ipairs(ary) do
5188     local n = #new
5189     new[n + 1] = v
5190     if i ~= l then
5191       new[n + 2] = x
5192     end
5193   end
5194   return new
5195 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  $1 \leq i \leq \#ary$ .

```

5196 function util.map(ary, f)
5197   local new = {}
5198   for i,v in ipairs(ary) do
5199     new[i] = f(v)
5200   end
5201   return new
5202 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.

```
5203 function util.escaper(char_escapes, string_escapes)
```

Build a string of escapable characters.

```
5204 local char_escapes_list = ""
5205 for i,_ in pairs(char_escapes) do
5206     char_escapes_list = char_escapes_list .. i
5207 end
```

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

```
5208 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.

```
5209 if string_escapes then
5210     for k,v in pairs(string_escapes) do
5211         escapable = P(k) / v + escapable
5212     end
5213 end
```

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

```
5214 local escape_string = Cs((escapable + any)^0)
```

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

```
5215 return function(s)
5216     return lpeg.match(escape_string, s)
5217 end
5218 end
```

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

```
5219 function util.pathname(dir, file)
5220     if #dir == 0 then
5221         return file
5222     else
5223         return dir .. "/" .. file
```



```

5224     end
5225 end

```

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

```

5226 function util.salt(options)
5227     local opt_string = {}
5228     for k, _ in pairs(defaultOptions) do
5229         local v = options[k]
5230         if type(v) == "table" then
5231             for _, i in ipairs(v) do
5232                 opt_string[#opt_string+1] = k .. "=" .. tostring(i)
5233             end

```

The `cacheDir` option is disregarded.

```

5234         elseif k ~= "cacheDir" then
5235             opt_string[#opt_string+1] = k .. "=" .. tostring(v)
5236         end
5237     end
5238     table.sort(opt_string)
5239     local salt = table.concat(opt_string, ",")
5240                 .. "," .. metadata.version
5241     return salt
5242 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.

```

5243 util.warning = (function()
5244     local function warning(s)
5245         io.stderr:write("Warning: " .. s .. "\n")
5246     end

5247     for _, message in ipairs(early_warnings) do
5248         warning(message)
5249     end

5250     return warning
5251 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.

```

5252 util.casefold = (function()
5253     local fail, any = P(false), P(1)
5254     local eof = -any

```

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

```

5255     local fold_character = fail

```

```

5256   for n = 1, 4 do
5257       fold_character
5258       = fold_character
5259       + unicode_data.casefold_mapping[n]
5260   end
5261   fold_character
5262   = fold_character
5263   + C(any)

```

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

```

5264   local fold_string = Ct(fold_character^0) * eof
5265   return function(s, form)
5266       local result = table.concat(lpeg.match(fold_string, s))
5267       assert(result ~= nil)

```

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

```

5268       if form == "nfd" or form == "nfkd" then
5269           result = util.normalize(result, form)
5270           result = table.concat(lpeg.match(fold_string, result))
5271           assert(result ~= nil)
5272           result = util.normalize(result, form)
5273       end
5274       return result
5275   end
5276 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.

```

5277 util.canonically_order = (function()
5278   local fail, any = P(false), P(1)
5279   local eof = -any
5280   local cont = R("\128\191")
5281   local utf8_character
5282       = R("\0\127")
5283       + R("\194\223") * cont
5284       + R("\224\239") * cont * cont
5285       + R("\240\244") * cont * cont * cont

```

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

```

5286   local classify_character = fail
5287   for n = 1, 4 do
5288       classify_character
5289       = classify_character
5290       + unicode_data.ccc[n]

```

```

5291     end
5292     classify_character
5293     = classify_character
5294     + utf8_character * Cc(0)

```

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

```

5295     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.

```

5296     return function(s)
5297         local s_len = utf8.len(s)
5298         if s == false or s_len <= 1 then
5299             return s
5300         end

```

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

```

5301         local classes = lpeg.match(classify_string, s)
5302         if classes == nil then
5303             return s
5304         end
5305         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.

```

5306         local non_starter_ranges = {}
5307         local first_non_starter, last_non_starter = nil, nil
5308         for i = 1, #classes do
5309             if first_non_starter == nil then
5310                 if classes[i] ~= 0 then
5311                     first_non_starter, last_non_starter = i, i
5312                 end
5313             else
5314                 if classes[i] == 0 then
5315                     table.insert(
5316                         non_starter_ranges,
5317                         {first_non_starter, last_non_starter}
5318                     )
5319                     first_non_starter, last_non_starter = nil, nil
5320                 else
5321                     last_non_starter = i
5322                 end
5323             end
5324         end
5325         if first_non_starter ~= nil then

```

```

5326     table.insert(
5327         non_starter_ranges,
5328         {first_non_starter, last_non_starter}
5329     )
5330 end
5331 if #non_starter_ranges == 0 then
5332     return s
5333 end
5334 local max_range_length = 0
5335 for _, range in ipairs(non_starter_ranges) do
5336     local range_start, range_end = table.unpack(range)
5337     local range_length = range_end - range_start + 1
5338     if range_length > max_range_length then
5339         max_range_length = range_length
5340     end
5341 end
5342 if max_range_length <= 1 then
5343     return s
5344 end

```

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

```

5345     local buffer = {}
5346     for _, code in utf8.codes(s) do
5347         local char = utf8.char(code)
5348         table.insert(buffer, char)
5349     end
5350     assert(#buffer == s_len)

```

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

```

5351     for _, range in ipairs(non_starter_ranges) do
5352         local range_start, range_end = table.unpack(range)
5353         local range_length = range_end - range_start + 1
5354         for _ = 1, range_length - 1 do
5355             local swapped = false
5356             for i = range_start, range_end - 1 do
5357                 local j = i + 1
5358                 if classes[i] > classes[j] then
5359                     classes[i], classes[j] = classes[j], classes[i]
5360                     buffer[i], buffer[j] = buffer[j], buffer[i]
5361                     swapped = true
5362                 end
5363             end
5364             if not swapped then
5365                 break
5366             end
5367         end
5368     end

```

Finally, concatenate the buffer and return an ordered string.

```
5369     return table.concat(buffer, "")
5370 end
5371 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.

```
5372 util.decompose = (function()
5373   local fail, any = P(false), P(1)
5374   local eof = -any
5375   local decomposition_types = {"canonical", "compatibility"}
```

First, define parsers that will decompose a character.

```
5376   local decompose_character = {}
5377   for _, decomposition_type in ipairs(decomposition_types) do
5378     decompose_character[decomposition_type] = fail
5379     for n = 1, 4 do
5380       decompose_character[decomposition_type]
5381         = decompose_character[decomposition_type]
5382           + unicode_data.decomposition_mapping[decomposition_type][n]
5383     end
5384     decompose_character[decomposition_type]
5385       = decompose_character[decomposition_type]
5386         + C(any)
5387   end
5388   local hangul = unicode_data.decomposition_mapping.hangul
5389   decompose_character.hangul = {}
5390   for syllable_type, _ in pairs(hangul) do
5391     decompose_character.hangul[syllable_type] = fail
5392     for n = 1, 4 do
5393       decompose_character.hangul[syllable_type]
5394         = decompose_character.hangul[syllable_type]
5395           + hangul[syllable_type][n]
5396     end
5397     decompose_character.hangul[syllable_type]
5398       = decompose_character.hangul[syllable_type]
5399         + C(any)
5400   end
```

Next, define a parser that will decompose a string.

```
5401   local decompose_string = {}
5402   for _, decomposition_type in ipairs(decomposition_types) do
5403     decompose_string[decomposition_type]
5404       = Ct(decompose_character[decomposition_type]^0) * eof
5405   end
```

```

5406 decompose_string.hangul = {}
5407 for syllable_type, _ in pairs(hangul) do
5408     decompose_string.hangul[syllable_type]
5409     = Ct(decompose_character.hangul[syllable_type]^0) * eof
5410 end
5411 local function _decompose(s, parser)
5412     assert(s ~= nil)
5413     local result = table.concat(lpeg.match(parser, s), "")
5414     assert(result ~= nil)
5415     return result
5416 end
5417 return function(s, decomposition_type)
5418     assert(
5419         decomposition_type == "canonical"
5420         or decomposition_type == "compatibility"
5421     )
5422     local prev_s
5423     local next_s = s
5424     repeat
5425         prev_s = next_s
5426         local function decompose(...) next_s = _decompose(next_s, ...) end
5427         decompose(decompose_string.canonical)
5428         if decomposition_type == "compatibility" then
5429             decompose(decompose_string.compatibility)
5430         end
5431         decompose(decompose_string.hangul.LVT)
5432         decompose(decompose_string.hangul.LV)
5433     until prev_s == next_s
5434     return util.canonically_order(next_s)
5435 end
5436 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. Unlike the low-level parsers, this high-level function is invariant to the number of bytes the Unicode characters occupy after conversion to UTF-8.

```

5437 util.compose = (function()
5438     local fail, any = P(false), P(1)
5439     local eof = -any
5440     local cont = R("\128\191")
5441     local utf8_character
5442         = R("\0\127")
5443         + R("\194\223") * cont
5444         + R("\224\239") * cont * cont
5445         + R("\240\244") * cont * cont * cont

```

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

```
5446 local classify_character = fail
5447 for n = 1, 4 do
5448     classify_character
5449     = classify_character
5450     + unicode_data.ccc[n]
5451 end
5452 classify_character
5453 = classify_character
5454 + utf8_character * Cc(0)
```

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

```
5455 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.

```
5456 local compose_characters = fail
5457 for m = 1, 4 do
5458     local starter = #unicode_data.composition_mapping.starters[m]
5459     local both = fail
5460     for n = 1, 4 do
5461         both = (
5462             both
5463             + #unicode_data.composition_mapping.both[m] [n]
5464             * unicode_data.composition_mapping.both[m] [n]
5465         )
5466     end
5467     compose_characters = compose_characters + starter * both
5468 end
```

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

```
5469 return function(s)
5470     local s_len = utf8.len(s)
5471     if s == false or s_len <= 1 then
5472         return s
5473     end
```

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

```
5474     local classes = lpeg.match(classify_string, s)
5475     if classes == nil then
5476         return s
5477     end
5478     assert(#classes == s_len)
```

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

```

5479     local buffer = {}
5480     for _, code in utf8.codes(s) do
5481         local char = utf8.char(code)
5482         table.insert(buffer, char)
5483     end
5484     assert(#buffer == s_len)

```

Finally, implement the composition algorithm.

First, find the first starter character in the string.

```

5485     local starter = 1
5486     while starter <= s_len and classes[starter] ~= 0 do
5487         starter = starter + 1
5488     end
5489     local candidate_combining_mark = starter + 1

```

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

```

5490     while candidate_combining_mark <= s_len do
5491         local L = buffer[starter]
5492         local C = buffer[candidate_combining_mark]
5493         local P = lpeg.match(compose_characters, L .. C)
5494         if P ~= nil then
5495             buffer[starter] = P
5496             buffer[candidate_combining_mark] = ""
5497         else
5498             if classes[candidate_combining_mark] == 0 then
5499                 starter = candidate_combining_mark
5500             end
5501             candidate_combining_mark = candidate_combining_mark + 1
5502         end
5503         assert(starter <= s_len)
5504     end

```

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

```

5505     for i = 1, s_len - 1 do
5506         local last, ch = buffer[i], buffer[i + 1]
5507         if last ~= "" and ch ~= "" then
5508             local composite = unicode_data.hangul_compose(last, ch)
5509             if composite ~= nil then
5510                 buffer[i] = ""
5511                 buffer[i + 1] = composite
5512             end
5513         end
5514     end

```

Finally, concatenate the buffer and return an ordered string.

```

5515     return table.concat(buffer, "")
5516 end
5517 end()

```



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

```
5518 function util.normalize(s, form)
5519   if form == "nfd" then
5520     return util.decompose(s, "canonical")
5521   elseif form == "nfkd" then
5522     return util.decompose(s, "compatibility")
5523   elseif form == "nfc" then
5524     return util.compose(util.decompose(s, "canonical"))
5525   elseif form == "nfkc" then
5526     return util.compose(util.decompose(s, "compatibility"))
5527   else
5528     error(string.format('Unexpected normal form "%s"', form))
5529   end
5530 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.

```
5531 function util.find_file(filename)
5532   if resolvers ~= nil then
5533     return resolvers.findfile(filename)
5534   else
5535     return kpse.find_file(filename)
5536   end
5537 end
5538 function util.find_files(filename)
5539   if resolvers ~= nil then
5540     return resolvers.findfiles(filename)
5541   else
5542     return {kpse.lookup(filename, {all=true})}
5543   end
5544 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.

```
5545 local entities = {}
5546
5547 local character_entities = {
5548   ["Tab"] = 9,
5549   ["NewLine"] = 10,
5550   ["excl"] = 33,
5551   ["QUOT"] = 34,
```

```

5552 ["quot"] = 34,
5553 ["num"] = 35,
5554 ["dollar"] = 36,
5555 ["percent"] = 37,
5556 ["AMP"] = 38,
5557 ["amp"] = 38,
5558 ["apos"] = 39,
5559 ["lpar"] = 40,
5560 ["rpar"] = 41,
5561 ["ast"] = 42,
5562 ["midast"] = 42,
5563 ["plus"] = 43,
5564 ["comma"] = 44,
5565 ["period"] = 46,
5566 ["sol"] = 47,
5567 ["colon"] = 58,
5568 ["semi"] = 59,
5569 ["LT"] = 60,
5570 ["lt"] = 60,
5571 ["nvlt"] = {60, 8402},
5572 ["bne"] = {61, 8421},
5573 ["equals"] = 61,
5574 ["GT"] = 62,
5575 ["gt"] = 62,
5576 ["nvgt"] = {62, 8402},
5577 ["quest"] = 63,
5578 ["commat"] = 64,
5579 ["lbrack"] = 91,
5580 ["lsqb"] = 91,
5581 ["bsol"] = 92,
5582 ["rbrack"] = 93,
5583 ["rsqb"] = 93,
5584 ["Hat"] = 94,
5585 ["UnderBar"] = 95,
5586 ["lowbar"] = 95,
5587 ["DiacriticalGrave"] = 96,
5588 ["grave"] = 96,
5589 ["fjlig"] = {102, 106},
5590 ["lbrace"] = 123,
5591 ["lcub"] = 123,
5592 ["VerticalLine"] = 124,
5593 ["verbar"] = 124,
5594 ["vert"] = 124,
5595 ["rbrace"] = 125,
5596 ["rcub"] = 125,
5597 ["NonBreakingSpace"] = 160,
5598 ["nbsp"] = 160,

```

```

5599 ["iexcl"] = 161,
5600 ["cent"] = 162,
5601 ["pound"] = 163,
5602 ["curren"] = 164,
5603 ["yen"] = 165,
5604 ["brvbar"] = 166,
5605 ["sect"] = 167,
5606 ["Dot"] = 168,
5607 ["DoubleDot"] = 168,
5608 ["die"] = 168,
5609 ["uml"] = 168,
5610 ["COPY"] = 169,
5611 ["copy"] = 169,
5612 ["ordf"] = 170,
5613 ["laquo"] = 171,
5614 ["not"] = 172,
5615 ["shy"] = 173,
5616 ["REG"] = 174,
5617 ["circledR"] = 174,
5618 ["reg"] = 174,
5619 ["macr"] = 175,
5620 ["strns"] = 175,
5621 ["deg"] = 176,
5622 ["PlusMinus"] = 177,
5623 ["plusmn"] = 177,
5624 ["pm"] = 177,
5625 ["sup2"] = 178,
5626 ["sup3"] = 179,
5627 ["DiacriticalAcute"] = 180,
5628 ["acute"] = 180,
5629 ["micro"] = 181,
5630 ["para"] = 182,
5631 ["CenterDot"] = 183,
5632 ["centerdot"] = 183,
5633 ["middot"] = 183,
5634 ["Cedilla"] = 184,
5635 ["cedil"] = 184,
5636 ["sup1"] = 185,
5637 ["ordm"] = 186,
5638 ["raquo"] = 187,
5639 ["frac14"] = 188,
5640 ["frac12"] = 189,
5641 ["half"] = 189,
5642 ["frac34"] = 190,
5643 ["iquest"] = 191,
5644 ["Agrave"] = 192,
5645 ["Aacute"] = 193,

```

5646 ["Acirc"] = 194,  
 5647 ["Atilde"] = 195,  
 5648 ["Auml"] = 196,  
 5649 ["Aring"] = 197,  
 5650 ["angst"] = 197,  
 5651 ["AElig"] = 198,  
 5652 ["Ccedil"] = 199,  
 5653 ["Egrave"] = 200,  
 5654 ["Eacute"] = 201,  
 5655 ["Ecirc"] = 202,  
 5656 ["Euml"] = 203,  
 5657 ["Igrave"] = 204,  
 5658 ["Iacute"] = 205,  
 5659 ["Icirc"] = 206,  
 5660 ["Iuml"] = 207,  
 5661 ["ETH"] = 208,  
 5662 ["Ntilde"] = 209,  
 5663 ["Ograve"] = 210,  
 5664 ["Oacute"] = 211,  
 5665 ["Ocirc"] = 212,  
 5666 ["Otilde"] = 213,  
 5667 ["Ouml"] = 214,  
 5668 ["times"] = 215,  
 5669 ["Oslash"] = 216,  
 5670 ["Ugrave"] = 217,  
 5671 ["Uacute"] = 218,  
 5672 ["Ucirc"] = 219,  
 5673 ["Uuml"] = 220,  
 5674 ["Yacute"] = 221,  
 5675 ["THORN"] = 222,  
 5676 ["szlig"] = 223,  
 5677 ["agrave"] = 224,  
 5678 ["aacute"] = 225,  
 5679 ["acirc"] = 226,  
 5680 ["atilde"] = 227,  
 5681 ["auml"] = 228,  
 5682 ["aring"] = 229,  
 5683 ["aelig"] = 230,  
 5684 ["ccedil"] = 231,  
 5685 ["egrave"] = 232,  
 5686 ["eacute"] = 233,  
 5687 ["ecirc"] = 234,  
 5688 ["euml"] = 235,  
 5689 ["igrave"] = 236,  
 5690 ["iacute"] = 237,  
 5691 ["icirc"] = 238,  
 5692 ["iuml"] = 239,

5693 ["eth"] = 240,  
 5694 ["ntilde"] = 241,  
 5695 ["ograve"] = 242,  
 5696 ["oacute"] = 243,  
 5697 ["ocirc"] = 244,  
 5698 ["otilde"] = 245,  
 5699 ["ouml"] = 246,  
 5700 ["div"] = 247,  
 5701 ["divide"] = 247,  
 5702 ["oslash"] = 248,  
 5703 ["ugrave"] = 249,  
 5704 ["uacute"] = 250,  
 5705 ["ucirc"] = 251,  
 5706 ["uuml"] = 252,  
 5707 ["yacute"] = 253,  
 5708 ["thorn"] = 254,  
 5709 ["yuml"] = 255,  
 5710 ["Amacr"] = 256,  
 5711 ["amacr"] = 257,  
 5712 ["Abreve"] = 258,  
 5713 ["abreve"] = 259,  
 5714 ["Aogon"] = 260,  
 5715 ["aogon"] = 261,  
 5716 ["Cacute"] = 262,  
 5717 ["cacute"] = 263,  
 5718 ["Ccirc"] = 264,  
 5719 ["ccirc"] = 265,  
 5720 ["Cdot"] = 266,  
 5721 ["cdot"] = 267,  
 5722 ["Ccaron"] = 268,  
 5723 ["ccaron"] = 269,  
 5724 ["Dcaron"] = 270,  
 5725 ["dcaron"] = 271,  
 5726 ["Dstrok"] = 272,  
 5727 ["dstrok"] = 273,  
 5728 ["Emacr"] = 274,  
 5729 ["emacr"] = 275,  
 5730 ["Edot"] = 278,  
 5731 ["edot"] = 279,  
 5732 ["Eogon"] = 280,  
 5733 ["eogon"] = 281,  
 5734 ["Ecaron"] = 282,  
 5735 ["ecaron"] = 283,  
 5736 ["Gcirc"] = 284,  
 5737 ["gcirc"] = 285,  
 5738 ["Gbreve"] = 286,  
 5739 ["gbreve"] = 287,

```

5740 ["Gdot"] = 288,
5741 ["gdot"] = 289,
5742 ["Gcedil"] = 290,
5743 ["Hcirc"] = 292,
5744 ["hcirc"] = 293,
5745 ["Hstrokr"] = 294,
5746 ["hstrokr"] = 295,
5747 ["Itilde"] = 296,
5748 ["itilde"] = 297,
5749 ["Imacr"] = 298,
5750 ["imacr"] = 299,
5751 ["Iogon"] = 302,
5752 ["iogon"] = 303,
5753 ["Idot"] = 304,
5754 ["imath"] = 305,
5755 ["inodot"] = 305,
5756 ["IJlig"] = 306,
5757 ["ijlig"] = 307,
5758 ["Jcirc"] = 308,
5759 ["jcirc"] = 309,
5760 ["Kcedil"] = 310,
5761 ["kcedil"] = 311,
5762 ["kgreen"] = 312,
5763 ["Lacute"] = 313,
5764 ["lacute"] = 314,
5765 ["Lcedil"] = 315,
5766 ["lcedil"] = 316,
5767 ["Lcaron"] = 317,
5768 ["lcaron"] = 318,
5769 ["Lmidot"] = 319,
5770 ["lmidot"] = 320,
5771 ["Lstrokr"] = 321,
5772 ["lstrokr"] = 322,
5773 ["Nacute"] = 323,
5774 ["nacute"] = 324,
5775 ["Ncedil"] = 325,
5776 ["ncedil"] = 326,
5777 ["Ncaron"] = 327,
5778 ["ncaron"] = 328,
5779 ["napos"] = 329,
5780 ["ENG"] = 330,
5781 ["eng"] = 331,
5782 ["Omacr"] = 332,
5783 ["omacr"] = 333,
5784 ["Odblac"] = 336,
5785 ["odblac"] = 337,
5786 ["OElig"] = 338,

```

5787 ["oelig"] = 339,  
 5788 ["Racute"] = 340,  
 5789 ["racute"] = 341,  
 5790 ["Rcedil"] = 342,  
 5791 ["rcedil"] = 343,  
 5792 ["Rcaron"] = 344,  
 5793 ["rcaron"] = 345,  
 5794 ["Sacute"] = 346,  
 5795 ["sacute"] = 347,  
 5796 ["Scirc"] = 348,  
 5797 ["scirc"] = 349,  
 5798 ["Scedil"] = 350,  
 5799 ["scedil"] = 351,  
 5800 ["Scaron"] = 352,  
 5801 ["scaron"] = 353,  
 5802 ["Tcedil"] = 354,  
 5803 ["tcedil"] = 355,  
 5804 ["Tcaron"] = 356,  
 5805 ["tcaron"] = 357,  
 5806 ["Tstrok"] = 358,  
 5807 ["tstrok"] = 359,  
 5808 ["Utilde"] = 360,  
 5809 ["utilde"] = 361,  
 5810 ["Umacr"] = 362,  
 5811 ["umacr"] = 363,  
 5812 ["Ubreve"] = 364,  
 5813 ["ubreve"] = 365,  
 5814 ["Uring"] = 366,  
 5815 ["uring"] = 367,  
 5816 ["Udblac"] = 368,  
 5817 ["udblac"] = 369,  
 5818 ["Uogon"] = 370,  
 5819 ["uogon"] = 371,  
 5820 ["Wcirc"] = 372,  
 5821 ["wcirc"] = 373,  
 5822 ["Ycirc"] = 374,  
 5823 ["ycirc"] = 375,  
 5824 ["Yuml"] = 376,  
 5825 ["Zacute"] = 377,  
 5826 ["zacute"] = 378,  
 5827 ["Zdot"] = 379,  
 5828 ["zdot"] = 380,  
 5829 ["Zcaron"] = 381,  
 5830 ["zcaron"] = 382,  
 5831 ["fnof"] = 402,  
 5832 ["imped"] = 437,  
 5833 ["gacute"] = 501,

```

5834 ["jmath"] = 567,
5835 ["circ"] = 710,
5836 ["Hacek"] = 711,
5837 ["caron"] = 711,
5838 ["Breve"] = 728,
5839 ["breve"] = 728,
5840 ["DiacriticalDot"] = 729,
5841 ["dot"] = 729,
5842 ["ring"] = 730,
5843 ["ogon"] = 731,
5844 ["DiacriticalTilde"] = 732,
5845 ["tilde"] = 732,
5846 ["DiacriticalDoubleAcute"] = 733,
5847 ["dblac"] = 733,
5848 ["DownBreve"] = 785,
5849 ["Alpha"] = 913,
5850 ["Beta"] = 914,
5851 ["Gamma"] = 915,
5852 ["Delta"] = 916,
5853 ["Epsilon"] = 917,
5854 ["Zeta"] = 918,
5855 ["Eta"] = 919,
5856 ["Theta"] = 920,
5857 ["Iota"] = 921,
5858 ["Kappa"] = 922,
5859 ["Lambda"] = 923,
5860 ["Mu"] = 924,
5861 ["Nu"] = 925,
5862 ["Xi"] = 926,
5863 ["Omicron"] = 927,
5864 ["Pi"] = 928,
5865 ["Rho"] = 929,
5866 ["Sigma"] = 931,
5867 ["Tau"] = 932,
5868 ["Upsilon"] = 933,
5869 ["Phi"] = 934,
5870 ["Chi"] = 935,
5871 ["Psi"] = 936,
5872 ["Omega"] = 937,
5873 ["ohm"] = 937,
5874 ["alpha"] = 945,
5875 ["beta"] = 946,
5876 ["gamma"] = 947,
5877 ["delta"] = 948,
5878 ["epsi"] = 949,
5879 ["epsilon"] = 949,
5880 ["zeta"] = 950,

```



```

5881 ["eta"] = 951,
5882 ["theta"] = 952,
5883 ["iota"] = 953,
5884 ["kappa"] = 954,
5885 ["lambda"] = 955,
5886 ["mu"] = 956,
5887 ["nu"] = 957,
5888 ["xi"] = 958,
5889 ["omicron"] = 959,
5890 ["pi"] = 960,
5891 ["rho"] = 961,
5892 ["sigmaf"] = 962,
5893 ["sigmav"] = 962,
5894 ["varsigma"] = 962,
5895 ["sigma"] = 963,
5896 ["tau"] = 964,
5897 ["upsilon"] = 965,
5898 ["upsilon"] = 965,
5899 ["phi"] = 966,
5900 ["chi"] = 967,
5901 ["psi"] = 968,
5902 ["omega"] = 969,
5903 ["thetasym"] = 977,
5904 ["thetav"] = 977,
5905 ["vartheta"] = 977,
5906 ["Upsilon"] = 978,
5907 ["upsih"] = 978,
5908 ["phiv"] = 981,
5909 ["straightphi"] = 981,
5910 ["varphi"] = 981,
5911 ["piv"] = 982,
5912 ["varpi"] = 982,
5913 ["Gammad"] = 988,
5914 ["digamma"] = 989,
5915 ["gammad"] = 989,
5916 ["kappav"] = 1008,
5917 ["varkappa"] = 1008,
5918 ["rhov"] = 1009,
5919 ["varrho"] = 1009,
5920 ["epsiv"] = 1013,
5921 ["straightepsilon"] = 1013,
5922 ["varepsilon"] = 1013,
5923 ["backepsilon"] = 1014,
5924 ["bepsi"] = 1014,
5925 ["IOcy"] = 1025,
5926 ["DJcy"] = 1026,
5927 ["GJcy"] = 1027,

```

```

5928 ["Jukcy"] = 1028,
5929 ["DScy"] = 1029,
5930 ["Iukcy"] = 1030,
5931 ["YIcy"] = 1031,
5932 ["Jsercy"] = 1032,
5933 ["LJcy"] = 1033,
5934 ["NJcy"] = 1034,
5935 ["TSHcy"] = 1035,
5936 ["KJcy"] = 1036,
5937 ["Ubrcy"] = 1038,
5938 ["DZcy"] = 1039,
5939 ["Acy"] = 1040,
5940 ["Bcy"] = 1041,
5941 ["Vcy"] = 1042,
5942 ["Gcy"] = 1043,
5943 ["Dcy"] = 1044,
5944 ["IEcy"] = 1045,
5945 ["ZHcy"] = 1046,
5946 ["Zcy"] = 1047,
5947 ["Icy"] = 1048,
5948 ["Jcy"] = 1049,
5949 ["Kcy"] = 1050,
5950 ["Lcy"] = 1051,
5951 ["Mcy"] = 1052,
5952 ["Ncy"] = 1053,
5953 ["Ocy"] = 1054,
5954 ["Pcy"] = 1055,
5955 ["Rcy"] = 1056,
5956 ["Scy"] = 1057,
5957 ["Tcy"] = 1058,
5958 ["Ucy"] = 1059,
5959 ["Fcy"] = 1060,
5960 ["KHcy"] = 1061,
5961 ["TScy"] = 1062,
5962 ["CHcy"] = 1063,
5963 ["SHcy"] = 1064,
5964 ["SHCHcy"] = 1065,
5965 ["HARDcy"] = 1066,
5966 ["Ycy"] = 1067,
5967 ["SOFTcy"] = 1068,
5968 ["Ecy"] = 1069,
5969 ["YUcy"] = 1070,
5970 ["YAcy"] = 1071,
5971 ["acy"] = 1072,
5972 ["bcy"] = 1073,
5973 ["vcy"] = 1074,
5974 ["gcy"] = 1075,

```

```

5975 ["dcy"] = 1076,
5976 ["iecy"] = 1077,
5977 ["zhcy"] = 1078,
5978 ["zcy"] = 1079,
5979 ["icy"] = 1080,
5980 ["jcy"] = 1081,
5981 ["kcy"] = 1082,
5982 ["lcy"] = 1083,
5983 ["mcy"] = 1084,
5984 ["ncy"] = 1085,
5985 ["ocy"] = 1086,
5986 ["pcy"] = 1087,
5987 ["rcy"] = 1088,
5988 ["scy"] = 1089,
5989 ["tcy"] = 1090,
5990 ["ucy"] = 1091,
5991 ["fcy"] = 1092,
5992 ["khcy"] = 1093,
5993 ["tscy"] = 1094,
5994 ["chcy"] = 1095,
5995 ["shcy"] = 1096,
5996 ["shchcy"] = 1097,
5997 ["hardcy"] = 1098,
5998 ["ycy"] = 1099,
5999 ["softcy"] = 1100,
6000 ["ecy"] = 1101,
6001 ["yucy"] = 1102,
6002 ["yacy"] = 1103,
6003 ["iocy"] = 1105,
6004 ["djcy"] = 1106,
6005 ["gjcy"] = 1107,
6006 ["jukcy"] = 1108,
6007 ["dscy"] = 1109,
6008 ["iukcy"] = 1110,
6009 ["yicy"] = 1111,
6010 ["jsercy"] = 1112,
6011 ["ljcy"] = 1113,
6012 ["njcy"] = 1114,
6013 ["tshcy"] = 1115,
6014 ["kjcy"] = 1116,
6015 ["ubrcy"] = 1118,
6016 ["dzcy"] = 1119,
6017 ["ensp"] = 8194,
6018 ["emsp"] = 8195,
6019 ["emsp13"] = 8196,
6020 ["emsp14"] = 8197,
6021 ["numsp"] = 8199,

```

```

6022 ["puncsp"] = 8200,
6023 ["ThinSpace"] = 8201,
6024 ["thinsp"] = 8201,
6025 ["VeryThinSpace"] = 8202,
6026 ["hairsp"] = 8202,
6027 ["NegativeMediumSpace"] = 8203,
6028 ["NegativeThickSpace"] = 8203,
6029 ["NegativeThinSpace"] = 8203,
6030 ["NegativeVeryThinSpace"] = 8203,
6031 ["ZeroWidthSpace"] = 8203,
6032 ["zwnj"] = 8204,
6033 ["zwj"] = 8205,
6034 ["lrm"] = 8206,
6035 ["rlm"] = 8207,
6036 ["dash"] = 8208,
6037 ["hyphen"] = 8208,
6038 ["ndash"] = 8211,
6039 ["mdash"] = 8212,
6040 ["horbar"] = 8213,
6041 ["Verbar"] = 8214,
6042 ["Vert"] = 8214,
6043 ["OpenCurlyQuote"] = 8216,
6044 ["lsquo"] = 8216,
6045 ["CloseCurlyQuote"] = 8217,
6046 ["rsquo"] = 8217,
6047 ["rsquor"] = 8217,
6048 ["lsquor"] = 8218,
6049 ["sbquo"] = 8218,
6050 ["OpenCurlyDoubleQuote"] = 8220,
6051 ["ldquo"] = 8220,
6052 ["CloseCurlyDoubleQuote"] = 8221,
6053 ["rdquo"] = 8221,
6054 ["rdquor"] = 8221,
6055 ["bdquo"] = 8222,
6056 ["ldquor"] = 8222,
6057 ["dagger"] = 8224,
6058 ["Dagger"] = 8225,
6059 ["ddagger"] = 8225,
6060 ["bull"] = 8226,
6061 ["bullet"] = 8226,
6062 ["nldr"] = 8229,
6063 ["hellip"] = 8230,
6064 ["mldr"] = 8230,
6065 ["permil"] = 8240,
6066 ["pertenk"] = 8241,
6067 ["prime"] = 8242,
6068 ["Prime"] = 8243,

```

```

6069 ["tprime"] = 8244,
6070 ["backprime"] = 8245,
6071 ["bprime"] = 8245,
6072 ["lsaquo"] = 8249,
6073 ["rsaquo"] = 8250,
6074 ["OverBar"] = 8254,
6075 ["oline"] = 8254,
6076 ["caret"] = 8257,
6077 ["hybull"] = 8259,
6078 ["frasl"] = 8260,
6079 ["bsemi"] = 8271,
6080 ["qprime"] = 8279,
6081 ["MediumSpace"] = 8287,
6082 ["ThickSpace"] = {8287, 8202},
6083 ["NoBreak"] = 8288,
6084 ["ApplyFunction"] = 8289,
6085 ["af"] = 8289,
6086 ["InvisibleTimes"] = 8290,
6087 ["it"] = 8290,
6088 ["InvisibleComma"] = 8291,
6089 ["ic"] = 8291,
6090 ["euro"] = 8364,
6091 ["TripleDot"] = 8411,
6092 ["tdot"] = 8411,
6093 ["DotDot"] = 8412,
6094 ["Copf"] = 8450,
6095 ["complexes"] = 8450,
6096 ["incare"] = 8453,
6097 ["gscr"] = 8458,
6098 ["HilbertSpace"] = 8459,
6099 ["Hscr"] = 8459,
6100 ["hamilt"] = 8459,
6101 ["Hfr"] = 8460,
6102 ["Poincareplane"] = 8460,
6103 ["Hopf"] = 8461,
6104 ["quaternions"] = 8461,
6105 ["planckh"] = 8462,
6106 ["hbar"] = 8463,
6107 ["hslash"] = 8463,
6108 ["planck"] = 8463,
6109 ["plankv"] = 8463,
6110 ["Iscr"] = 8464,
6111 ["imagline"] = 8464,
6112 ["Ifr"] = 8465,
6113 ["Im"] = 8465,
6114 ["image"] = 8465,
6115 ["imagpart"] = 8465,

```

```

6116 ["Laplacetrif"] = 8466,
6117 ["Lscr"] = 8466,
6118 ["lagran"] = 8466,
6119 ["ell"] = 8467,
6120 ["Nopf"] = 8469,
6121 ["naturals"] = 8469,
6122 ["numero"] = 8470,
6123 ["copysr"] = 8471,
6124 ["weierp"] = 8472,
6125 ["wp"] = 8472,
6126 ["Popf"] = 8473,
6127 ["primes"] = 8473,
6128 ["Qopf"] = 8474,
6129 ["rationals"] = 8474,
6130 ["Rscr"] = 8475,
6131 ["realine"] = 8475,
6132 ["Re"] = 8476,
6133 ["Rfr"] = 8476,
6134 ["real"] = 8476,
6135 ["realpart"] = 8476,
6136 ["Ropf"] = 8477,
6137 ["reals"] = 8477,
6138 ["rx"] = 8478,
6139 ["TRADE"] = 8482,
6140 ["trade"] = 8482,
6141 ["Zopf"] = 8484,
6142 ["integers"] = 8484,
6143 ["mho"] = 8487,
6144 ["Zfr"] = 8488,
6145 ["zeetrif"] = 8488,
6146 ["iiota"] = 8489,
6147 ["Bernoullis"] = 8492,
6148 ["Bscr"] = 8492,
6149 ["bernou"] = 8492,
6150 ["Cayleys"] = 8493,
6151 ["Cfr"] = 8493,
6152 ["escr"] = 8495,
6153 ["Escr"] = 8496,
6154 ["expectation"] = 8496,
6155 ["Fouriertrif"] = 8497,
6156 ["Fscr"] = 8497,
6157 ["Mellintrif"] = 8499,
6158 ["Mscr"] = 8499,
6159 ["phmmat"] = 8499,
6160 ["order"] = 8500,
6161 ["orderof"] = 8500,
6162 ["oscr"] = 8500,

```

```

6163 ["alefsym"] = 8501,
6164 ["aleph"] = 8501,
6165 ["beth"] = 8502,
6166 ["gimel"] = 8503,
6167 ["daleth"] = 8504,
6168 ["CapitalDifferentialD"] = 8517,
6169 ["DD"] = 8517,
6170 ["DifferentialD"] = 8518,
6171 ["dd"] = 8518,
6172 ["ExponentialE"] = 8519,
6173 ["ee"] = 8519,
6174 ["exponentiale"] = 8519,
6175 ["ImaginaryI"] = 8520,
6176 ["ii"] = 8520,
6177 ["frac13"] = 8531,
6178 ["frac23"] = 8532,
6179 ["frac15"] = 8533,
6180 ["frac25"] = 8534,
6181 ["frac35"] = 8535,
6182 ["frac45"] = 8536,
6183 ["frac16"] = 8537,
6184 ["frac56"] = 8538,
6185 ["frac18"] = 8539,
6186 ["frac38"] = 8540,
6187 ["frac58"] = 8541,
6188 ["frac78"] = 8542,
6189 ["LeftArrow"] = 8592,
6190 ["ShortLeftArrow"] = 8592,
6191 ["larr"] = 8592,
6192 ["leftarrow"] = 8592,
6193 ["slarr"] = 8592,
6194 ["ShortUpArrow"] = 8593,
6195 ["UpArrow"] = 8593,
6196 ["uarr"] = 8593,
6197 ["uparrow"] = 8593,
6198 ["RightArrow"] = 8594,
6199 ["ShortRightArrow"] = 8594,
6200 ["rarr"] = 8594,
6201 ["rightarrow"] = 8594,
6202 ["srarr"] = 8594,
6203 ["DownArrow"] = 8595,
6204 ["ShortDownArrow"] = 8595,
6205 ["darr"] = 8595,
6206 ["downarrow"] = 8595,
6207 ["LeftRightArrow"] = 8596,
6208 ["harr"] = 8596,
6209 ["leftrightarrow"] = 8596,

```

```

6210 ["UpDownArrow"] = 8597,
6211 ["updownarrow"] = 8597,
6212 ["varr"] = 8597,
6213 ["UpperLeftArrow"] = 8598,
6214 ["nwarr"] = 8598,
6215 ["nwarrow"] = 8598,
6216 ["UpperRightArrow"] = 8599,
6217 ["nearr"] = 8599,
6218 ["nearrow"] = 8599,
6219 ["LowerRightArrow"] = 8600,
6220 ["searr"] = 8600,
6221 ["searrow"] = 8600,
6222 ["LowerLeftArrow"] = 8601,
6223 ["swarr"] = 8601,
6224 ["swarrow"] = 8601,
6225 ["nlarr"] = 8602,
6226 ["nleftarrow"] = 8602,
6227 ["nrarr"] = 8603,
6228 ["nrightarrow"] = 8603,
6229 ["nrarrw"] = {8605, 824},
6230 ["rarrw"] = 8605,
6231 ["rightsquigarrow"] = 8605,
6232 ["Larr"] = 8606,
6233 ["twoheadleftarrow"] = 8606,
6234 ["Uarr"] = 8607,
6235 ["Rarr"] = 8608,
6236 ["twoheadrightarrow"] = 8608,
6237 ["Darr"] = 8609,
6238 ["larrtl"] = 8610,
6239 ["leftarrowtail"] = 8610,
6240 ["rarrtl"] = 8611,
6241 ["rightarrowtail"] = 8611,
6242 ["LeftTeeArrow"] = 8612,
6243 ["mapstoleft"] = 8612,
6244 ["UpTeeArrow"] = 8613,
6245 ["mapstoup"] = 8613,
6246 ["RightTeeArrow"] = 8614,
6247 ["map"] = 8614,
6248 ["mapsto"] = 8614,
6249 ["DownTeeArrow"] = 8615,
6250 ["mapstodown"] = 8615,
6251 ["hookleftarrow"] = 8617,
6252 ["larrhk"] = 8617,
6253 ["hookrightarrow"] = 8618,
6254 ["rarrhk"] = 8618,
6255 ["larrlp"] = 8619,
6256 ["looparrowleft"] = 8619,

```



```

6257 ["looparrowright"] = 8620,
6258 ["rarrlp"] = 8620,
6259 ["harrw"] = 8621,
6260 ["leftrightsquigarrow"] = 8621,
6261 ["nharr"] = 8622,
6262 ["nleftrightarrow"] = 8622,
6263 ["Lsh"] = 8624,
6264 ["lsh"] = 8624,
6265 ["Rsh"] = 8625,
6266 ["rsh"] = 8625,
6267 ["ldsh"] = 8626,
6268 ["rdsh"] = 8627,
6269 ["crarr"] = 8629,
6270 ["cularr"] = 8630,
6271 ["curvearrowleft"] = 8630,
6272 ["curarr"] = 8631,
6273 ["curvearrowright"] = 8631,
6274 ["circlearrowleft"] = 8634,
6275 ["olarr"] = 8634,
6276 ["circlearrowright"] = 8635,
6277 ["orarr"] = 8635,
6278 ["LeftVector"] = 8636,
6279 ["leftharpoonup"] = 8636,
6280 ["lharu"] = 8636,
6281 ["DownLeftVector"] = 8637,
6282 ["leftharpoondown"] = 8637,
6283 ["lhard"] = 8637,
6284 ["RightUpVector"] = 8638,
6285 ["uharr"] = 8638,
6286 ["upharpoonright"] = 8638,
6287 ["LeftUpVector"] = 8639,
6288 ["uharl"] = 8639,
6289 ["upharpoonleft"] = 8639,
6290 ["RightVector"] = 8640,
6291 ["rharu"] = 8640,
6292 ["rightharpoonup"] = 8640,
6293 ["DownRightVector"] = 8641,
6294 ["rhard"] = 8641,
6295 ["rightharpoondown"] = 8641,
6296 ["RightDownVector"] = 8642,
6297 ["dharr"] = 8642,
6298 ["downharpoonright"] = 8642,
6299 ["LeftDownVector"] = 8643,
6300 ["dharl"] = 8643,
6301 ["downharpoonleft"] = 8643,
6302 ["RightArrowLeftArrow"] = 8644,
6303 ["rightleftarrows"] = 8644,

```

```

6304 ["rlarr"] = 8644,
6305 ["UpArrowDownArrow"] = 8645,
6306 ["udarr"] = 8645,
6307 ["LeftArrowRightArrow"] = 8646,
6308 ["leftrightharpoons"] = 8646,
6309 ["lrarr"] = 8646,
6310 ["leftleftarrows"] = 8647,
6311 ["llarr"] = 8647,
6312 ["upuparrows"] = 8648,
6313 ["uuarr"] = 8648,
6314 ["rightrightarrows"] = 8649,
6315 ["rrarr"] = 8649,
6316 ["ddarr"] = 8650,
6317 ["downdownarrows"] = 8650,
6318 ["ReverseEquilibrium"] = 8651,
6319 ["leftrightharpoons"] = 8651,
6320 ["lrhar"] = 8651,
6321 ["Equilibrium"] = 8652,
6322 ["rightleftharpoons"] = 8652,
6323 ["rlhar"] = 8652,
6324 ["nLeftarrow"] = 8653,
6325 ["nLArr"] = 8653,
6326 ["nLeftrightarrow"] = 8654,
6327 ["nhArr"] = 8654,
6328 ["nRightarrow"] = 8655,
6329 ["nrArr"] = 8655,
6330 ["DoubleLeftArrow"] = 8656,
6331 ["Leftarrow"] = 8656,
6332 ["lArr"] = 8656,
6333 ["DoubleUpArrow"] = 8657,
6334 ["Uparrow"] = 8657,
6335 ["uArr"] = 8657,
6336 ["DoubleRightArrow"] = 8658,
6337 ["Implies"] = 8658,
6338 ["Rightarrow"] = 8658,
6339 ["rArr"] = 8658,
6340 ["DoubleDownArrow"] = 8659,
6341 ["Downarrow"] = 8659,
6342 ["dArr"] = 8659,
6343 ["DoubleLeftRightArrow"] = 8660,
6344 ["Leftrightarrow"] = 8660,
6345 ["hArr"] = 8660,
6346 ["iff"] = 8660,
6347 ["DoubleUpDownArrow"] = 8661,
6348 ["Updownarrow"] = 8661,
6349 ["vArr"] = 8661,
6350 ["nwArr"] = 8662,

```

```

6351 ["neArr"] = 8663,
6352 ["seArr"] = 8664,
6353 ["swArr"] = 8665,
6354 ["Lleftarrow"] = 8666,
6355 ["lAarr"] = 8666,
6356 ["Rrightarrow"] = 8667,
6357 ["rAarr"] = 8667,
6358 ["zigrarr"] = 8669,
6359 ["LeftArrowBar"] = 8676,
6360 ["larrb"] = 8676,
6361 ["RightArrowBar"] = 8677,
6362 ["rarrb"] = 8677,
6363 ["DownArrowUpArrow"] = 8693,
6364 ["duarr"] = 8693,
6365 ["loarr"] = 8701,
6366 ["roarr"] = 8702,
6367 ["hoarr"] = 8703,
6368 ["ForAll"] = 8704,
6369 ["forall"] = 8704,
6370 ["comp"] = 8705,
6371 ["complement"] = 8705,
6372 ["PartialD"] = 8706,
6373 ["npart"] = {8706, 824},
6374 ["part"] = 8706,
6375 ["Exists"] = 8707,
6376 ["exist"] = 8707,
6377 ["NotExists"] = 8708,
6378 ["nexist"] = 8708,
6379 ["nexists"] = 8708,
6380 ["empty"] = 8709,
6381 ["emptyset"] = 8709,
6382 ["emptyv"] = 8709,
6383 ["varnothing"] = 8709,
6384 ["Del"] = 8711,
6385 ["nabla"] = 8711,
6386 ["Element"] = 8712,
6387 ["in"] = 8712,
6388 ["isin"] = 8712,
6389 ["isinv"] = 8712,
6390 ["NotElement"] = 8713,
6391 ["notin"] = 8713,
6392 ["notinva"] = 8713,
6393 ["ReverseElement"] = 8715,
6394 ["SuchThat"] = 8715,
6395 ["ni"] = 8715,
6396 ["niv"] = 8715,
6397 ["NotReverseElement"] = 8716,

```

```

6398 ["notni"] = 8716,
6399 ["notniva"] = 8716,
6400 ["Product"] = 8719,
6401 ["prod"] = 8719,
6402 ["Coproduct"] = 8720,
6403 ["coprod"] = 8720,
6404 ["Sum"] = 8721,
6405 ["sum"] = 8721,
6406 ["minus"] = 8722,
6407 ["MinusPlus"] = 8723,
6408 ["mnplus"] = 8723,
6409 ["mp"] = 8723,
6410 ["dotplus"] = 8724,
6411 ["plusdo"] = 8724,
6412 ["Backslash"] = 8726,
6413 ["setminus"] = 8726,
6414 ["setmn"] = 8726,
6415 ["smallsetminus"] = 8726,
6416 ["ssetmn"] = 8726,
6417 ["lowast"] = 8727,
6418 ["SmallCircle"] = 8728,
6419 ["compfn"] = 8728,
6420 ["Sqrt"] = 8730,
6421 ["radic"] = 8730,
6422 ["Proportional"] = 8733,
6423 ["prop"] = 8733,
6424 ["propto"] = 8733,
6425 ["varpropto"] = 8733,
6426 ["vprop"] = 8733,
6427 ["infin"] = 8734,
6428 ["angrt"] = 8735,
6429 ["ang"] = 8736,
6430 ["angle"] = 8736,
6431 ["nang"] = {8736, 8402},
6432 ["angmsd"] = 8737,
6433 ["measuredangle"] = 8737,
6434 ["angsph"] = 8738,
6435 ["VerticalBar"] = 8739,
6436 ["mid"] = 8739,
6437 ["shortmid"] = 8739,
6438 ["smid"] = 8739,
6439 ["NotVerticalBar"] = 8740,
6440 ["nmid"] = 8740,
6441 ["nshortmid"] = 8740,
6442 ["nsmid"] = 8740,
6443 ["DoubleVerticalBar"] = 8741,
6444 ["par"] = 8741,

```

```

6445 ["parallel"] = 8741,
6446 ["shortparallel"] = 8741,
6447 ["spar"] = 8741,
6448 ["NotDoubleVerticalBar"] = 8742,
6449 ["npar"] = 8742,
6450 ["nparallel"] = 8742,
6451 ["nshortparallel"] = 8742,
6452 ["nspar"] = 8742,
6453 ["and"] = 8743,
6454 ["wedge"] = 8743,
6455 ["or"] = 8744,
6456 ["vee"] = 8744,
6457 ["cap"] = 8745,
6458 ["caps"] = {8745, 65024},
6459 ["cup"] = 8746,
6460 ["cups"] = {8746, 65024},
6461 ["Integral"] = 8747,
6462 ["int"] = 8747,
6463 ["Int"] = 8748,
6464 ["iiint"] = 8749,
6465 ["tint"] = 8749,
6466 ["ContourIntegral"] = 8750,
6467 ["conint"] = 8750,
6468 ["oint"] = 8750,
6469 ["Conint"] = 8751,
6470 ["DoubleContourIntegral"] = 8751,
6471 ["Cconint"] = 8752,
6472 ["cwint"] = 8753,
6473 ["ClockwiseContourIntegral"] = 8754,
6474 ["cwconint"] = 8754,
6475 ["CounterClockwiseContourIntegral"] = 8755,
6476 ["awconint"] = 8755,
6477 ["Therefore"] = 8756,
6478 ["there4"] = 8756,
6479 ["therefore"] = 8756,
6480 ["Because"] = 8757,
6481 ["becaus"] = 8757,
6482 ["because"] = 8757,
6483 ["ratio"] = 8758,
6484 ["Colon"] = 8759,
6485 ["Proportion"] = 8759,
6486 ["dotminus"] = 8760,
6487 ["minusd"] = 8760,
6488 ["mDDot"] = 8762,
6489 ["homtht"] = 8763,
6490 ["Tilde"] = 8764,
6491 ["nvsim"] = {8764, 8402},

```

```

6492 ["sim"] = 8764,
6493 ["thicksim"] = 8764,
6494 ["thksim"] = 8764,
6495 ["backsim"] = 8765,
6496 ["bsim"] = 8765,
6497 ["race"] = {8765, 817},
6498 ["ac"] = 8766,
6499 ["acE"] = {8766, 819},
6500 ["mstpos"] = 8766,
6501 ["acd"] = 8767,
6502 ["VerticalTilde"] = 8768,
6503 ["wr"] = 8768,
6504 ["wreath"] = 8768,
6505 ["NotTilde"] = 8769,
6506 ["nsim"] = 8769,
6507 ["EqualTilde"] = 8770,
6508 ["NotEqualTilde"] = {8770, 824},
6509 ["eqsim"] = 8770,
6510 ["esim"] = 8770,
6511 ["nesim"] = {8770, 824},
6512 ["TildeEqual"] = 8771,
6513 ["sime"] = 8771,
6514 ["simeq"] = 8771,
6515 ["NotTildeEqual"] = 8772,
6516 ["nsime"] = 8772,
6517 ["nsimeq"] = 8772,
6518 ["TildeFullEqual"] = 8773,
6519 ["cong"] = 8773,
6520 ["simne"] = 8774,
6521 ["NotTildeFullEqual"] = 8775,
6522 ["ncong"] = 8775,
6523 ["TildeTilde"] = 8776,
6524 ["ap"] = 8776,
6525 ["approx"] = 8776,
6526 ["asyp"] = 8776,
6527 ["thickapprox"] = 8776,
6528 ["thkap"] = 8776,
6529 ["NotTildeTilde"] = 8777,
6530 ["nap"] = 8777,
6531 ["napprox"] = 8777,
6532 ["ape"] = 8778,
6533 ["approxeq"] = 8778,
6534 ["apid"] = 8779,
6535 ["napid"] = {8779, 824},
6536 ["backcong"] = 8780,
6537 ["bcong"] = 8780,
6538 ["CupCap"] = 8781,

```

```

6539 ["asympeq"] = 8781,
6540 ["nvap"] = {8781, 8402},
6541 ["Bumpeq"] = 8782,
6542 ["HumpDownHump"] = 8782,
6543 ["NotHumpDownHump"] = {8782, 824},
6544 ["bump"] = 8782,
6545 ["nbump"] = {8782, 824},
6546 ["HumpEqual"] = 8783,
6547 ["NotHumpEqual"] = {8783, 824},
6548 ["bumpe"] = 8783,
6549 ["bumpeq"] = 8783,
6550 ["nbumpe"] = {8783, 824},
6551 ["DotEqual"] = 8784,
6552 ["doteq"] = 8784,
6553 ["esdot"] = 8784,
6554 ["nedot"] = {8784, 824},
6555 ["doteqdot"] = 8785,
6556 ["eDot"] = 8785,
6557 ["efDot"] = 8786,
6558 ["fallingdotseq"] = 8786,
6559 ["erDot"] = 8787,
6560 ["risingdotseq"] = 8787,
6561 ["Assign"] = 8788,
6562 ["colone"] = 8788,
6563 ["coloneq"] = 8788,
6564 ["ecolon"] = 8789,
6565 ["eqcolon"] = 8789,
6566 ["ecir"] = 8790,
6567 ["eqcirc"] = 8790,
6568 ["circeq"] = 8791,
6569 ["cire"] = 8791,
6570 ["wedgeq"] = 8793,
6571 ["veeeq"] = 8794,
6572 ["triangleq"] = 8796,
6573 ["trie"] = 8796,
6574 ["equest"] = 8799,
6575 ["questeq"] = 8799,
6576 ["NotEqual"] = 8800,
6577 ["ne"] = 8800,
6578 ["Congruent"] = 8801,
6579 ["bnequiv"] = {8801, 8421},
6580 ["equiv"] = 8801,
6581 ["NotCongruent"] = 8802,
6582 ["nequiv"] = 8802,
6583 ["le"] = 8804,
6584 ["leq"] = 8804,
6585 ["nvle"] = {8804, 8402},

```

```

6586 ["GreaterEqual"] = 8805,
6587 ["ge"] = 8805,
6588 ["geq"] = 8805,
6589 ["nvge"] = {8805, 8402},
6590 ["LessFullEqual"] = 8806,
6591 ["lE"] = 8806,
6592 ["leqq"] = 8806,
6593 ["nlE"] = {8806, 824},
6594 ["nleqq"] = {8806, 824},
6595 ["GreaterFullEqual"] = 8807,
6596 ["NotGreaterFullEqual"] = {8807, 824},
6597 ["gE"] = 8807,
6598 ["geqq"] = 8807,
6599 ["ngE"] = {8807, 824},
6600 ["ngeqq"] = {8807, 824},
6601 ["lnE"] = 8808,
6602 ["lneqq"] = 8808,
6603 ["lvertneqq"] = {8808, 65024},
6604 ["lvnE"] = {8808, 65024},
6605 ["gnE"] = 8809,
6606 ["gneqq"] = 8809,
6607 ["gvertneqq"] = {8809, 65024},
6608 ["gvnE"] = {8809, 65024},
6609 ["Lt"] = 8810,
6610 ["NestedLessLess"] = 8810,
6611 ["NotLessLess"] = {8810, 824},
6612 ["ll"] = 8810,
6613 ["nLt"] = {8810, 8402},
6614 ["nLtv"] = {8810, 824},
6615 ["Gt"] = 8811,
6616 ["NestedGreaterGreater"] = 8811,
6617 ["NotGreaterGreater"] = {8811, 824},
6618 ["gg"] = 8811,
6619 ["nGt"] = {8811, 8402},
6620 ["nGtv"] = {8811, 824},
6621 ["between"] = 8812,
6622 ["twixt"] = 8812,
6623 ["NotCupCap"] = 8813,
6624 ["NotLess"] = 8814,
6625 ["nless"] = 8814,
6626 ["nlt"] = 8814,
6627 ["NotGreater"] = 8815,
6628 ["ngt"] = 8815,
6629 ["ngtr"] = 8815,
6630 ["NotLessEqual"] = 8816,
6631 ["nle"] = 8816,
6632 ["nleq"] = 8816,

```



```

6633 ["NotGreaterEqual"] = 8817,
6634 ["nge"] = 8817,
6635 ["ngeq"] = 8817,
6636 ["LessTilde"] = 8818,
6637 ["lesssim"] = 8818,
6638 ["lsim"] = 8818,
6639 ["GreaterTilde"] = 8819,
6640 ["gsim"] = 8819,
6641 ["gtrsim"] = 8819,
6642 ["NotLessTilde"] = 8820,
6643 ["nlsim"] = 8820,
6644 ["NotGreaterTilde"] = 8821,
6645 ["ngsim"] = 8821,
6646 ["LessGreater"] = 8822,
6647 ["lessgtr"] = 8822,
6648 ["lg"] = 8822,
6649 ["GreaterLess"] = 8823,
6650 ["gl"] = 8823,
6651 ["gtrless"] = 8823,
6652 ["NotLessGreater"] = 8824,
6653 ["ntl原因"] = 8824,
6654 ["NotGreaterLess"] = 8825,
6655 ["ntgl"] = 8825,
6656 ["Precedes"] = 8826,
6657 ["pr"] = 8826,
6658 ["prec"] = 8826,
6659 ["Succeeds"] = 8827,
6660 ["sc"] = 8827,
6661 ["succ"] = 8827,
6662 ["PrecedesSlantEqual"] = 8828,
6663 ["prcue"] = 8828,
6664 ["preccurlyeq"] = 8828,
6665 ["SucceedsSlantEqual"] = 8829,
6666 ["sccue"] = 8829,
6667 ["succcurlyeq"] = 8829,
6668 ["PrecedesTilde"] = 8830,
6669 ["precsim"] = 8830,
6670 ["prsim"] = 8830,
6671 ["NotSucceedsTilde"] = {8831, 824},
6672 ["SucceedsTilde"] = 8831,
6673 ["scsim"] = 8831,
6674 ["succsim"] = 8831,
6675 ["NotPrecedes"] = 8832,
6676 ["npr"] = 8832,
6677 ["nprec"] = 8832,
6678 ["NotSucceeds"] = 8833,
6679 ["nsc"] = 8833,

```

```

6680 ["nsucc"] = 8833,
6681 ["NotSubset"] = {8834, 8402},
6682 ["nsubset"] = {8834, 8402},
6683 ["sub"] = 8834,
6684 ["subset"] = 8834,
6685 ["vnsup"] = {8834, 8402},
6686 ["NotSuperset"] = {8835, 8402},
6687 ["Superset"] = 8835,
6688 ["nsupset"] = {8835, 8402},
6689 ["sup"] = 8835,
6690 ["supset"] = 8835,
6691 ["vnsup"] = {8835, 8402},
6692 ["nsub"] = 8836,
6693 ["nsup"] = 8837,
6694 ["SubsetEqual"] = 8838,
6695 ["sube"] = 8838,
6696 ["subseteq"] = 8838,
6697 ["SupersetEqual"] = 8839,
6698 ["supe"] = 8839,
6699 ["supseteq"] = 8839,
6700 ["NotSubsetEqual"] = 8840,
6701 ["nsube"] = 8840,
6702 ["nsubseteq"] = 8840,
6703 ["NotSupersetEqual"] = 8841,
6704 ["nsupe"] = 8841,
6705 ["nsupseteq"] = 8841,
6706 ["subne"] = 8842,
6707 ["subsetneq"] = 8842,
6708 ["varsubsetneq"] = {8842, 65024},
6709 ["vsubne"] = {8842, 65024},
6710 ["supne"] = 8843,
6711 ["supsetneq"] = 8843,
6712 ["varsupsetneq"] = {8843, 65024},
6713 ["vsupne"] = {8843, 65024},
6714 ["cupdot"] = 8845,
6715 ["UnionPlus"] = 8846,
6716 ["uplus"] = 8846,
6717 ["NotSquareSubset"] = {8847, 824},
6718 ["SquareSubset"] = 8847,
6719 ["sqsub"] = 8847,
6720 ["sqsubset"] = 8847,
6721 ["NotSquareSuperset"] = {8848, 824},
6722 ["SquareSuperset"] = 8848,
6723 ["sqsup"] = 8848,
6724 ["sqsupset"] = 8848,
6725 ["SquareSubsetEqual"] = 8849,
6726 ["sqsube"] = 8849,

```

```

6727 ["sqsubsepeq"] = 8849,
6728 ["SquareSupersetEqual"] = 8850,
6729 ["sqsupe"] = 8850,
6730 ["sqsupseteq"] = 8850,
6731 ["SquareIntersection"] = 8851,
6732 ["sqcap"] = 8851,
6733 ["sqcaps"] = {8851, 65024},
6734 ["SquareUnion"] = 8852,
6735 ["sqcup"] = 8852,
6736 ["sqcups"] = {8852, 65024},
6737 ["CirclePlus"] = 8853,
6738 ["oplus"] = 8853,
6739 ["CircleMinus"] = 8854,
6740 ["ominus"] = 8854,
6741 ["CircleTimes"] = 8855,
6742 ["otimes"] = 8855,
6743 ["osol"] = 8856,
6744 ["CircleDot"] = 8857,
6745 ["odot"] = 8857,
6746 ["circledcirc"] = 8858,
6747 ["ocir"] = 8858,
6748 ["circledast"] = 8859,
6749 ["oast"] = 8859,
6750 ["circleddash"] = 8861,
6751 ["odash"] = 8861,
6752 ["boxplus"] = 8862,
6753 ["plusb"] = 8862,
6754 ["boxminus"] = 8863,
6755 ["minusb"] = 8863,
6756 ["boxtimes"] = 8864,
6757 ["timesb"] = 8864,
6758 ["dotsquare"] = 8865,
6759 ["sdotb"] = 8865,
6760 ["RightTee"] = 8866,
6761 ["vdash"] = 8866,
6762 ["LeftTee"] = 8867,
6763 ["dashv"] = 8867,
6764 ["DownTee"] = 8868,
6765 ["top"] = 8868,
6766 ["UpTee"] = 8869,
6767 ["bot"] = 8869,
6768 ["bottom"] = 8869,
6769 ["perp"] = 8869,
6770 ["models"] = 8871,
6771 ["DoubleRightTee"] = 8872,
6772 ["vDash"] = 8872,
6773 ["Vdash"] = 8873,

```

```

6774 ["Vvdash"] = 8874,
6775 ["VDash"] = 8875,
6776 ["nvdash"] = 8876,
6777 ["nvDash"] = 8877,
6778 ["nVdash"] = 8878,
6779 ["nVDash"] = 8879,
6780 ["prurel"] = 8880,
6781 ["LeftTriangle"] = 8882,
6782 ["vartriangleleft"] = 8882,
6783 ["vltri"] = 8882,
6784 ["RightTriangle"] = 8883,
6785 ["vartriangleright"] = 8883,
6786 ["vrtri"] = 8883,
6787 ["LeftTriangleEqual"] = 8884,
6788 ["ltrie"] = 8884,
6789 ["nvltrie"] = {8884, 8402},
6790 ["trianglelefteq"] = 8884,
6791 ["RightTriangleEqual"] = 8885,
6792 ["nvrtrie"] = {8885, 8402},
6793 ["rtrie"] = 8885,
6794 ["trianglerighteq"] = 8885,
6795 ["origof"] = 8886,
6796 ["imof"] = 8887,
6797 ["multimap"] = 8888,
6798 ["mumap"] = 8888,
6799 ["hercon"] = 8889,
6800 ["intcal"] = 8890,
6801 ["intercal"] = 8890,
6802 ["veebar"] = 8891,
6803 ["barvee"] = 8893,
6804 ["angrtvb"] = 8894,
6805 ["lrltri"] = 8895,
6806 ["Wedge"] = 8896,
6807 ["bigwedge"] = 8896,
6808 ["xwedge"] = 8896,
6809 ["Vee"] = 8897,
6810 ["bigvee"] = 8897,
6811 ["xvee"] = 8897,
6812 ["Intersection"] = 8898,
6813 ["bigcap"] = 8898,
6814 ["xcap"] = 8898,
6815 ["Union"] = 8899,
6816 ["bigcup"] = 8899,
6817 ["xcup"] = 8899,
6818 ["Diamond"] = 8900,
6819 ["diam"] = 8900,
6820 ["diamond"] = 8900,

```

```

6821 ["sdot"] = 8901,
6822 ["Star"] = 8902,
6823 ["sstarf"] = 8902,
6824 ["divideontimes"] = 8903,
6825 ["divonx"] = 8903,
6826 ["bowtie"] = 8904,
6827 ["ltimes"] = 8905,
6828 ["rtimes"] = 8906,
6829 ["leftthreetimes"] = 8907,
6830 ["lthree"] = 8907,
6831 ["rightthreetimes"] = 8908,
6832 ["rthree"] = 8908,
6833 ["backsimeq"] = 8909,
6834 ["bsime"] = 8909,
6835 ["curlyvee"] = 8910,
6836 ["cuvee"] = 8910,
6837 ["curlywedge"] = 8911,
6838 ["cuwed"] = 8911,
6839 ["Sub"] = 8912,
6840 ["Subset"] = 8912,
6841 ["Sup"] = 8913,
6842 ["Supset"] = 8913,
6843 ["Cap"] = 8914,
6844 ["Cup"] = 8915,
6845 ["fork"] = 8916,
6846 ["pitchfork"] = 8916,
6847 ["epar"] = 8917,
6848 ["lessdot"] = 8918,
6849 ["ltdot"] = 8918,
6850 ["gtdot"] = 8919,
6851 ["gtrdot"] = 8919,
6852 ["Ll"] = 8920,
6853 ["nLl"] = {8920, 824},
6854 ["Gg"] = 8921,
6855 ["ggg"] = 8921,
6856 ["nGg"] = {8921, 824},
6857 ["LessEqualGreater"] = 8922,
6858 ["leg"] = 8922,
6859 ["lesg"] = {8922, 65024},
6860 ["lesseqgtr"] = 8922,
6861 ["GreaterEqualLess"] = 8923,
6862 ["gel"] = 8923,
6863 ["gesl"] = {8923, 65024},
6864 ["gtreqless"] = 8923,
6865 ["cuepr"] = 8926,
6866 ["curlyeqprec"] = 8926,
6867 ["cuesc"] = 8927,

```

6868 ["curlyeqsucc"] = 8927,  
 6869 ["NotPrecedesSlantEqual"] = 8928,  
 6870 ["nprcue"] = 8928,  
 6871 ["NotSucceedsSlantEqual"] = 8929,  
 6872 ["nsccue"] = 8929,  
 6873 ["NotSquareSubsetEqual"] = 8930,  
 6874 ["nsqsube"] = 8930,  
 6875 ["NotSquareSupersetEqual"] = 8931,  
 6876 ["nsqsupe"] = 8931,  
 6877 ["lnsim"] = 8934,  
 6878 ["gnsim"] = 8935,  
 6879 ["precnsim"] = 8936,  
 6880 ["prnsim"] = 8936,  
 6881 ["scnsim"] = 8937,  
 6882 ["succnsim"] = 8937,  
 6883 ["NotLeftTriangle"] = 8938,  
 6884 ["nltri"] = 8938,  
 6885 ["ntriangleleft"] = 8938,  
 6886 ["NotRightTriangle"] = 8939,  
 6887 ["nrtri"] = 8939,  
 6888 ["ntriangleright"] = 8939,  
 6889 ["NotLeftTriangleEqual"] = 8940,  
 6890 ["nltrie"] = 8940,  
 6891 ["ntrianglelefteq"] = 8940,  
 6892 ["NotRightTriangleEqual"] = 8941,  
 6893 ["nrtrie"] = 8941,  
 6894 ["ntrianglerighteq"] = 8941,  
 6895 ["vellip"] = 8942,  
 6896 ["ctdot"] = 8943,  
 6897 ["utdot"] = 8944,  
 6898 ["dtdot"] = 8945,  
 6899 ["disin"] = 8946,  
 6900 ["isinsv"] = 8947,  
 6901 ["isins"] = 8948,  
 6902 ["isindot"] = 8949,  
 6903 ["notin dot"] = {8949, 824},  
 6904 ["notinvc"] = 8950,  
 6905 ["notin vb"] = 8951,  
 6906 ["isinE"] = 8953,  
 6907 ["notinE"] = {8953, 824},  
 6908 ["nisd"] = 8954,  
 6909 ["xnis"] = 8955,  
 6910 ["nis"] = 8956,  
 6911 ["notnivc"] = 8957,  
 6912 ["notnivb"] = 8958,  
 6913 ["barwed"] = 8965,  
 6914 ["barwedge"] = 8965,

```

6915 ["Barwed"] = 8966,
6916 ["doublebarwedge"] = 8966,
6917 ["LeftCeiling"] = 8968,
6918 ["lceil"] = 8968,
6919 ["RightCeiling"] = 8969,
6920 ["rceil"] = 8969,
6921 ["LeftFloor"] = 8970,
6922 ["lfloor"] = 8970,
6923 ["RightFloor"] = 8971,
6924 ["rfloor"] = 8971,
6925 ["drcrop"] = 8972,
6926 ["dlcrop"] = 8973,
6927 ["urcrop"] = 8974,
6928 ["ulcrop"] = 8975,
6929 ["bnot"] = 8976,
6930 ["profline"] = 8978,
6931 ["profsurf"] = 8979,
6932 ["telrec"] = 8981,
6933 ["target"] = 8982,
6934 ["ulcorn"] = 8988,
6935 ["ulcorner"] = 8988,
6936 ["urcorn"] = 8989,
6937 ["urcorner"] = 8989,
6938 ["dlcorn"] = 8990,
6939 ["llcorner"] = 8990,
6940 ["drcorn"] = 8991,
6941 ["lrcorner"] = 8991,
6942 ["frown"] = 8994,
6943 ["sfrown"] = 8994,
6944 ["smile"] = 8995,
6945 ["ssmile"] = 8995,
6946 ["cylcty"] = 9005,
6947 ["profalar"] = 9006,
6948 ["topbot"] = 9014,
6949 ["ovbar"] = 9021,
6950 ["solbar"] = 9023,
6951 ["angzarr"] = 9084,
6952 ["lmoust"] = 9136,
6953 ["lmoustache"] = 9136,
6954 ["rmoust"] = 9137,
6955 ["rmoustache"] = 9137,
6956 ["OverBracket"] = 9140,
6957 ["tbrk"] = 9140,
6958 ["UnderBracket"] = 9141,
6959 ["bbrk"] = 9141,
6960 ["bbrktbrk"] = 9142,
6961 ["OverParenthesis"] = 9180,

```

```

6962 ["UnderParenthesis"] = 9181,
6963 ["OverBrace"] = 9182,
6964 ["UnderBrace"] = 9183,
6965 ["trpezium"] = 9186,
6966 ["elinters"] = 9191,
6967 ["blank"] = 9251,
6968 ["circledS"] = 9416,
6969 ["oS"] = 9416,
6970 ["HorizontalLine"] = 9472,
6971 ["boxh"] = 9472,
6972 ["boxv"] = 9474,
6973 ["boxdr"] = 9484,
6974 ["boxdl"] = 9488,
6975 ["boxur"] = 9492,
6976 ["boxul"] = 9496,
6977 ["boxvr"] = 9500,
6978 ["boxvl"] = 9508,
6979 ["boxhd"] = 9516,
6980 ["boxhu"] = 9524,
6981 ["boxvh"] = 9532,
6982 ["boxH"] = 9552,
6983 ["boxV"] = 9553,
6984 ["boxdR"] = 9554,
6985 ["boxDr"] = 9555,
6986 ["boxDR"] = 9556,
6987 ["boxdL"] = 9557,
6988 ["boxDL"] = 9558,
6989 ["boxDL"] = 9559,
6990 ["boxuR"] = 9560,
6991 ["boxUr"] = 9561,
6992 ["boxUR"] = 9562,
6993 ["boxuL"] = 9563,
6994 ["boxUL"] = 9564,
6995 ["boxUL"] = 9565,
6996 ["boxvR"] = 9566,
6997 ["boxVr"] = 9567,
6998 ["boxVR"] = 9568,
6999 ["boxvL"] = 9569,
7000 ["boxVl"] = 9570,
7001 ["boxVL"] = 9571,
7002 ["boxHd"] = 9572,
7003 ["boxhD"] = 9573,
7004 ["boxHD"] = 9574,
7005 ["boxHu"] = 9575,
7006 ["boxhU"] = 9576,
7007 ["boxHU"] = 9577,
7008 ["boxvH"] = 9578,

```



```

7009 ["boxVh"] = 9579,
7010 ["boxVH"] = 9580,
7011 ["uhblk"] = 9600,
7012 ["lhblk"] = 9604,
7013 ["block"] = 9608,
7014 ["blk14"] = 9617,
7015 ["blk12"] = 9618,
7016 ["blk34"] = 9619,
7017 ["Square"] = 9633,
7018 ["squ"] = 9633,
7019 ["square"] = 9633,
7020 ["FilledVerySmallSquare"] = 9642,
7021 ["blacksquare"] = 9642,
7022 ["suarf"] = 9642,
7023 ["squf"] = 9642,
7024 ["EmptyVerySmallSquare"] = 9643,
7025 ["rect"] = 9645,
7026 ["marker"] = 9646,
7027 ["fltns"] = 9649,
7028 ["bigtriangleup"] = 9651,
7029 ["xutri"] = 9651,
7030 ["blacktriangle"] = 9652,
7031 ["utrif"] = 9652,
7032 ["triangle"] = 9653,
7033 ["utri"] = 9653,
7034 ["blacktriangleright"] = 9656,
7035 ["rtrif"] = 9656,
7036 ["rtri"] = 9657,
7037 ["triangleright"] = 9657,
7038 ["bigtriangledown"] = 9661,
7039 ["xdtri"] = 9661,
7040 ["blacktriangledown"] = 9662,
7041 ["dtrif"] = 9662,
7042 ["dtri"] = 9663,
7043 ["triangledown"] = 9663,
7044 ["blacktriangleleft"] = 9666,
7045 ["ltrif"] = 9666,
7046 ["ltri"] = 9667,
7047 ["triangleleft"] = 9667,
7048 ["loz"] = 9674,
7049 ["lozenge"] = 9674,
7050 ["cir"] = 9675,
7051 ["tridot"] = 9708,
7052 ["bigcirc"] = 9711,
7053 ["xcirc"] = 9711,
7054 ["ultri"] = 9720,
7055 ["urtri"] = 9721,

```

7056 ["lltri"] = 9722,  
 7057 ["EmptySmallSquare"] = 9723,  
 7058 ["FilledSmallSquare"] = 9724,  
 7059 ["bigstar"] = 9733,  
 7060 ["starf"] = 9733,  
 7061 ["star"] = 9734,  
 7062 ["phone"] = 9742,  
 7063 ["female"] = 9792,  
 7064 ["male"] = 9794,  
 7065 ["spades"] = 9824,  
 7066 ["spadesuit"] = 9824,  
 7067 ["clubs"] = 9827,  
 7068 ["clubsuit"] = 9827,  
 7069 ["hearts"] = 9829,  
 7070 ["heartsuit"] = 9829,  
 7071 ["diamondsuit"] = 9830,  
 7072 ["diams"] = 9830,  
 7073 ["sung"] = 9834,  
 7074 ["flat"] = 9837,  
 7075 ["natur"] = 9838,  
 7076 ["natural"] = 9838,  
 7077 ["sharp"] = 9839,  
 7078 ["check"] = 10003,  
 7079 ["checkmark"] = 10003,  
 7080 ["cross"] = 10007,  
 7081 ["malt"] = 10016,  
 7082 ["maltese"] = 10016,  
 7083 ["sext"] = 10038,  
 7084 ["VerticalSeparator"] = 10072,  
 7085 ["lbbbrk"] = 10098,  
 7086 ["rbbrk"] = 10099,  
 7087 ["bsolhsub"] = 10184,  
 7088 ["suphsol"] = 10185,  
 7089 ["LeftDoubleBracket"] = 10214,  
 7090 ["lobrk"] = 10214,  
 7091 ["RightDoubleBracket"] = 10215,  
 7092 ["robrk"] = 10215,  
 7093 ["LeftAngleBracket"] = 10216,  
 7094 ["lang"] = 10216,  
 7095 ["langle"] = 10216,  
 7096 ["RightAngleBracket"] = 10217,  
 7097 ["rang"] = 10217,  
 7098 ["rangle"] = 10217,  
 7099 ["Lang"] = 10218,  
 7100 ["Rang"] = 10219,  
 7101 ["loang"] = 10220,  
 7102 ["roang"] = 10221,

```

7103 ["LongLeftArrow"] = 10229,
7104 ["longleftarrow"] = 10229,
7105 ["xlarr"] = 10229,
7106 ["LongRightArrow"] = 10230,
7107 ["longrightarrow"] = 10230,
7108 ["xrarr"] = 10230,
7109 ["LongLeftRightArrow"] = 10231,
7110 ["longlefttrightarrow"] = 10231,
7111 ["xharr"] = 10231,
7112 ["DoubleLongLeftArrow"] = 10232,
7113 ["Longleftarrow"] = 10232,
7114 ["xlArr"] = 10232,
7115 ["DoubleLongRightArrow"] = 10233,
7116 ["Longrightarrow"] = 10233,
7117 ["xrArr"] = 10233,
7118 ["DoubleLongLeftRightArrow"] = 10234,
7119 ["Longlefttrightarrow"] = 10234,
7120 ["xhArr"] = 10234,
7121 ["longmapsto"] = 10236,
7122 ["xmap"] = 10236,
7123 ["dzigrarr"] = 10239,
7124 ["nvlArr"] = 10498,
7125 ["nvrArr"] = 10499,
7126 ["nvHarr"] = 10500,
7127 ["Map"] = 10501,
7128 ["lbarr"] = 10508,
7129 ["bkarow"] = 10509,
7130 ["rbarr"] = 10509,
7131 ["lBarr"] = 10510,
7132 ["dbkarow"] = 10511,
7133 ["rBarr"] = 10511,
7134 ["RBarr"] = 10512,
7135 ["drbkarow"] = 10512,
7136 ["DDottrahd"] = 10513,
7137 ["UpArrowBar"] = 10514,
7138 ["DownArrowBar"] = 10515,
7139 ["Rarrtl"] = 10518,
7140 ["latail"] = 10521,
7141 ["ratail"] = 10522,
7142 ["lAtail"] = 10523,
7143 ["rAtail"] = 10524,
7144 ["larrfs"] = 10525,
7145 ["rarrfs"] = 10526,
7146 ["larrbfs"] = 10527,
7147 ["rarrbfs"] = 10528,
7148 ["nwarhk"] = 10531,
7149 ["nearhk"] = 10532,

```

```

7150 ["hksearow"] = 10533,
7151 ["searhk"] = 10533,
7152 ["hkswarow"] = 10534,
7153 ["swarhk"] = 10534,
7154 ["nwnear"] = 10535,
7155 ["nesear"] = 10536,
7156 ["toea"] = 10536,
7157 ["seswar"] = 10537,
7158 ["tosa"] = 10537,
7159 ["swnwar"] = 10538,
7160 ["nrarrc"] = {10547, 824},
7161 ["rarrc"] = 10547,
7162 ["cudarr"] = 10549,
7163 ["ldca"] = 10550,
7164 ["rdca"] = 10551,
7165 ["cudarrr"] = 10552,
7166 ["larrpl"] = 10553,
7167 ["curarrm"] = 10556,
7168 ["cularrp"] = 10557,
7169 ["rarrpl"] = 10565,
7170 ["harrcir"] = 10568,
7171 ["Uarrocir"] = 10569,
7172 ["lurdshar"] = 10570,
7173 ["ldrushar"] = 10571,
7174 ["LeftRightVector"] = 10574,
7175 ["RightUpDownVector"] = 10575,
7176 ["DownLeftRightVector"] = 10576,
7177 ["LeftUpDownVector"] = 10577,
7178 ["LeftVectorBar"] = 10578,
7179 ["RightVectorBar"] = 10579,
7180 ["RightUpVectorBar"] = 10580,
7181 ["RightDownVectorBar"] = 10581,
7182 ["DownLeftVectorBar"] = 10582,
7183 ["DownRightVectorBar"] = 10583,
7184 ["LeftUpVectorBar"] = 10584,
7185 ["LeftDownVectorBar"] = 10585,
7186 ["LeftTeeVector"] = 10586,
7187 ["RightTeeVector"] = 10587,
7188 ["RightUpTeeVector"] = 10588,
7189 ["RightDownTeeVector"] = 10589,
7190 ["DownLeftTeeVector"] = 10590,
7191 ["DownRightTeeVector"] = 10591,
7192 ["LeftUpTeeVector"] = 10592,
7193 ["LeftDownTeeVector"] = 10593,
7194 ["lHar"] = 10594,
7195 ["uHar"] = 10595,
7196 ["rHar"] = 10596,

```

```

7197 ["dHar"] = 10597,
7198 ["luruhar"] = 10598,
7199 ["ldrdhar"] = 10599,
7200 ["ruluhar"] = 10600,
7201 ["rdldhar"] = 10601,
7202 ["lharul"] = 10602,
7203 ["llhard"] = 10603,
7204 ["rharul"] = 10604,
7205 ["lrhard"] = 10605,
7206 ["UpEquilibrium"] = 10606,
7207 ["udhar"] = 10606,
7208 ["ReverseUpEquilibrium"] = 10607,
7209 ["duhar"] = 10607,
7210 ["RoundImplies"] = 10608,
7211 ["erarr"] = 10609,
7212 ["simrarr"] = 10610,
7213 ["larrsim"] = 10611,
7214 ["rarrsim"] = 10612,
7215 ["rarrap"] = 10613,
7216 ["ltlarr"] = 10614,
7217 ["gtrarr"] = 10616,
7218 ["subrarr"] = 10617,
7219 ["suplarr"] = 10619,
7220 ["lfisht"] = 10620,
7221 ["rfisht"] = 10621,
7222 ["ufisht"] = 10622,
7223 ["dfisht"] = 10623,
7224 ["lopar"] = 10629,
7225 ["ropar"] = 10630,
7226 ["lbrke"] = 10635,
7227 ["rbrke"] = 10636,
7228 ["lbrkslu"] = 10637,
7229 ["rbrksld"] = 10638,
7230 ["lbrksld"] = 10639,
7231 ["rbrkslu"] = 10640,
7232 ["langd"] = 10641,
7233 ["rangd"] = 10642,
7234 ["lparlt"] = 10643,
7235 ["rpargt"] = 10644,
7236 ["gtlPar"] = 10645,
7237 ["ltrPar"] = 10646,
7238 ["vzigzag"] = 10650,
7239 ["vangrt"] = 10652,
7240 ["angrtvbd"] = 10653,
7241 ["ange"] = 10660,
7242 ["range"] = 10661,
7243 ["dwangle"] = 10662,

```

```

7244 ["uwangle"] = 10663,
7245 ["angmsdaa"] = 10664,
7246 ["angmsdab"] = 10665,
7247 ["angmsdac"] = 10666,
7248 ["angmsdad"] = 10667,
7249 ["angmsdae"] = 10668,
7250 ["angmsdaf"] = 10669,
7251 ["angmsdag"] = 10670,
7252 ["angmsdah"] = 10671,
7253 ["bemptyv"] = 10672,
7254 ["demptyv"] = 10673,
7255 ["cemptyv"] = 10674,
7256 ["raemptyv"] = 10675,
7257 ["laemptyv"] = 10676,
7258 ["ohbar"] = 10677,
7259 ["omid"] = 10678,
7260 ["opar"] = 10679,
7261 ["operp"] = 10681,
7262 ["olcross"] = 10683,
7263 ["odsold"] = 10684,
7264 ["olcir"] = 10686,
7265 ["ofcir"] = 10687,
7266 ["olt"] = 10688,
7267 ["ogt"] = 10689,
7268 ["cirscir"] = 10690,
7269 ["cirE"] = 10691,
7270 ["solb"] = 10692,
7271 ["bsolb"] = 10693,
7272 ["boxbox"] = 10697,
7273 ["trish"] = 10701,
7274 ["rtriltri"] = 10702,
7275 ["LeftTriangleBar"] = 10703,
7276 ["NotLeftTriangleBar"] = {10703, 824},
7277 ["NotRightTriangleBar"] = {10704, 824},
7278 ["RightTriangleBar"] = 10704,
7279 ["iinfin"] = 10716,
7280 ["infintie"] = 10717,
7281 ["nvinfin"] = 10718,
7282 ["eparsl"] = 10723,
7283 ["smeparsl"] = 10724,
7284 ["eqvparsl"] = 10725,
7285 ["blacklozenge"] = 10731,
7286 ["lozf"] = 10731,
7287 ["RuleDelayed"] = 10740,
7288 ["dsol"] = 10742,
7289 ["bigodot"] = 10752,
7290 ["xodot"] = 10752,

```

```

7291 ["bigoplus"] = 10753,
7292 ["xoplus"] = 10753,
7293 ["bigotimes"] = 10754,
7294 ["xotime"] = 10754,
7295 ["biguplus"] = 10756,
7296 ["xuplus"] = 10756,
7297 ["bigsqcup"] = 10758,
7298 ["xsqcup"] = 10758,
7299 ["iiiint"] = 10764,
7300 ["qint"] = 10764,
7301 ["fpartint"] = 10765,
7302 ["cirfnint"] = 10768,
7303 ["awint"] = 10769,
7304 ["rppolint"] = 10770,
7305 ["scpolint"] = 10771,
7306 ["npolint"] = 10772,
7307 ["pointint"] = 10773,
7308 ["quatint"] = 10774,
7309 ["intlarhk"] = 10775,
7310 ["pluscir"] = 10786,
7311 ["plusacir"] = 10787,
7312 ["simplus"] = 10788,
7313 ["plusdu"] = 10789,
7314 ["plussim"] = 10790,
7315 ["plustwo"] = 10791,
7316 ["mcomma"] = 10793,
7317 ["minusdu"] = 10794,
7318 ["loplus"] = 10797,
7319 ["roplus"] = 10798,
7320 ["Cross"] = 10799,
7321 ["timesd"] = 10800,
7322 ["timesbar"] = 10801,
7323 ["smashp"] = 10803,
7324 ["lotimes"] = 10804,
7325 ["rotimes"] = 10805,
7326 ["otimesas"] = 10806,
7327 ["Otimes"] = 10807,
7328 ["odiv"] = 10808,
7329 ["triplus"] = 10809,
7330 ["triminus"] = 10810,
7331 ["tritime"] = 10811,
7332 ["intprod"] = 10812,
7333 ["iproduct"] = 10812,
7334 ["amalg"] = 10815,
7335 ["capdot"] = 10816,
7336 ["ncup"] = 10818,
7337 ["ncap"] = 10819,

```

7338 ["capand"] = 10820,  
7339 ["cupor"] = 10821,  
7340 ["cupcap"] = 10822,  
7341 ["capcup"] = 10823,  
7342 ["cupbrcap"] = 10824,  
7343 ["capbrcup"] = 10825,  
7344 ["cupcup"] = 10826,  
7345 ["capcap"] = 10827,  
7346 ["ccups"] = 10828,  
7347 ["ccaps"] = 10829,  
7348 ["ccupssm"] = 10832,  
7349 ["And"] = 10835,  
7350 ["Or"] = 10836,  
7351 ["andand"] = 10837,  
7352 ["oror"] = 10838,  
7353 ["orslope"] = 10839,  
7354 ["andslope"] = 10840,  
7355 ["andv"] = 10842,  
7356 ["orv"] = 10843,  
7357 ["andd"] = 10844,  
7358 ["ord"] = 10845,  
7359 ["wedbar"] = 10847,  
7360 ["sdote"] = 10854,  
7361 ["simdot"] = 10858,  
7362 ["congdote"] = 10861,  
7363 ["ncongdot"] = {10861, 824},  
7364 ["easter"] = 10862,  
7365 ["apacir"] = 10863,  
7366 ["apE"] = 10864,  
7367 ["napE"] = {10864, 824},  
7368 ["eplus"] = 10865,  
7369 ["pluse"] = 10866,  
7370 ["Esim"] = 10867,  
7371 ["Colone"] = 10868,  
7372 ["Equal"] = 10869,  
7373 ["ddotseq"] = 10871,  
7374 ["eDDot"] = 10871,  
7375 ["equivDD"] = 10872,  
7376 ["ltcir"] = 10873,  
7377 ["gtcir"] = 10874,  
7378 ["ltquest"] = 10875,  
7379 ["gtquest"] = 10876,  
7380 ["LessSlantEqual"] = 10877,  
7381 ["NotLessSlantEqual"] = {10877, 824},  
7382 ["leqslant"] = 10877,  
7383 ["les"] = 10877,  
7384 ["nleqslant"] = {10877, 824},



```

7385 ["nles"] = {10877, 824},
7386 ["GreaterSlantEqual"] = 10878,
7387 ["NotGreaterSlantEqual"] = {10878, 824},
7388 ["geqslant"] = 10878,
7389 ["ges"] = 10878,
7390 ["ngeqslant"] = {10878, 824},
7391 ["nges"] = {10878, 824},
7392 ["lesdot"] = 10879,
7393 ["gesdot"] = 10880,
7394 ["lesdoto"] = 10881,
7395 ["gesdoto"] = 10882,
7396 ["lesdotor"] = 10883,
7397 ["gesdoto1"] = 10884,
7398 ["lap"] = 10885,
7399 ["lessapprox"] = 10885,
7400 ["gap"] = 10886,
7401 ["gtrapprox"] = 10886,
7402 ["lne"] = 10887,
7403 ["lneq"] = 10887,
7404 ["gne"] = 10888,
7405 ["gneq"] = 10888,
7406 ["lnap"] = 10889,
7407 ["lnapprox"] = 10889,
7408 ["gnap"] = 10890,
7409 ["gnapprox"] = 10890,
7410 ["lEg"] = 10891,
7411 ["lesseqqgtr"] = 10891,
7412 ["gEl"] = 10892,
7413 ["gtreqqless"] = 10892,
7414 ["lsime"] = 10893,
7415 ["gsime"] = 10894,
7416 ["lsimg"] = 10895,
7417 ["gsiml"] = 10896,
7418 ["lgE"] = 10897,
7419 ["glE"] = 10898,
7420 ["lesges"] = 10899,
7421 ["gesles"] = 10900,
7422 ["els"] = 10901,
7423 ["eqslantless"] = 10901,
7424 ["egs"] = 10902,
7425 ["eqslantgtr"] = 10902,
7426 ["elsdot"] = 10903,
7427 ["egsdot"] = 10904,
7428 ["el"] = 10905,
7429 ["eg"] = 10906,
7430 ["siml"] = 10909,
7431 ["simg"] = 10910,

```

```

7432 ["simlE"] = 10911,
7433 ["simgE"] = 10912,
7434 ["LessLess"] = 10913,
7435 ["NotNestedLessLess"] = {10913, 824},
7436 ["GreaterGreater"] = 10914,
7437 ["NotNestedGreaterGreater"] = {10914, 824},
7438 ["glj"] = 10916,
7439 ["gla"] = 10917,
7440 ["ltcc"] = 10918,
7441 ["gtcc"] = 10919,
7442 ["lescc"] = 10920,
7443 ["gescc"] = 10921,
7444 ["smt"] = 10922,
7445 ["lat"] = 10923,
7446 ["smte"] = 10924,
7447 ["smtes"] = {10924, 65024},
7448 ["late"] = 10925,
7449 ["lates"] = {10925, 65024},
7450 ["bumpE"] = 10926,
7451 ["NotPrecedesEqual"] = {10927, 824},
7452 ["PrecedesEqual"] = 10927,
7453 ["npre"] = {10927, 824},
7454 ["npreceq"] = {10927, 824},
7455 ["pre"] = 10927,
7456 ["preceq"] = 10927,
7457 ["NotSucceedsEqual"] = {10928, 824},
7458 ["SucceedsEqual"] = 10928,
7459 ["nsce"] = {10928, 824},
7460 ["nsucceq"] = {10928, 824},
7461 ["sce"] = 10928,
7462 ["succeq"] = 10928,
7463 ["prE"] = 10931,
7464 ["scE"] = 10932,
7465 ["precneqq"] = 10933,
7466 ["prnE"] = 10933,
7467 ["scnE"] = 10934,
7468 ["succneqq"] = 10934,
7469 ["prap"] = 10935,
7470 ["precapprox"] = 10935,
7471 ["scap"] = 10936,
7472 ["succapprox"] = 10936,
7473 ["precnapprox"] = 10937,
7474 ["prnap"] = 10937,
7475 ["scnap"] = 10938,
7476 ["succnapprox"] = 10938,
7477 ["Pr"] = 10939,
7478 ["Sc"] = 10940,

```

7479 ["subdot"] = 10941,  
 7480 ["supdot"] = 10942,  
 7481 ["subplus"] = 10943,  
 7482 ["supplus"] = 10944,  
 7483 ["submult"] = 10945,  
 7484 ["supmult"] = 10946,  
 7485 ["subedot"] = 10947,  
 7486 ["supedot"] = 10948,  
 7487 ["nsubE"] = {10949, 824},  
 7488 ["nsubseteqq"] = {10949, 824},  
 7489 ["subE"] = 10949,  
 7490 ["subseteqq"] = 10949,  
 7491 ["nsupE"] = {10950, 824},  
 7492 ["nsupseteqq"] = {10950, 824},  
 7493 ["supE"] = 10950,  
 7494 ["supseteqq"] = 10950,  
 7495 ["subsim"] = 10951,  
 7496 ["supsim"] = 10952,  
 7497 ["subnE"] = 10955,  
 7498 ["subsetneqq"] = 10955,  
 7499 ["varsubsetneqq"] = {10955, 65024},  
 7500 ["vsubnE"] = {10955, 65024},  
 7501 ["supnE"] = 10956,  
 7502 ["supsetneqq"] = 10956,  
 7503 ["varsupsetneqq"] = {10956, 65024},  
 7504 ["vsupnE"] = {10956, 65024},  
 7505 ["csub"] = 10959,  
 7506 ["csup"] = 10960,  
 7507 ["csube"] = 10961,  
 7508 ["csupe"] = 10962,  
 7509 ["subsup"] = 10963,  
 7510 ["supsub"] = 10964,  
 7511 ["subsub"] = 10965,  
 7512 ["supsup"] = 10966,  
 7513 ["suphsub"] = 10967,  
 7514 ["supdsub"] = 10968,  
 7515 ["forkv"] = 10969,  
 7516 ["topfork"] = 10970,  
 7517 ["mlcp"] = 10971,  
 7518 ["Dashv"] = 10980,  
 7519 ["DoubleLeftTee"] = 10980,  
 7520 ["Vdashl"] = 10982,  
 7521 ["Barv"] = 10983,  
 7522 ["vBar"] = 10984,  
 7523 ["vBarv"] = 10985,  
 7524 ["Vbar"] = 10987,  
 7525 ["Not"] = 10988,

```

7526 ["bNot"] = 10989,
7527 ["rnmid"] = 10990,
7528 ["cirmid"] = 10991,
7529 ["midcir"] = 10992,
7530 ["topcir"] = 10993,
7531 ["nhpar"] = 10994,
7532 ["parsim"] = 10995,
7533 ["nparsl"] = {11005, 8421},
7534 ["parsl"] = 11005,
7535 ["fflig"] = 64256,
7536 ["filig"] = 64257,
7537 ["fllig"] = 64258,
7538 ["ffilig"] = 64259,
7539 ["ffllig"] = 64260,
7540 ["Ascr"] = 119964,
7541 ["Cscr"] = 119966,
7542 ["Dscr"] = 119967,
7543 ["Gscr"] = 119970,
7544 ["Jscr"] = 119973,
7545 ["Kscr"] = 119974,
7546 ["Nscr"] = 119977,
7547 ["Oscr"] = 119978,
7548 ["Pscr"] = 119979,
7549 ["Qscr"] = 119980,
7550 ["Sscr"] = 119982,
7551 ["Tscr"] = 119983,
7552 ["Uscr"] = 119984,
7553 ["Vscr"] = 119985,
7554 ["Wscr"] = 119986,
7555 ["Xscr"] = 119987,
7556 ["Yscr"] = 119988,
7557 ["Zscr"] = 119989,
7558 ["ascr"] = 119990,
7559 ["bscr"] = 119991,
7560 ["cscr"] = 119992,
7561 ["dscr"] = 119993,
7562 ["fscr"] = 119995,
7563 ["hscr"] = 119997,
7564 ["iscr"] = 119998,
7565 ["jscr"] = 119999,
7566 ["kscr"] = 120000,
7567 ["lscr"] = 120001,
7568 ["mscr"] = 120002,
7569 ["nscr"] = 120003,
7570 ["pscr"] = 120005,
7571 ["qscr"] = 120006,
7572 ["rscr"] = 120007,

```

```

7573 ["sscr"] = 120008,
7574 ["tscr"] = 120009,
7575 ["uscr"] = 120010,
7576 ["vscr"] = 120011,
7577 ["wscr"] = 120012,
7578 ["xscr"] = 120013,
7579 ["yscr"] = 120014,
7580 ["zscr"] = 120015,
7581 ["Afr"] = 120068,
7582 ["Bfr"] = 120069,
7583 ["Dfr"] = 120071,
7584 ["Efr"] = 120072,
7585 ["Ffr"] = 120073,
7586 ["Gfr"] = 120074,
7587 ["Jfr"] = 120077,
7588 ["Kfr"] = 120078,
7589 ["Lfr"] = 120079,
7590 ["Mfr"] = 120080,
7591 ["Nfr"] = 120081,
7592 ["Ofr"] = 120082,
7593 ["Pfr"] = 120083,
7594 ["Qfr"] = 120084,
7595 ["Sfr"] = 120086,
7596 ["Tfr"] = 120087,
7597 ["Ufr"] = 120088,
7598 ["Vfr"] = 120089,
7599 ["Wfr"] = 120090,
7600 ["Xfr"] = 120091,
7601 ["Yfr"] = 120092,
7602 ["afr"] = 120094,
7603 ["bfr"] = 120095,
7604 ["cfr"] = 120096,
7605 ["dfr"] = 120097,
7606 ["efr"] = 120098,
7607 ["ffr"] = 120099,
7608 ["gfr"] = 120100,
7609 ["hfr"] = 120101,
7610 ["ifr"] = 120102,
7611 ["jfr"] = 120103,
7612 ["kfr"] = 120104,
7613 ["lfr"] = 120105,
7614 ["mfr"] = 120106,
7615 ["nfr"] = 120107,
7616 ["ofr"] = 120108,
7617 ["pfr"] = 120109,
7618 ["qfr"] = 120110,
7619 ["rfr"] = 120111,

```

```

7620 ["sfr"] = 120112,
7621 ["tfr"] = 120113,
7622 ["ufr"] = 120114,
7623 ["vfr"] = 120115,
7624 ["wfr"] = 120116,
7625 ["xfr"] = 120117,
7626 ["yfr"] = 120118,
7627 ["zfr"] = 120119,
7628 ["Aopf"] = 120120,
7629 ["Bopf"] = 120121,
7630 ["Dopf"] = 120123,
7631 ["Eopf"] = 120124,
7632 ["Fopf"] = 120125,
7633 ["Gopf"] = 120126,
7634 ["Iopf"] = 120128,
7635 ["Jopf"] = 120129,
7636 ["Kopf"] = 120130,
7637 ["Lopf"] = 120131,
7638 ["Mopf"] = 120132,
7639 ["Oopf"] = 120134,
7640 ["Sopf"] = 120138,
7641 ["Topf"] = 120139,
7642 ["Uopf"] = 120140,
7643 ["Vopf"] = 120141,
7644 ["Wopf"] = 120142,
7645 ["Xopf"] = 120143,
7646 ["Yopf"] = 120144,
7647 ["aopf"] = 120146,
7648 ["bopf"] = 120147,
7649 ["copf"] = 120148,
7650 ["dopf"] = 120149,
7651 ["eopf"] = 120150,
7652 ["fopf"] = 120151,
7653 ["gopf"] = 120152,
7654 ["hopf"] = 120153,
7655 ["iopf"] = 120154,
7656 ["jopf"] = 120155,
7657 ["kopf"] = 120156,
7658 ["lopf"] = 120157,
7659 ["mopf"] = 120158,
7660 ["nopf"] = 120159,
7661 ["oopf"] = 120160,
7662 ["popf"] = 120161,
7663 ["qopf"] = 120162,
7664 ["ropf"] = 120163,
7665 ["sopf"] = 120164,
7666 ["topf"] = 120165,

```

```

7667  ["uopf"] = 120166,
7668  ["vopf"] = 120167,
7669  ["wopf"] = 120168,
7670  ["xopf"] = 120169,
7671  ["yopf"] = 120170,
7672  ["zopf"] = 120171,
7673  }

```

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

```

7674 function entities.dec_entity(s)
7675   local n = tonumber(s)
7676   if n == nil then
7677     return "&#" .. s .. ";" -- fallback for unknown entities
7678   end
7679   return utf8.char(n)
7680 end

```

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

```

7681 function entities.hex_entity(s)
7682   local n = tonumber("0x"..s)
7683   if n == nil then
7684     return "&#x" .. s .. ";" -- fallback for unknown entities
7685   end
7686   return utf8.char(n)
7687 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.

```

7688 function entities.hex_entity_with_x_char(x, s)
7689   local n = tonumber("0x"..s)
7690   if n == nil then
7691     return "&#" .. x .. s .. ";" -- fallback for unknown entities
7692   end
7693   return utf8.char(n)
7694 end

```

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

```

7695 function entities.char_entity(s)
7696   local code_points = character_entities[s]
7697   if code_points == nil then
7698     return "&" .. s .. ";"
7699   end
7700   if type(code_points) ~= 'table' then
7701     code_points = {code_points}

```

```

7702     end
7703     local char_table = {}
7704     for _, code_point in ipairs(code_points) do
7705         table.insert(char_table, utf8.char(code_point))
7706     end
7707     return table.concat(char_table)
7708 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.

```

7709 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.

```

7710 local parsers
7711 function M.writer.new(options)
7712     local self = {}

```

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

```

7713     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.

```

7714     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.

```

7715     local slice_specifiers = {}
7716     for specifier in options.slice:gmatch("[^%s]+") do

```



```

7717     table.insert(slice_specifiers, specifier)
7718 end
7719
7720 if #slice_specifiers == 2 then
7721     self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
7722     local slice_begin_type = self.slice_begin:sub(1, 1)
7723     if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
7724         self.slice_begin = "^" .. self.slice_begin
7725     end
7726     local slice_end_type = self.slice_end:sub(1, 1)
7727     if slice_end_type ~= "^" and slice_end_type ~= "$" then
7728         self.slice_end = "$" .. self.slice_end
7729     end
7730 elseif #slice_specifiers == 1 then
7731     self.slice_begin = "^" .. slice_specifiers[1]
7732     self.slice_end = "$" .. slice_specifiers[1]
7733 end
7734
7735 self.slice_begin_type = self.slice_begin:sub(1, 1)
7736 self.slice_begin_identifier = self.slice_begin:sub(2) or ""
7737 self.slice_end_type = self.slice_end:sub(1, 1)
7738 self.slice_end_identifier = self.slice_end:sub(2) or ""
7739
7740 if self.slice_begin == "^" and self.slice_end ~= "^" then
7741     self.is_writing = true
7742 else
7743     self.is_writing = false
7744 end

```

### 3.1.4.2 Basic Formatter Variables and Functions

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

```

7745     self.space = " "

```

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

```

7746     self.nbsp = "\\markdownRendererNbsp{}"

```

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

```

7747     function self.plain(s)
7748         return s
7749     end

```

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

```

7750     function self.paragraph(s)
7751         if not self.is_writing then return "" end
7752         return s
7753     end

```

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

```
7754 self.interblocksep_text = "\\markdownRendererInterblockSeparator\n{}"
7755 function self.interblocksep()
7756     if not self.is_writing then return "" end
7757     return self.interblocksep_text
7758 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.

```
7759 self.paragraphsep_text = "\\markdownRendererParagraphSeparator\n{}"
7760 function self.paragraphsep()
7761     if not self.is_writing then return "" end
7762     return self.paragraphsep_text
7763 end
```

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

```
7764 self.undosep_text = "\\markdownRendererUndoSeparator\n{}"
7765 function self.undosep()
7766     if not self.is_writing then return "" end
7767     return self.undosep_text
7768 end
```

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

```
7769 self.soft_line_break = function()
7770     if self.flatten_inlines then return "\n" end
7771     return "\\markdownRendererSoftLineBreak\n{}"
7772 end
```

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

```
7773 self.hard_line_break = function()
7774     if self.flatten_inlines then return "\n" end
7775     return "\\markdownRendererHardLineBreak\n{}"
7776 end
```

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

```
7777 self.ellipsis = "\\markdownRendererEllipsis{}"
```

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

```
7778 function self.thematic_break()
7779     if not self.is_writing then return "" end
7780     return "\\markdownRendererThematicBreak{}"
7781 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.

```

7782 self.escaped_uri_chars = {
7783     ["{"] = "\\markdownRendererLeftBrace{}",
7784     ["}"] = "\\markdownRendererRightBrace{}",
7785     ["\\"] = "\\markdownRendererBackslash{}",
7786     ["\r"] = " ",
7787     ["\n"] = " ",
7788 }
7789 self.escaped_minimal_strings = {
7790     ["^"] = "\\markdownRendererCircumflex",
7791     ["."] = "\\markdownRendererCircumflex ",
7792     ["☒"] = "\\markdownRendererTickedBox{}",
7793     ["☐"] = "\\markdownRendererHalfTickedBox{}",
7794     ["□"] = "\\markdownRendererUntickedBox{}",
7795     [entities.hex_entity('FFFD')]
7796     = "\\markdownRendererReplacementCharacter{}",
7797 }

```

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

```

7798 self.escaped_strings = util.table_copy(self.escaped_minimal_strings)
7799 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  $\text{Con}\TeX$ t) that need to be escaped in typeset content.

```

7800 self.escaped_chars = {
7801     ["{"] = "\\markdownRendererLeftBrace{}",
7802     ["}"] = "\\markdownRendererRightBrace{}",
7803     ["%"] = "\\markdownRendererPercentSign{}",
7804     ["\\"] = "\\markdownRendererBackslash{}",
7805     ["#"] = "\\markdownRendererHash{}",
7806     ["$"] = "\\markdownRendererDollarSign{}",
7807     ["&"] = "\\markdownRendererAmpersand{}",
7808     ["_"] = "\\markdownRendererUnderscore{}",
7809     ["^"] = "\\markdownRendererCircumflex{}",
7810     ["~"] = "\\markdownRendererTilde{}",
7811     ["|"] = "\\markdownRendererPipe{}",
7812     [entities.hex_entity('0000')]
7813     = "\\markdownRendererReplacementCharacter{}",
7814 }

```

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

```

7815 local function create_escaper(char_escapes, string_escapes)
7816     local escape = util.escaper(char_escapes, string_escapes)
7817     return function(s)
7818         if self.flatten_inlines then return s end
7819         return escape(s)
7820     end
7821 end
7822 local escape_typographic_text = create_escaper(
7823     self.escaped_chars, self.escaped_strings)
7824 local escape_programmatic_text = create_escaper(
7825     self.escaped_uri_chars, self.escaped_minimal_strings)
7826 local escape_minimal = create_escaper(
7827     {}, 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.

```

7828 self.escape = escape_typographic_text
7829 self.math = escape_minimal
7830 if options.hybrid then
7831     self.identifier = escape_minimal
7832     self.string = escape_minimal
7833     self.uri = escape_minimal
7834     self.infostring = escape_minimal
7835 else
7836     self.identifier = escape_programmatic_text
7837     self.string = escape_typographic_text
7838     self.uri = escape_programmatic_text
7839     self.infostring = escape_programmatic_text
7840 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.

```

7841 function self.warning(t, m)
7842     return {"\\markdownRendererWarning{", self.escape(t), "}{" ,
7843         escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
7844         escape_minimal(m or ""), "}"}
7845 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.

```

7846 function self.error(t, m)
7847   return {"\\markdownRendererError{" , self.escape(t), "}{" ,
7848           escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
7849           escape_minimal(m or ""), "}"}
7850 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.

```

7851 function self.code(s, attributes)
7852   if self.flatten_inlines then return s end
7853   local buf = {}
7854   if attributes ~= nil then
7855     table.insert(buf,
7856                 "\\markdownRendererCodeSpanAttributeContextBegin\n")
7857     table.insert(buf, self.attributes(attributes))
7858   end
7859   table.insert(buf,
7860               {"\\markdownRendererCodeSpan{" , self.escape(s), "}"}
7861   if attributes ~= nil then
7862     table.insert(buf,
7863                 "\\markdownRendererCodeSpanAttributeContextEnd{")
7864   end
7865   return buf
7866 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.

```

7867 function self.link(lab, src, tit, attributes)
7868   if self.flatten_inlines then return lab end
7869   local buf = {}
7870   if attributes ~= nil then
7871     table.insert(buf,
7872                 "\\markdownRendererLinkAttributeContextBegin\n")
7873     table.insert(buf, self.attributes(attributes))
7874   end
7875   table.insert(buf, {"\\markdownRendererLink{" , lab, "}"} ,
7876               {"", self.escape(src), "}"} ,
7877               {"", self.uri(src), "}"} ,
7878               {"", self.string(tit or ""), "}"}
7879   if attributes ~= nil then

```

```

7880         table.insert(buf,
7881             "\\markdownRendererLinkAttributeContextEnd{ }")
7882     end
7883     return buf
7884 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.

```

7885 function self.image(lab, src, tit, attributes)
7886     if self.flatten_inlines then return lab end
7887     local buf = {}
7888     if attributes ~= nil then
7889         table.insert(buf,
7890             "\\markdownRendererImageAttributeContextBegin\n")
7891         table.insert(buf, self.attributes(attributes))
7892     end
7893     table.insert(buf, {"\\markdownRendererImage{",lab,""},
7894         {"",self.string(src),""},
7895         {"",self.uri(src),""},
7896         {"",tit,""})
7897     if attributes ~= nil then
7898         table.insert(buf,
7899             "\\markdownRendererImageAttributeContextEnd{ }")
7900     end
7901     return buf
7902 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.

```

7903 function self.bulletlist(items,tight)
7904     if not self.is_writing then return "" end
7905     local buffer = {}
7906     for _,item in ipairs(items) do
7907         if item ~= "" then
7908             buffer[#buffer + 1] = self.bulletitem(item)
7909         end
7910     end
7911     local contents = util.intersperse(buffer,"\n")
7912     if tight and options.tightLists then
7913         return {"\\markdownRendererULBeginTight\n",contents,
7914             "\n\\markdownRendererULEndTight "}

```

```

7915     else
7916         return {"\\markdownRendererUlBegin\n",contents,
7917             "\n\\markdownRendererUlEnd "}
7918     end
7919 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.

```

7920 function self.bulletitem(s)
7921     return {"\\markdownRendererUlItem ",s,
7922         "\\markdownRendererUlItemEnd "}
7923 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.

```

7924 function self.orderedlist(items,tight,startnum)
7925     if not self.is_writing then return "" end
7926     local buffer = {}
7927     local num = startnum
7928     for _,item in ipairs(items) do
7929         if item ~= "" then
7930             buffer[#buffer + 1] = self.ordereditem(item,num)
7931         end
7932         if num ~= nil and item ~= "" then
7933             num = num + 1
7934         end
7935     end
7936     local contents = util.intersperse(buffer,"\n")
7937     if tight and options.tightLists then
7938         return {"\\markdownRendererOlBeginTight\n",contents,
7939             "\n\\markdownRendererOlEndTight "}
7940     else
7941         return {"\\markdownRendererOlBegin\n",contents,
7942             "\n\\markdownRendererOlEnd "}
7943     end
7944 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.

```

7945 function self.ordereditem(s,num)
7946     if num ~= nil then
7947         return {"\\markdownRendererOlItemWithNumber{",num,"}",s,
7948             "\\markdownRendererOlItemEnd "}
7949     else
7950         return {"\\markdownRendererOlItem ",s,

```

```

7951             "\\markdownRendererOlItemEnd "}
7952     end
7953 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.

```

7954 function self.inline_html_comment(contents)
7955     if self.flatten_inlines then return contents end
7956     return {"\\markdownRendererInlineHtmlComment{",contents,""}
7957 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.

```

7958 function self.inline_html_tag(contents)
7959     if self.flatten_inlines then return contents end
7960     return {"\\markdownRendererInlineHtmlTag{",
7961             self.string(contents),""}
7962 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.

```

7963 function self.block_html_element(s)
7964     if not self.is_writing then return "" end
7965     local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
7966     return {"\\markdownRendererInputBlockHtmlElement{",name,""}
7967 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.

```

7968 function self.emphasis(s)
7969     if self.flatten_inlines then return s end
7970     return {"\\markdownRendererEmphasis{",s,""}
7971 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.

```

7972 function self.strong(s)
7973     if self.flatten_inlines then return s end

```



```

7974     return {"\\markdownRendererStrongEmphasis{" ,s,""}
7975 end

```

#### 3.1.4.12 Formatter of Tickboxes

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

```

7976 function self.tickbox(f)
7977     if f == 1.0 then
7978         return "☒ "
7979     elseif f == 0.0 then
7980         return "☐ "
7981     else
7982         return "◻ "
7983     end
7984 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.

```

7985 function self.blockquote(s)
7986     if not self.is_writing then return "" end
7987     return {"\\markdownRendererBlockQuoteBegin\n",s,
7988         "\\markdownRendererBlockQuoteEnd "}
7989 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.

```

7990 function self.verbatim(s)
7991     if not self.is_writing then return "" end
7992     s = s:gsub("\n$", "")
7993     local name = util.cache_verbatim(options.cacheDir, s)
7994     return {"\\markdownRendererInputVerbatim{" ,name,""}
7995 end

```

#### 3.1.4.15 Formatter of Documents

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

```

7996 function self.document(d)
7997     local buf = {"\\markdownRendererDocumentBegin\n"}
7998
7999     -- warn against the `hybrid` option
8000     if options.hybrid then

```

```

8001     local text = "The `hybrid` option has been soft-deprecated."
8002     local more = "Consider using one of the following better options "
8003         .. "for mixing TeX and markdown: `contentBlocks`, "
8004         .. "`rawAttribute`, `texComments`, `texMathDollars`, "
8005         .. "`texMathSingleBackslash`, and "
8006         .. "`texMathDoubleBackslash`. "
8007         .. "For more information, see the user manual at "
8008         .. "<https://witiko.github.io/markdown/>."
8009     table.insert(buf, self.warning(text, more))
8010 end
8011
8012 -- insert the text of the document
8013 table.insert(buf, d)
8014
8015 -- pop all attributes
8016 table.insert(buf, self.pop_attributes())
8017
8018 table.insert(buf, "\\markdownRendererDocumentEnd")
8019
8020 return buf
8021 end

```

### 3.1.4.16 Formatter of Attributes

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

```

8022 local seen_identifiers = {}
8023 local key_value_regex = "([~= ]+)%s*=%s*(.*)"
8024 local function normalize_attributes(attributes, auto_identifiers)
8025     -- normalize attributes
8026     local normalized_attributes = {}
8027     local has_explicit_identifiers = false
8028     local key, value
8029     for _, attribute in ipairs(attributes or {}) do
8030         if attribute:sub(1, 1) == "#" then
8031             table.insert(normalized_attributes, attribute)
8032             has_explicit_identifiers = true
8033             seen_identifiers[attribute:sub(2)] = true
8034         elseif attribute:sub(1, 1) == "." then
8035             table.insert(normalized_attributes, attribute)
8036         else
8037             key, value = attribute:match(key_value_regex)
8038             if key:lower() == "id" then
8039                 table.insert(normalized_attributes, "#" .. value)
8040             elseif key:lower() == "class" then
8041                 local classes = {}
8042                 for class in value:gmatch("%S+") do

```

```

8043         table.insert(classes, class)
8044     end
8045     table.sort(classes)
8046     for _, class in ipairs(classes) do
8047         table.insert(normalized_attributes, "." .. class)
8048     end
8049     else
8050         table.insert(normalized_attributes, attribute)
8051     end
8052 end
8053 end
8054
8055 -- if no explicit identifiers exist, add auto identifiers
8056 if not has_explicit_identifiers and auto_identifiers ~= nil then
8057     local seen_auto_identifiers = {}
8058     for _, auto_identifier in ipairs(auto_identifiers) do
8059         if seen_auto_identifiers[auto_identifier] == nil then
8060             seen_auto_identifiers[auto_identifier] = true
8061             if seen_identifiers[auto_identifier] == nil then
8062                 seen_identifiers[auto_identifier] = true
8063                 table.insert(normalized_attributes,
8064                     "#" .. auto_identifier)
8065             else
8066                 local auto_identifier_number = 1
8067                 while true do
8068                     local numbered_auto_identifier = auto_identifier .. "-"
8069   .. auto_identifier_number
8070                     if seen_identifiers[numbered_auto_identifier] == nil then
8071                         seen_identifiers[numbered_auto_identifier] = true
8072                         table.insert(normalized_attributes,
8073                             "#" .. numbered_auto_identifier)
8074                     break
8075                 end
8076                 auto_identifier_number = auto_identifier_number + 1
8077             end
8078         end
8079     end
8080 end
8081 end
8082
8083 -- sort and deduplicate normalized attributes
8084 table.sort(normalized_attributes)
8085 local seen_normalized_attributes = {}
8086 local deduplicated_normalized_attributes = {}
8087 for _, attribute in ipairs(normalized_attributes) do
8088     if seen_normalized_attributes[attribute] == nil then
8089         seen_normalized_attributes[attribute] = true

```

```

8090         table.insert(deduplicated_normalized_attributes, attribute)
8091     end
8092 end
8093
8094     return deduplicated_normalized_attributes
8095 end
8096
8097 function self.attributes(attributes, should_normalize_attributes)
8098     local normalized_attributes
8099     if should_normalize_attributes == false then
8100         normalized_attributes = attributes
8101     else
8102         normalized_attributes = normalize_attributes(attributes)
8103     end
8104
8105     local buf = {}
8106     local key, value
8107     for _, attribute in ipairs(normalized_attributes) do
8108         if attribute:sub(1, 1) == "#" then
8109             table.insert(buf, {"\\markdownRendererAttributeIdentifier{" ,
8110                 attribute:sub(2), "}"})
8111         elseif attribute:sub(1, 1) == "." then
8112             table.insert(buf, {"\\markdownRendererAttributeName{" ,
8113                 attribute:sub(2), "}"})
8114         else
8115             key, value = attribute:match(key_value_regex)
8116             table.insert(buf, {"\\markdownRendererAttributeKeyValue{" ,
8117                 key, "{" , value, "}"})
8118         end
8119     end
8120
8121     return buf
8122 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.

```

8123     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`.

```

8124     self.attribute_type_levels = {}
8125     setmetatable(self.attribute_type_levels,
8126         { __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`.

```

8127 local function apply_attributes()
8128   local buf = {}
8129   for i = 1, #self.active_attributes do
8130     local start_output = self.active_attributes[i][3]
8131     if start_output ~= nil then
8132       table.insert(buf, start_output)
8133     end
8134   end
8135   return buf
8136 end
8137
8138 local function tear_down_attributes()
8139   local buf = {}
8140   for i = #self.active_attributes, 1, -1 do
8141     local end_output = self.active_attributes[i][4]
8142     if end_output ~= nil then
8143       table.insert(buf, end_output)
8144     end
8145   end
8146   return buf
8147 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.

```

8148 function self.push_attributes(attribute_type, attributes,
8149                               start_output, end_output)
8150   local attribute_type_level
8151     = self.attribute_type_levels[attribute_type]
8152   self.attribute_type_levels[attribute_type]
8153     = attribute_type_level + 1
8154
8155   -- index attributes in a hash table for easy lookup
8156   attributes = attributes or {}
8157   for i = 1, #attributes do
8158     attributes[attributes[i]] = true
8159   end
8160
8161   local buf = {}
8162   -- handle slicing
8163   if attributes["#" .. self.slice_end_identifier] ~= nil and
8164     self.slice_end_type == "^" then

```

```

8165         if self.is_writing then
8166             table.insert(buf, self.undosep())
8167             table.insert(buf, tear_down_attributes())
8168         end
8169         self.is_writing = false
8170     end
8171     if attributes["#" .. self.slice_begin_identifier] ~= nil and
8172        self.slice_begin_type == "^" then
8173         table.insert(buf, apply_attributes())
8174         self.is_writing = true
8175     end
8176     if self.is_writing and start_output ~= nil then
8177         table.insert(buf, start_output)
8178     end
8179     table.insert(self.active_attributes,
8180                 {attribute_type, attributes,
8181                  start_output, end_output})
8182     return buf
8183 end
8184

```

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`).

```

8185 function self.pop_attributes(attribute_type)
8186     local buf = {}
8187     -- pop attributes until we find attributes of correct type
8188     -- or until no attributes remain
8189     local current_attribute_type = false
8190     while current_attribute_type ~= attribute_type and
8191        #self.active_attributes > 0 do
8192         local attributes, _, end_output
8193         current_attribute_type, attributes, _, end_output = table.unpack(
8194             self.active_attributes[#self.active_attributes])
8195         local attribute_type_level
8196         = self.attribute_type_levels[current_attribute_type]
8197         self.attribute_type_levels[current_attribute_type]
8198         = attribute_type_level - 1
8199         if self.is_writing and end_output ~= nil then
8200             table.insert(buf, end_output)
8201         end
8202         table.remove(self.active_attributes, #self.active_attributes)
8203         -- handle slicing
8204         if attributes["#" .. self.slice_end_identifier] ~= nil

```

```

8205         and self.slice_end_type == "$" then
8206         if self.is_writing then
8207             table.insert(buf, self.undosep())
8208             table.insert(buf, tear_down_attributes())
8209         end
8210         self.is_writing = false
8211     end
8212     if attributes["#" .. self.slice_begin_identfier] ~= nil and
8213         self.slice_begin_type == "$" then
8214         self.is_writing = true
8215         table.insert(buf, apply_attributes())
8216     end
8217 end
8218 return buf
8219 end

```

### 3.1.4.18 Automatically Generated Identifiers for Headings

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

```

8220 local function create_auto_identifier(s)
8221     local buffer = {}
8222     local prev_space = false
8223     local letter_found = false
8224     local normalized_s = s
8225     if not options.unicodeNormalization
8226         or options.unicodeNormalizationForm ~= "nfc" then
8227         normalized_s = util.normalize(normalized_s, "nfc")
8228     end
8229
8230     for _, code in utf8.codes(normalized_s) do
8231         local char = utf8.char(code)
8232
8233         -- Remove everything up to the first letter.
8234         if not letter_found then
8235             local is_letter = lpeg.match(
8236                 parsers.unicode.alpha,
8237                 char
8238             )
8239             if is_letter then
8240                 letter_found = true
8241             else
8242                 goto continue
8243             end
8244         end
8245
8246         -- Remove all non-alphanumeric characters, except underscores,

```

```

8247     -- hyphens, and periods.
8248     if not lpeg.match(
8249         ( parsers.underscore
8250         + parsers.dash
8251         + parsers.period
8252         + parsers.unicode.word
8253         + #parsers.unicode.whitespace ),
8254         char
8255     ) then
8256         goto continue
8257     end
8258
8259     -- Replace all spaces and newlines with hyphens.
8260     if lpeg.match(
8261         ( parsers.newline
8262         + #parsers.unicode.whitespace ),
8263         char
8264     ) then
8265         char = "-"
8266         if prev_space then
8267             goto continue
8268         else
8269             prev_space = true
8270         end
8271     else
8272         -- Case-fold all alphabetic characters.
8273         local form = nil
8274         if options.unicodeNormalization then
8275             form = options.unicodeNormalizationForm
8276         end
8277         char = util.casefold(char, form)
8278         prev_space = false
8279     end
8280
8281     table.insert(buffer, char)
8282
8283     ::continue::
8284 end
8285
8286 if prev_space then
8287     table.remove(buffer)
8288 end
8289
8290 local identifier = #buffer == 0 and "section"
8291                  or table.concat(buffer, "")
8292 return identifier
8293 end

```



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

```
8294 local function create_gfm_auto_identifier(s)
8295   local buffer = {}
8296   local prev_space = false
8297   local letter_found = false
8298   local normalized_s = s
8299   if not options.unicodeNormalization
8300     or options.unicodeNormalizationForm ~= "nfc" then
8301     normalized_s = util.normalize(normalized_s, "nfc")
8302   end
8303
8304   for _, code in utf8.codes(normalized_s) do
8305     local char = utf8.char(code)
8306
8307     -- Remove everything up to the first non-space.
8308     if not letter_found then
8309       local is_letter = not lpeg.match(
8310         #parsers.unicode.whitespace,
8311         char
8312       )
8313       if is_letter then
8314         letter_found = true
8315       else
8316         goto continue
8317       end
8318     end
8319
8320     -- Remove all non-alphanumeric characters, except underscores
8321     -- and hyphens.
8322     if not lpeg.match(
8323       ( parsers.underscore
8324         + parsers.dash
8325         + parsers.unicode.word
8326         + #parsers.unicode.whitespace ),
8327       char
8328     ) then
8329       prev_space = false
8330       goto continue
8331     end
8332
8333     -- Replace all spaces and newlines with hyphens.
8334     if lpeg.match(
8335       ( parsers.newline
8336         + #parsers.unicode.whitespace ),
8337       char
8338     ) then
```

```

8339     char = "-"
8340     if prev_space then
8341         goto continue
8342     else
8343         prev_space = true
8344     end
8345 else
8346     -- Case-fold all alphabetic characters.
8347     local form = nil
8348     if options.unicodeNormalization then
8349         form = options.unicodeNormalizationForm
8350     end
8351     char = util.casefold(char, form)
8352     prev_space = false
8353 end
8354
8355 table.insert(buffer, char)
8356
8357 ::continue::
8358 end
8359
8360 if prev_space then
8361     table.remove(buffer)
8362 end
8363
8364 local identifier = #buffer == 0 and "section"
8365                  or table.concat(buffer, "")
8366 return identifier
8367 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.

```

8368 self.secbegin_text = "\\markdownRendererSectionBegin\n"
8369 self.seccend_text = "\n\\markdownRendererSectionEnd "
8370 function self.heading(s, level, attributes)
8371     local buf = {}
8372     local flat_text, inlines = table.unpack(s)
8373
8374     -- push empty attributes for implied sections
8375     while self.attribute_type_levels["heading"] < level - 1 do
8376         table.insert(buf,
8377             self.push_attributes("heading",
8378                 nil,
8379                 self.secbegin_text,
8380                 self.seccend_text))

```

```

8381     end
8382
8383     -- pop attributes for sections that have ended
8384     while self.attribute_type_levels["heading"] >= level do
8385         table.insert(buf, self.pop_attributes("heading"))
8386     end
8387
8388     -- construct attributes for the new section
8389     local auto_identifiers = {}
8390     if self.options.autoIdentifiers then
8391         table.insert(auto_identifiers, create_auto_identifier(flat_text))
8392     end
8393     if self.options.gfmAutoIdentifiers then
8394         table.insert(auto_identifiers,
8395             create_gfm_auto_identifier(flat_text))
8396     end
8397     local normalized_attributes = normalize_attributes(attributes,
8398   auto_identifiers)
8399
8400     -- push attributes for the new section
8401     local start_output = {}
8402     local end_output = {}
8403     table.insert(start_output, self.secbegin_text)
8404     table.insert(end_output, self.secend_text)
8405
8406     table.insert(buf, self.push_attributes("heading",
8407   normalized_attributes,
8408   start_output,
8409   end_output))
8410     assert(self.attribute_type_levels["heading"] == level)
8411
8412     -- render the heading and its attributes
8413     if self.is_writing and #normalized_attributes > 0 then
8414         table.insert(buf,
8415             "\\markdownRendererHeaderAttributeContextBegin\n")
8416         table.insert(buf, self.attributes(normalized_attributes, false))
8417     end
8418
8419     local cmd
8420     level = level + options.shiftHeadings
8421     if level <= 1 then
8422         cmd = "\\markdownRendererHeadingOne"
8423     elseif level == 2 then
8424         cmd = "\\markdownRendererHeadingTwo"
8425     elseif level == 3 then
8426         cmd = "\\markdownRendererHeadingThree"
8427     elseif level == 4 then

```

```

8428     cmd = "\\markdownRendererHeadingFour"
8429 elseif level == 5 then
8430     cmd = "\\markdownRendererHeadingFive"
8431 elseif level >= 6 then
8432     cmd = "\\markdownRendererHeadingSix"
8433 else
8434     cmd = ""
8435 end
8436 if self.is_writing then
8437     table.insert(buf, {cmd, "{", inlines, "}"})
8438 end
8439
8440 if self.is_writing and #normalized_attributes > 0 then
8441     table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd{")
8442 end
8443
8444 return buf
8445 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.

```

8446 function self.get_state()
8447     return {
8448         is_writing=self.is_writing,
8449         flatten_inlines=self.flatten_inlines,
8450         active_attributes={table.unpack(self.active_attributes)},
8451     }
8452 end

```

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

```

8453 function self.set_state(s)
8454     local previous_state = self.get_state()
8455     for key, value in pairs(s) do
8456         self[key] = value
8457     end
8458     return previous_state
8459 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`.

```

8460 function self.defer_call(f)
8461     local previous_state = self.get_state()
8462     return function(...)
8463         local state = self.set_state(previous_state)

```

```

8464     local return_value = f(...)
8465     self.set_state(state)
8466     return return_value
8467 end
8468 end
8469
8470 return self
8471 end

```

### 3.1.5 Parsers

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

```

8472 parsers = {}

```

#### 3.1.5.1 Basic Parsers

```

8473 parsers.percent = P("%")
8474 parsers.at = P("@")
8475 parsers.comma = P(",")
8476 parsers.asterisk = P("*")
8477 parsers.dash = P("-")
8478 parsers.plus = P("+")
8479 parsers.underscore = P("_")
8480 parsers.period = P(".")
8481 parsers.hash = P("#")
8482 parsers.dollar = P("$")
8483 parsers.ampersand = P("&")
8484 parsers.backtick = P("`")
8485 parsers.less = P("<")
8486 parsers.more = P(">")
8487 parsers.space = P(" ")
8488 parsers.squote = P("'")
8489 parsers.dquote = P('"')
8490 parsers.lparent = P("(")
8491 parsers.rparent = P(")")
8492 parsers.lbracket = P("[")
8493 parsers.rbracket = P("]")
8494 parsers.lbrace = P("{")
8495 parsers.rbrace = P("}")
8496 parsers.circumflex = P("^")
8497 parsers.slash = P("/")
8498 parsers.equal = P("=")
8499 parsers.colon = P(":")
8500 parsers.semicolon = P(";")
8501 parsers.exclamation = P("!")
8502 parsers.pipe = P("|")

```

```

8503 parsers.tilde = P("~")
8504 parsers.backslash = P("\\")
8505 parsers.tab = P("\t")
8506 parsers.newline = P("\n")
8507
8508 parsers.digit = R("09")
8509 parsers.hexdigit = R("09", "af", "AF")
8510 parsers.letter = R("AZ", "az")
8511 parsers.alphanumeric = R("AZ", "az", "09")
8512 parsers.keyword = parsers.letter
8513 * (parsers.alphanumeric + parsers.dash)^0
8514
8515 parsers.doubleasterisks = P("**")
8516 parsers.doubleunderscores = P("__")
8517 parsers.doubletildes = P("~~")
8518 parsers.fourspace = P("    ")
8519
8520 parsers.any = P(1)
8521 parsers.eof = P(-1)
8522 parsers.succeed = P(true)
8523 parsers.fail = P(false)
8524
8525 parsers.internal_punctuation = S(":,;,.?")
8526 parsers.ascii_punctuation = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")
8527
8528 parsers.escapable = parsers.ascii_punctuation
8529 parsers.anyescaped = parsers.backslash / ""
8530 * parsers.escapable
8531 + parsers.any
8532
8533 parsers.spacechar = S("\t ")
8534 parsers.spacing = S(" \n\r\t")
8535 parsers.nonspacechar = parsers.any - parsers.spacing
8536 parsers.optionalspace = parsers.spacechar^0
8537
8538 parsers.normalchar = parsers.any - (V("SpecialChar")
8539 + parsers.spacing)
8540 parsers.nonindentpace = parsers.space^-3 * - parsers.spacechar
8541 parsers.indent = parsers.space^-3 * parsers.tab
8542 + parsers.fourspace / ""
8543 parsers.linechar = P(1 - parsers.newline)
8544
8545 parsers.blankline = parsers.optionalspace
8546 * parsers.newline / "\n"
8547 parsers.blanklines = parsers.blankline^0
8548 parsers.skipblanklines = ( parsers.optionalspace
8549 * parsers.newline)^0

```

```

8550 parsers.indentedline      = parsers.indent    /""
8551                          * C( parsers.linechar~1
8552                          * parsers.newline~-1)
8553 parsers.optionallyindentedline = parsers.indent~-1 /""
8554                          * C( parsers.linechar~1
8555                          * parsers.newline~-1)
8556 parsers.sp                  = parsers.spacing~0
8557 parsers.spnl                = parsers.optionalspace
8558                          * ( parsers.newline
8559                          * parsers.optionalspace)~-1
8560 parsers.line                 = parsers.linechar~0 * parsers.newline
8561 parsers.nonemptyline        = parsers.line - parsers.blankline
8562

```

### 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.

```

8563 parsers.unicode = {}
8564 parsers.unicode.preceding_punctuation = parsers.fail
8565 parsers.unicode.punctuation = parsers.fail
8566 parsers.unicode.alpha = parsers.fail
8567 parsers.unicode.numeric = parsers.fail
8568 parsers.unicode.word = parsers.fail
8569 parsers.unicode.preceding_whitespace = parsers.fail
8570 parsers.unicode.whitespace = parsers.fail
8571 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>36</sup>.

```

8572   local punctuation_of_length_n
8573   = unicode_data.categories.P[n]
8574   + unicode_data.categories.S[n]
8575   parsers.unicode.preceding_punctuation
8576   = parsers.unicode.preceding_punctuation
8577   + B(punctuation_of_length_n)
8578   parsers.unicode.punctuation
8579   = parsers.unicode.punctuation
8580   + punctuation_of_length_n

```

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

```

8581   local alpha_of_length_n = unicode_data.categories.L[n]
8582   parsers.unicode.alpha

```

<sup>36</sup>See <https://spec.commonmark.org/0.31.2/#unicode-punctuation-character>.

```

8583     = parsers.unicode.alpha
8584     + alpha_of_length_n

```

For numeric characters, accept any characters from Unicode category N (number), similar to the character class ‘Unicode.

```

8585     local numeric_of_length_n = unicode_data.categories.N[n]
8586     parsers.unicode.numeric
8587     = parsers.unicode.numeric
8588     + numeric_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 ‘

```

8589     local word_of_length_n
8590     = alpha_of_length_n
8591     + numeric_of_length_n
8592     + unicode_data.categories.Pc[n]
8593     parsers.unicode.word
8594     = parsers.unicode.word
8595     + 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.

```

8596     local whitespace_of_length_n = unicode_data.categories.Z[n]
8597     if n == 1 then
8598         whitespace_of_length_n
8599         = whitespace_of_length_n
8600         + R("\t\r")
8601     end
8602     parsers.unicode.preceding_whitespace
8603     = parsers.unicode.preceding_whitespace
8604     + B(whitespace_of_length_n)
8605     parsers.unicode.whitespace
8606     = parsers.unicode.whitespace
8607     + whitespace_of_length_n
8608 end

```

### 3.1.5.3 Parsers Used for Indentation

```

8609
8610 parsers.leader      = parsers.space^-3
8611

```

Check if a trail exists and is non-empty in the indent table `indent_table`.

```

8612 local function has_trail(indent_table)
8613     return indent_table ~= nil and
8614         indent_table.trail ~= nil and
8615         next(indent_table.trail) ~= nil
8616 end

```



8617

Check if indent table `indent_table` has any indents.

```
8618 local function has_indents(indent_table)
8619     return indent_table ~= nil and
8620         indent_table.indents ~= nil and
8621         next(indent_table.indents) ~= nil
8622 end
8623
```

Add a trail `trail_info` to the indent table `indent_table`.

```
8624 local function add_trail(indent_table, trail_info)
8625     indent_table.trail = trail_info
8626     return indent_table
8627 end
8628
```

Remove a trail `trail_info` from the indent table `indent_table`.

```
8629 local function remove_trail(indent_table)
8630     indent_table.trail = nil
8631     return indent_table
8632 end
8633
```

Update the indent table `indent_table` by adding or removing a new indent `add`.

```
8634 local function update_indent_table(indent_table, new_indent, add)
8635     indent_table = remove_trail(indent_table)
8636
8637     if not has_indents(indent_table) then
8638         indent_table.indents = {}
8639     end
8640
8641
8642     if add then
8643         indent_table.indents[#indent_table.indents + 1] = new_indent
8644     else
8645         if indent_table.indents[#indent_table.indents].name
8646             == new_indent.name then
8647             indent_table.indents[#indent_table.indents] = nil
8648         end
8649     end
8650
8651     return indent_table
8652 end
8653
```

Remove an indent by its name `name`.

```
8654 local function remove_indent(name)
8655     local remove_indent_level =
```

```

8656     function(s, i, indent_table) -- luacheck: ignore s i
8657         indent_table = update_indent_table(indent_table, {name=name},
8658   false)
8659         return true, indent_table
8660     end
8661
8662     return Cg(Cmt(Cb("indent_info"), remove_indent_level), "indent_info")
8663 end
8664

```

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`.

```

8665 local function process_starter_spacing(indent, spacing,
8666                                       minimum, left_strip_length)
8667     left_strip_length = left_strip_length or 0
8668
8669     local count = 0
8670     local tab_value = 4 - (indent) % 4
8671
8672     local code_started, minimum_found = false, false
8673     local code_start, minimum_remainder = "", ""
8674
8675     local left_total_stripped = 0
8676     local full_remainder = ""
8677
8678     if spacing ~= nil then
8679         for i = 1, #spacing do
8680             local character = spacing:sub(i, i)
8681
8682             if character == "\t" then
8683                 count = count + tab_value
8684                 tab_value = 4
8685             elseif character == " " then
8686                 count = count + 1
8687                 tab_value = 4 - (1 - tab_value) % 4
8688             end
8689
8690             if (left_strip_length ~= 0) then
8691                 local possible_to_strip = math.min(count, left_strip_length)
8692                 count = count - possible_to_strip
8693                 left_strip_length = left_strip_length - possible_to_strip
8694                 left_total_stripped = left_total_stripped + possible_to_strip
8695             else
8696                 full_remainder = full_remainder .. character

```

```

8697     end
8698
8699     if (minimum_found) then
8700         minimum_remainder = minimum_remainder .. character
8701     elseif (count >= minimum) then
8702         minimum_found = true
8703         minimum_remainder = minimum_remainder
8704             .. string.rep(" ", count - minimum)
8705     end
8706
8707     if (code_started) then
8708         code_start = code_start .. character
8709     elseif (count >= minimum + 4) then
8710         code_started = true
8711         code_start = code_start
8712             .. string.rep(" ", count - (minimum + 4))
8713     end
8714 end
8715 end
8716
8717 local remainder
8718 if (code_started) then
8719     remainder = code_start
8720 else
8721     remainder = string.rep(" ", count - minimum)
8722 end
8723
8724 local is_minimum = count >= minimum
8725 return {
8726     is_code = code_started,
8727     remainder = remainder,
8728     left_total_stripped = left_total_stripped,
8729     is_minimum = is_minimum,
8730     minimum_remainder = minimum_remainder,
8731     total_length = count,
8732     full_remainder = full_remainder
8733 }
8734 end
8735

```

Count the total width of all indents in the indent table [indent\\_table](#).

```

8736 local function count_indent_tab_level(indent_table)
8737     local count = 0
8738     if not has_indents(indent_table) then
8739         return count
8740     end
8741
8742     for i=1, #indent_table.indents do

```

```

8743     count = count + indent_table.indents[i].length
8744     end
8745     return count
8746 end
8747

```

Count the total width of a delimiter `delimiter`.

```

8748 local function total_delimiter_length(delimiter)
8749     local count = 0
8750     if type(delimiter) == "string" then return #delimiter end
8751     for _, value in pairs(delimiter) do
8752         count = count + total_delimiter_length(value)
8753     end
8754     return count
8755 end
8756

```

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`.

```

8757 local function process_starter_indent(_, _, indent_table, starter,
8758                                     is_blank, indent_type, breakable)
8759     local last_trail = starter[1]
8760     local delimiter = starter[2]
8761     local raw_new_trail = starter[3]
8762
8763     if indent_type == "bq" and not breakable then
8764         indent_table.ignore_blockquote_blank = true
8765     end
8766
8767     if has_trail(indent_table) then
8768         local trail = indent_table.trail
8769         if trail.is_code then
8770             return false
8771         end
8772         last_trail = trail.remainder
8773     else
8774         local sp = process_starter_spacing(0, last_trail, 0, 0)
8775
8776         if sp.is_code then
8777             return false
8778         end
8779         last_trail = sp.remainder
8780     end
8781
8782     local preceding_indentation = count_indent_tab_level(indent_table) % 4
8783     local last_trail_length = #last_trail
8784     local delimiter_length = total_delimiter_length(delimiter)
8785

```

```

8786 local total_indent_level = preceding_indentation + last_trail_length
8787                               + delimiter_length
8788
8789 local sp = {}
8790 if not is_blank then
8791     sp = process_starter_spacing(total_indent_level, raw_new_trail,
8792                                 0, 1)
8793 end
8794
8795 local del_trail_length = sp.left_total_stripped
8796 if is_blank then
8797     del_trail_length = 1
8798 elseif not sp.is_code then
8799     del_trail_length = del_trail_length + #sp.remainder
8800 end
8801
8802 local indent_length = last_trail_length + delimiter_length
8803                       + del_trail_length
8804 local new_indent_info = {name=indent_type, length=indent_length}
8805
8806 indent_table = update_indent_table(indent_table, new_indent_info,
8807                                     true)
8808 indent_table = add_trail(indent_table,
8809                           {is_code=sp.is_code,
8810                             remainder=sp.remainder,
8811                             total_length=sp.total_length,
8812                             full_remainder=sp.full_remainder})
8813
8814 return true, indent_table
8815 end
8816

```

Return the pattern corresponding with the indent name `name`.

```

8817 local function decode_pattern(name)
8818     local delimiter = parsers.succeed
8819     if name == "bq" then
8820         delimiter = parsers.more
8821     end
8822
8823     return C(parsers.optionalspace) * C(delimiter)
8824           * C(parsers.optionalspace) * Cp()
8825 end
8826

```

Find the first blank-only indent of the indent table `indent_table` followed by blank-only indents.

```

8827 local function left_blank_starter(indent_table)
8828     local blank_starter_index

```

```

8829
8830   if not has_indents(indent_table) then
8831       return
8832   end
8833
8834   for i = #indent_table.indents,1,-1 do
8835       local value = indent_table.indents[i]
8836       if value.name == "li" then
8837           blank_starter_index = i
8838       else
8839           break
8840       end
8841   end
8842
8843   return blank_starter_index
8844 end
8845

```

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.

```

8846 local function traverse_indent(s, i, indent_table, is_optional,
8847                               is_blank, current_line_indents)
8848     local new_index = i
8849
8850     local preceding_indentation = 0
8851     local current_trail = {}
8852
8853     local blank_starter = left_blank_starter(indent_table)
8854
8855     if current_line_indents == nil then
8856         current_line_indents = {}
8857     end
8858
8859     for index = 1,#indent_table.indents do
8860         local value = indent_table.indents[index]
8861         local pattern = decode_pattern(value.name)
8862
8863         -- match decoded pattern
8864         local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
8865         if new_indent_info == nil then
8866             local blankline_end = lpeg.match(
8867                 Ct(parsers.blankline * Cg(Cp(), "pos")), s, new_index)
8868             if is_optional or not indent_table.ignore_blockquote_blank
8869                 or not blankline_end then

```

```

8870         return is_optional, new_index, current_trail,
8871                current_line_indents
8872     end
8873
8874     return traverse_indent(s, tonumber(blankline_end.pos),
8875                           indent_table, is_optional, is_blank,
8876                           current_line_indents)
8877 end
8878
8879 local raw_last_trail = new_indent_info[1]
8880 local delimiter = new_indent_info[2]
8881 local raw_new_trail = new_indent_info[3]
8882 local next_index = new_indent_info[4]
8883
8884 local space_only = delimiter == ""
8885
8886 -- check previous trail
8887 if not space_only and next(current_trail) == nil then
8888     local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
8889     current_trail = {is_code=sp.is_code, remainder=sp.remainder,
8890                     total_length=sp.total_length,
8891                     full_remainder=sp.full_remainder}
8892 end
8893
8894 if next(current_trail) ~= nil then
8895     if not space_only and current_trail.is_code then
8896         return is_optional, new_index, current_trail,
8897                current_line_indents
8898     end
8899     if current_trail.internal_remainder ~= nil then
8900         raw_last_trail = current_trail.internal_remainder
8901     end
8902 end
8903
8904 local raw_last_trail_length = 0
8905 local delimiter_length = 0
8906
8907 if not space_only then
8908     delimiter_length = #delimiter
8909     raw_last_trail_length = #raw_last_trail
8910 end
8911
8912 local total_indent_level = preceding_indentation
8913                          + raw_last_trail_length + delimiter_length
8914
8915 local spacing_to_process
8916 local minimum = 0

```

```

8917     local left_strip_length = 0
8918
8919     if not space_only then
8920         spacing_to_process = raw_new_trail
8921         left_strip_length = 1
8922     else
8923         spacing_to_process = raw_last_trail
8924         minimum = value.length
8925     end
8926
8927     local sp = process_starter_spacing(total_indent_level,
8928                                       spacing_to_process, minimum,
8929                                       left_strip_length)
8930
8931     if space_only and not sp.is_minimum then
8932         return is_optional or (is_blank and blank_starter <= index),
8933             new_index, current_trail, current_line_indents
8934     end
8935
8936     local indent_length = raw_last_trail_length + delimiter_length
8937                       + sp.left_total_stripped
8938
8939     -- update info for the next pattern
8940     if not space_only then
8941         preceding_indentation = preceding_indentation + indent_length
8942     else
8943         preceding_indentation = preceding_indentation + value.length
8944     end
8945
8946     current_trail = {is_code=sp.is_code, remainder=sp.remainder,
8947                    internal_remainder=sp.minimum_remainder,
8948                    total_length=sp.total_length,
8949                    full_remainder=sp.full_remainder}
8950
8951     current_line_indents[#current_line_indents + 1] = new_indent_info
8952     new_index = next_index
8953 end
8954
8955 return true, new_index, current_trail, current_line_indents
8956 end
8957

```

Check if a code trail is expected.

```

8958 local function check_trail(expect_code, is_code)
8959     return (expect_code and is_code) or (not expect_code and not is_code)
8960 end
8961

```



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`.

```

8962 local check_trail_joined =
8963   function(s, i, indent_table, -- luacheck: ignore s i
8964           spacing, expect_code, omit_remainder)
8965     local is_code
8966     local remainder
8967
8968     if has_trail(indent_table) then
8969       local trail = indent_table.trail
8970       is_code = trail.is_code
8971       if is_code then
8972         remainder = trail.remainder
8973       else
8974         remainder = trail.full_remainder
8975       end
8976     else
8977       local sp = process_starter_spacing(0, spacing, 0, 0)
8978       is_code = sp.is_code
8979       if is_code then
8980         remainder = sp.remainder
8981       else
8982         remainder = sp.full_remainder
8983       end
8984     end
8985
8986     local result = check_trail(expect_code, is_code)
8987     if omit_remainder then
8988       return result
8989     end
8990     return result, remainder
8991   end
8992
```

Check if the current trail of the `indent_table` is of length between `min` and `max`.

```

8993 local check_trail_length =
8994   function(s, i, indent_table, -- luacheck: ignore s i
8995           spacing, min, max)
8996     local trail
8997
8998     if has_trail(indent_table) then
8999       trail = indent_table.trail
9000     else
9001       trail = process_starter_spacing(0, spacing, 0, 0)
9002     end
9003
```

```

9004     local total_length = trail.total_length
9005     if total_length == nil then
9006         return false
9007     end
9008
9009     return min <= total_length and total_length <= max
9010 end
9011

```

Check the indentation of the continuation line, optionally with the mode `is_optional` selected. Check blank line exclusively with `is_blank`.

```

9012 local function check_continuation_indentation(s, i, indent_table,
9013   is_optional, is_blank)
9014     if not has_indents(indent_table) then
9015         return true
9016     end
9017
9018     local passes, new_index, current_trail, current_line_indents =
9019         traverse_indent(s, i, indent_table, is_optional, is_blank)
9020
9021     if passes then
9022         indent_table.current_line_indents = current_line_indents
9023         indent_table = add_trail(indent_table, current_trail)
9024         return new_index, indent_table
9025     end
9026     return false
9027 end
9028

```

Get name of the last indent from the `indent_table`.

```

9029 local function get_last_indent_name(indent_table)
9030     if has_indents(indent_table) then
9031         return indent_table.indents[#indent_table.indents].name
9032     end
9033 end
9034

```

Remove the remainder altogether if the last indent from the `indent_table` is blank-only.

```

9035 local function remove_remainder_if_blank(indent_table, remainder)
9036     if get_last_indent_name(indent_table) == "li" then
9037         return ""
9038     end
9039     return remainder
9040 end
9041

```

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.

```

9042 local check_trail_type =
9043   function(s, i, -- luacheck: ignore s i
9044             trail, spacing, trail_type)
9045     if trail == nil then
9046       trail = process_starter_spacing(0, spacing, 0, 0)
9047     end
9048
9049     if trail_type == "non-code" then
9050       return check_trail(false, trail.is_code)
9051     end
9052     if trail_type == "code" then
9053       return check_trail(true, trail.is_code)
9054     end
9055     if trail_type == "full-code" then
9056       if (trail.is_code) then
9057         return i, trail.remainder
9058       end
9059       return i, ""
9060     end
9061     if trail_type == "full-any" then
9062       return i, trail.internal_remainder
9063     end
9064   end
9065
```

Stores or restores an `is_freezing` trail from indent table `indent_table`.

```

9066 local trail_freezing =
9067   function(s, i, -- luacheck: ignore s i
9068             indent_table, is_freezing)
9069     if is_freezing then
9070       if indent_table.is_trail_frozen then
9071         indent_table.trail = indent_table.frozen_trail
9072       else
9073         indent_table.frozen_trail = indent_table.trail
9074         indent_table.is_trail_frozen = true
9075       end
9076     else
9077       indent_table.frozen_trail = nil
9078       indent_table.is_trail_frozen = false
9079     end
9080     return true, indent_table
9081   end
9082
```

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`.

```
9083 local check_continuation_indentation_and_trail =
9084   function (s, i, indent_table, is_optional, is_blank, trail_type,
9085             reset_rem, omit_remainder)
9086     if not has_indents(indent_table) then
9087       local spacing, new_index = lpeg.match( C(parsers.spacechar^0)
9088   * Cp(), s, i)
9089       local result, remainder = check_trail_type(s, i,
9090         indent_table.trail, spacing, trail_type)
9091       if remainder == nil then
9092         if result then
9093           return new_index
9094         end
9095         return false
9096       end
9097       if result then
9098         return new_index, remainder
9099       end
9100       return false
9101     end
9102
9103     local passes, new_index, current_trail = traverse_indent(s, i,
9104       indent_table, is_optional, is_blank)
9105
9106     if passes then
9107       local spacing
9108       if current_trail == nil then
9109         local newer_spacing, newer_index = lpeg.match(
9110           C(parsers.spacechar^0) * Cp(), s, i)
9111         current_trail = process_starter_spacing(0, newer_spacing, 0, 0)
9112         new_index = newer_index
9113         spacing = newer_spacing
9114       else
9115         spacing = current_trail.remainder
9116       end
9117       local result, remainder = check_trail_type(s, new_index,
9118         current_trail, spacing, trail_type)
9119       if remainder == nil or omit_remainder then
9120         if result then
9121           return new_index
9122         end
9123         return false
9124       end
9125
9126       if is_blank and reset_rem then
9127         remainder = remove_remainder_if_blank(indent_table, remainder)
```

```

9128     end
9129     if result then
9130         return new_index, remainder
9131     end
9132     return false
9133 end
9134 return false
9135 end
9136

```

The following patterns check whitespace indentation at the start of a block.

```

9137 parsers.check_trail = Cmt( Cb("indent_info") * C(parsers.spacechar^0)
9138                        * Cc(false), check_trail_joined)
9139
9140 parsers.check_trail_no_rem = Cmt( Cb("indent_info")
9141                                * C(parsers.spacechar^0) * Cc(false)
9142                                * Cc(true), check_trail_joined)
9143
9144 parsers.check_code_trail  = Cmt( Cb("indent_info")
9145                                * C(parsers.spacechar^0)
9146                                * Cc(true), check_trail_joined)
9147
9148 parsers.check_trail_length_range = function(min, max)
9149     return Cmt( Cb("indent_info") * C(parsers.spacechar^0) * Cc(min)
9150               * Cc(max), check_trail_length)
9151 end
9152
9153 parsers.check_trail_length = function(n)
9154     return parsers.check_trail_length_range(n, n)
9155 end
9156

```

The following patterns handle trail backup, to prevent a failing pattern to modify it before passing it to the next.

```

9157 parsers.freeze_trail = Cg( Cmt(Cb("indent_info")
9158                        * Cc(true), trail_freezing), "indent_info")
9159
9160 parsers.unfreeze_trail = Cg(Cmt(Cb("indent_info") * Cc(false),
9161                        trail_freezing), "indent_info")
9162

```

The following patterns check indentation in continuation lines as defined by the container start.

```

9163 parsers.check_minimal_indent = Cmt(Cb("indent_info") * Cc(false),
9164                                check_continuation_indentation)
9165
9166 parsers.check_optional_indent = Cmt(Cb("indent_info") * Cc(true),
9167                                check_continuation_indentation)

```

```

9168
9169 parsers.check_minimal_blank_indent
9170     = Cmt( Cb("indent_info") * Cc(false)
9171           * Cc(true)
9172           , check_continuation_indentation)
9173

```

The following patterns check indentation in continuation lines as defined by the container start. Additionally the subsequent trail is also directly checked.

```

9174
9175 parsers.check_minimal_indent_and_trail =
9176     Cmt( Cb("indent_info")
9177         * Cc(false) * Cc(false) * Cc("non-code") * Cc(true)
9178         , check_continuation_indentation_and_trail)
9179
9180 parsers.check_minimal_indent_and_code_trail =
9181     Cmt( Cb("indent_info")
9182         * Cc(false) * Cc(false) * Cc("code") * Cc(false)
9183         , check_continuation_indentation_and_trail)
9184
9185 parsers.check_minimal_blank_indent_and_full_code_trail =
9186     Cmt( Cb("indent_info")
9187         * Cc(false) * Cc(true) * Cc("full-code") * Cc(true)
9188         , check_continuation_indentation_and_trail)
9189
9190 parsers.check_minimal_indent_and_any_trail =
9191     Cmt( Cb("indent_info")
9192         * Cc(false) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
9193         , check_continuation_indentation_and_trail)
9194
9195 parsers.check_minimal_blank_indent_and_any_trail =
9196     Cmt( Cb("indent_info")
9197         * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
9198         , check_continuation_indentation_and_trail)
9199
9200 parsers.check_minimal_blank_indent_and_any_trail_no_rem =
9201     Cmt( Cb("indent_info")
9202         * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(true)
9203         , check_continuation_indentation_and_trail)
9204
9205 parsers.check_optional_indent_and_any_trail =
9206     Cmt( Cb("indent_info")
9207         * Cc(true) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
9208         , check_continuation_indentation_and_trail)
9209
9210 parsers.check_optional_blank_indent_and_any_trail =
9211     Cmt( Cb("indent_info")

```

```

9212      * Cc(true) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
9213      , check_continuation_indentation_and_trail)
9214

```

The following patterns specify behaviour around newlines.

```

9215
9216 parsers.spnlc_noexc = parsers.optionalspace
9217                      * ( parsers.newline
9218                        * parsers.check_minimal_indent_and_any_trail)^-1
9219
9220 parsers.spnlc = parsers.optionalspace
9221               * (V("EndlineNoSub"))^-1
9222
9223 parsers.spnlc_sep = parsers.optionalspace * V("EndlineNoSub")
9224                  + parsers.spacechar^1
9225
9226 parsers.only_blank = parsers.spacechar^0
9227                   * (parsers.newline + parsers.eof)
9228

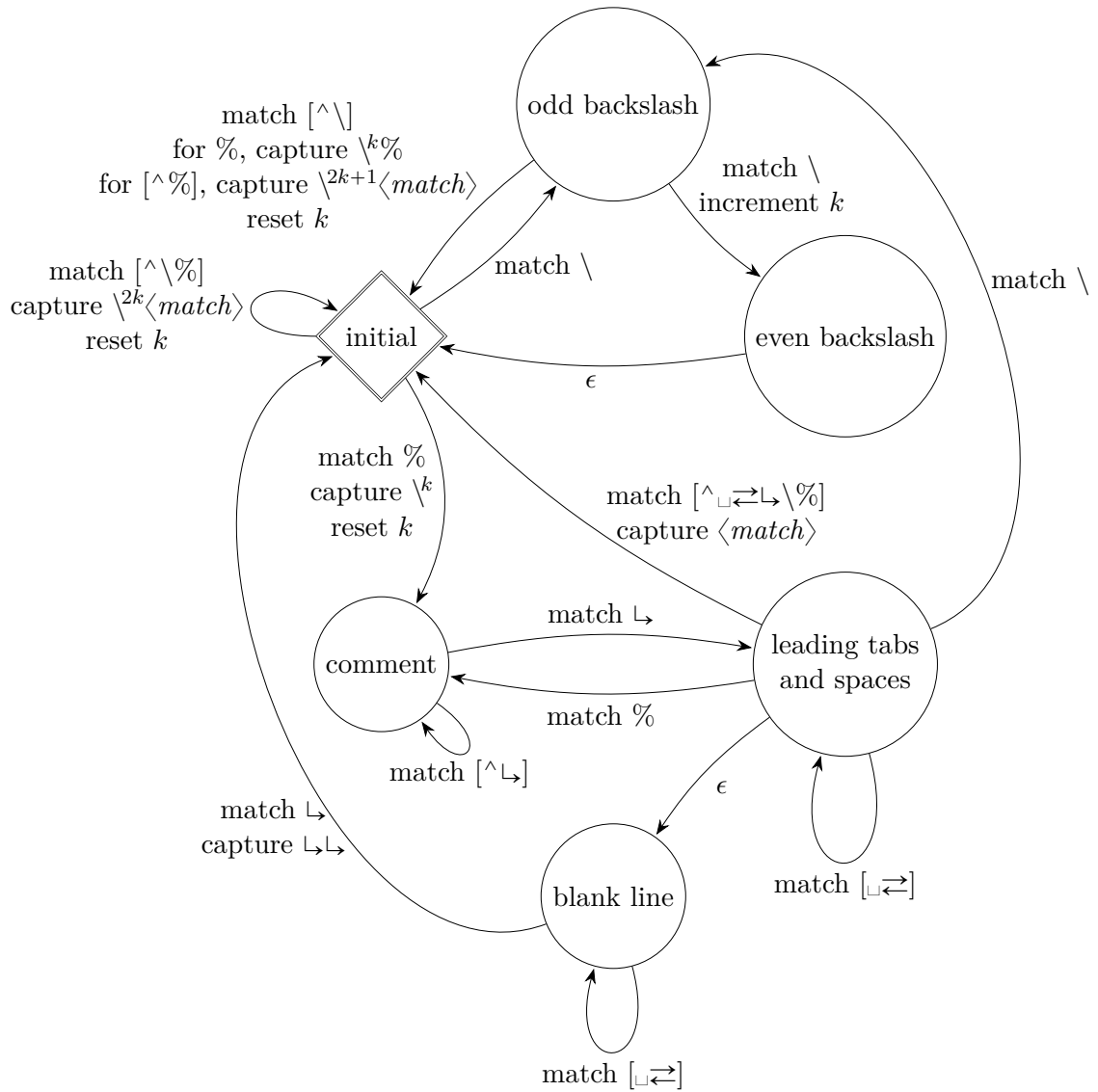
```

The `parsers.commented\_line^1` parser recognizes the regular language of  $\text{T}_{\text{E}}\text{X}$  comments, see an equivalent finite automaton in Figure 8.

```

9229 parsers.commented_line_letter = parsers.linechar
9230                               + parsers.newline
9231                               - parsers.backslash
9232                               - parsers.percent
9233 parsers.commented_line = Cg(Cc(""), "backslashes")
9234                        * Cg(Cc(true), "pure_comment")
9235                        * ((#(parsers.commented_line_letter
9236                          - parsers.newline)
9237                          * Cb("backslashes")
9238                          * Cs(parsers.commented_line_letter
9239                            - parsers.newline)^1 -- initial
9240                          * Cg(Cc(""), "backslashes")
9241                          * Cg(Cc(false), "pure_comment"))
9242                        + #( parsers.backslash
9243                          * (parsers.backslash + parsers.newline))
9244                        * Cg((parsers.backslash -- even backslash
9245                          * ( parsers.backslash
9246                            + #parsers.newline))^1, "backslashes")
9247                        * Cg(Cc(false), "pure_comment")
9248                        + (parsers.backslash
9249                          * (#parsers.percent
9250                            * Cb("backslashes")
9251                            / function(backslashes)
9252                              return string.rep("\\", #backslashes / 2)
9253                            end
9254                          * C(parsers.percent)

```



**Figure 8: A pushdown automaton that recognizes  $\text{\TeX}$  comments**



```

9255         + #parsers.commented_line_letter
9256         * Cb("backslashes")
9257         * Cc("\\")
9258         * C(parsers.commented_line_letter))
9259         * Cg(Cc(""), "backslashes")
9260         * Cg(Cc(false), "pure_comment"))^0
9261     * (#parsers.percent
9262     * Cb("backslashes")
9263     / function(backslashes)
9264     return string.rep("\\", #backslashes / 2)
9265     end
9266     * ((Cb("pure_comment")
9267     * parsers.percent -- comment
9268     * parsers.line
9269     * #parsers.blankline) -- blank line
9270     / function(is_pure)
9271     return is_pure and "" or "\n"
9272     end
9273     + parsers.percent -- comment
9274     * parsers.line
9275     * parsers.optionalspace) -- leading spaces
9276     + #parsers.newline)
9277     * Cb("backslashes")
9278     * C(parsers.newline))
9279
9280 parsers.chunk = parsers.line * (parsers.optionallyindentedline
9281                               - parsers.blankline)^0
9282
9283 parsers.attribute_key_char = parsers.unicode.alpha
9284                             + parsers.unicode.numeric
9285                             + S("-_:.")
9286 parsers.attribute_raw_char = parsers.unicode.alpha
9287                             + parsers.unicode.numeric
9288                             + S("-_")
9289 parsers.attribute_key = (parsers.attribute_key_char
9290                        - parsers.dash - parsers.digit)
9291                        * parsers.attribute_key_char^0
9292 parsers.attribute_value = ( (parsers.dquote / "")
9293                          * (parsers.anyescaped - parsers.dquote)^0
9294                          * (parsers.dquote / ""))
9295 + ( (parsers.squote / "")
9296   * (parsers.anyescaped - parsers.squote)^0
9297   * (parsers.squote / ""))
9298 + ( parsers.anyescaped
9299   - parsers.dquote
9300   - parsers.rbrace
9301   - parsers.space)^0

```

```

9302 parsers.attribute_identifier = parsers.attribute_key_char^1
9303 parsers.attribute_classname = parsers.unicode.alpha
9304                               * parsers.attribute_key_char^0
9305 parsers.attribute_raw = parsers.attribute_raw_char^1
9306
9307 parsers.attribute = (parsers.dash * Cc(".unnumbered"))
9308                   + C( parsers.hash
9309                       * parsers.attribute_identifier)
9310                   + C( parsers.period
9311                       * parsers.attribute_classname)
9312                   + Cs( parsers.attribute_key
9313                       * parsers.optionalspace
9314                       * parsers.equal
9315                       * parsers.optionalspace
9316                       * parsers.attribute_value)
9317 parsers.attributes = parsers.lbrace
9318                   * parsers.optionalspace
9319                   * parsers.attribute
9320                   * (parsers.spacechar^1
9321                   * parsers.attribute)^0
9322                   * parsers.optionalspace
9323                   * parsers.rbrace
9324
9325 parsers.raw_attribute = parsers.lbrace
9326                   * parsers.optionalspace
9327                   * parsers.equal
9328                   * C(parsers.attribute_raw)
9329                   * parsers.optionalspace
9330                   * parsers.rbrace
9331
9332 -- block followed by 0 or more optionally
9333 -- indented blocks with first line indented.
9334 parsers.indented_blocks = function(bl)
9335   return Cs( bl
9336             * ( parsers.blankline^1
9337               * parsers.indent
9338               * -parsers.blankline
9339               * bl)^0
9340             * (parsers.blankline^1 + parsers.eof) )
9341 end

```

#### 3.1.5.4 Parsers Used for html Entities

```

9342 local function repeat_between(pattern, min, max)
9343   return -pattern^(max + 1) * pattern^min
9344 end
9345

```

```

9346 parsers.hexentity = parsers.ampersand * parsers.hash * C(S("Xx"))
9347         * C(repeat_between(parsers.hexdigit, 1, 6))
9348         * parsers.semicolon
9349 parsers.decentity = parsers.ampersand * parsers.hash
9350         * C(repeat_between(parsers.digit, 1, 7))
9351         * parsers.semicolon
9352 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
9353         * parsers.semicolon
9354
9355 parsers.html_entities
9356   = parsers.hexentity / entities.hex_entity_with_x_char
9357   + parsers.decentity / entities.dec_entity
9358   + parsers.tagentity / entities.char_entity
9359 parsers.html_space_entity_char
9360   = Cmt(parsers.html_entities, function(_, i, entity)
9361     return entity == " " and i or nil
9362   end)

```

### 3.1.5.5 Parsers Used for Markdown Lists

```

9363 parsers.bullet = function(bullet_char, interrupting)
9364   local allowed_end
9365   if interrupting then
9366     allowed_end = C(parsers.spacechar^1) * #parsers.linechar
9367   else
9368     allowed_end = C(parsers.spacechar^1)
9369                 + #(parsers.newline + parsers.eof)
9370   end
9371   return parsers.check_trail
9372     * Ct(C(bullet_char) * Cc(""))
9373     * allowed_end
9374 end
9375
9376 local function tickbox(interior)
9377   return parsers.optionalspace * parsers.lbracket
9378     * interior * parsers.rbracket * parsers.spacechar^1
9379 end
9380
9381 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
9382 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
9383 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
9384

```

### 3.1.5.6 Parsers Used for Markdown Code Spans

```

9385 parsers.openticks = Cg(parsers.backtick^1, "ticks")
9386
9387 local function captures_equal_length(_, i, a, b)

```

```

9388     return #a == #b and i
9389 end
9390
9391 parsers.closeticks = Cmt(C(parsers.backtick^1)
9392                        * Cb("ticks"), captures_equal_length)
9393
9394 parsers.intickschar = (parsers.any - S("\n\r`"))
9395                      + V("NoSoftLineBreakEndline")
9396                      + (parsers.backtick^1 - parsers.closeticks)
9397
9398 local function process_inticks(s)
9399     s = s:gsub("\n", " ")
9400     s = s:gsub("^ (.*) $", "%1")
9401     return s
9402 end
9403
9404 parsers.inticks = parsers.openticks
9405                 * C(parsers.space^0)
9406                 * parsers.closeticks
9407                 + parsers.openticks
9408                 * Cs(Cs(parsers.intickschar^0) / process_inticks)
9409                 * parsers.closeticks
9410

```

### 3.1.5.7 Parsers Used for html

```

9411 -- case-insensitive match (we assume s is lowercase)
9412 -- must be single byte encoding
9413 parsers.keyword_exact = function(s)
9414     local parser = P(0)
9415     for i=1,#s do
9416         local c = s:sub(i,i)
9417         local m = c .. upper(c)
9418         parser = parser * S(m)
9419     end
9420     return parser
9421 end
9422
9423 parsers.special_block_keyword =
9424     parsers.keyword_exact("pre") +
9425     parsers.keyword_exact("script") +
9426     parsers.keyword_exact("style") +
9427     parsers.keyword_exact("textarea")
9428
9429 parsers.block_keyword =
9430     parsers.keyword_exact("address") +
9431     parsers.keyword_exact("article") +

```

```
9432 parsers.keyword_exact("aside") +
9433 parsers.keyword_exact("base") +
9434 parsers.keyword_exact("basefont") +
9435 parsers.keyword_exact("blockquote") +
9436 parsers.keyword_exact("body") +
9437 parsers.keyword_exact("caption") +
9438 parsers.keyword_exact("center") +
9439 parsers.keyword_exact("col") +
9440 parsers.keyword_exact("colgroup") +
9441 parsers.keyword_exact("dd") +
9442 parsers.keyword_exact("details") +
9443 parsers.keyword_exact("dialog") +
9444 parsers.keyword_exact("dir") +
9445 parsers.keyword_exact("div") +
9446 parsers.keyword_exact("dl") +
9447 parsers.keyword_exact("dt") +
9448 parsers.keyword_exact("fieldset") +
9449 parsers.keyword_exact("figcaption") +
9450 parsers.keyword_exact("figure") +
9451 parsers.keyword_exact("footer") +
9452 parsers.keyword_exact("form") +
9453 parsers.keyword_exact("frame") +
9454 parsers.keyword_exact("frameset") +
9455 parsers.keyword_exact("h1") +
9456 parsers.keyword_exact("h2") +
9457 parsers.keyword_exact("h3") +
9458 parsers.keyword_exact("h4") +
9459 parsers.keyword_exact("h5") +
9460 parsers.keyword_exact("h6") +
9461 parsers.keyword_exact("head") +
9462 parsers.keyword_exact("header") +
9463 parsers.keyword_exact("hr") +
9464 parsers.keyword_exact("html") +
9465 parsers.keyword_exact("iframe") +
9466 parsers.keyword_exact("legend") +
9467 parsers.keyword_exact("li") +
9468 parsers.keyword_exact("link") +
9469 parsers.keyword_exact("main") +
9470 parsers.keyword_exact("menu") +
9471 parsers.keyword_exact("menuitem") +
9472 parsers.keyword_exact("nav") +
9473 parsers.keyword_exact("noframes") +
9474 parsers.keyword_exact("ol") +
9475 parsers.keyword_exact("optgroup") +
9476 parsers.keyword_exact("option") +
9477 parsers.keyword_exact("p") +
9478 parsers.keyword_exact("param") +
```

```

9479     parsers.keyword_exact("section") +
9480     parsers.keyword_exact("source") +
9481     parsers.keyword_exact("summary") +
9482     parsers.keyword_exact("table") +
9483     parsers.keyword_exact("tbody") +
9484     parsers.keyword_exact("td") +
9485     parsers.keyword_exact("tfoot") +
9486     parsers.keyword_exact("th") +
9487     parsers.keyword_exact("thead") +
9488     parsers.keyword_exact("title") +
9489     parsers.keyword_exact("tr") +
9490     parsers.keyword_exact("track") +
9491     parsers.keyword_exact("ul")
9492
9493 -- end conditions
9494 parsers.html_blankline_end_condition
9495   = parsers.linechar^0
9496   * ( parsers.newline
9497       * (parsers.check_minimal_blank_indent_and_any_trail
9498         * #parsers.blankline
9499         + parsers.check_minimal_indent_and_any_trail)
9500       * parsers.linechar^1)^0
9501   * (parsers.newline^-1 / "")
9502
9503 local function remove_trailing_blank_lines(s)
9504   return s:gsub("[\n\r]+%s*$", "")
9505 end
9506
9507 parsers.html_until_end = function(end_marker)
9508   return Cs(Cs((parsers.newline
9509                 * (parsers.check_minimal_blank_indent_and_any_trail
9510                   * #parsers.blankline
9511                   + parsers.check_minimal_indent_and_any_trail)
9512                 + parsers.linechar - end_marker)^0
9513             * parsers.linechar^0 * parsers.newline^-1)
9514           / remove_trailing_blank_lines)
9515 end
9516
9517 -- attributes
9518 parsers.html_attribute_spacing = parsers.optionalspace
9519                                * V("NoSoftLineBreakEndline")
9520                                * parsers.optionalspace
9521                                + parsers.spacechar^1
9522
9523 parsers.html_attribute_name = ( parsers.letter
9524                                + parsers.colon
9525                                + parsers.underscore)

```

```

9526             * ( parsers.alphanumeric
9527                 + parsers.colon
9528                 + parsers.underscore
9529                 + parsers.period
9530                 + parsers.dash)^0
9531
9532 parsers.html_attribute_value = parsers.squote
9533                               * (parsers.linechar - parsers.squote)^0
9534                               * parsers.squote
9535                               + parsers.dquote
9536                               * (parsers.linechar - parsers.dquote)^0
9537                               * parsers.dquote
9538                               + ( parsers.any
9539                                 - parsers.spacechar
9540                                 - parsers.newline
9541                                 - parsers.dquote
9542                                 - parsers.squote
9543                                 - parsers.backtick
9544                                 - parsers.equal
9545                                 - parsers.less
9546                                 - parsers.more)^1
9547
9548 parsers.html_inline_attribute_value = parsers.squote
9549                                     * (V("NoSoftLineBreakEndline")
9550                                       + parsers.any
9551                                       - parsers.blankline^2
9552                                       - parsers.squote)^0
9553                                     * parsers.squote
9554                                     + parsers.dquote
9555                                     * (V("NoSoftLineBreakEndline")
9556                                       + parsers.any
9557                                       - parsers.blankline^2
9558                                       - parsers.dquote)^0
9559                                     * parsers.dquote
9560                                     + (parsers.any
9561                                       - parsers.spacechar
9562                                       - parsers.newline
9563                                       - parsers.dquote
9564                                       - parsers.squote
9565                                       - parsers.backtick
9566                                       - parsers.equal
9567                                       - parsers.less
9568                                       - parsers.more)^1
9569
9570 parsers.html_attribute_value_specification
9571   = parsers.optionalspace
9572   * parsers.equal

```

```

9573 * parsers.optionalspace
9574 * parsers.html_attribute_value
9575
9576 parsers.html_spnl = parsers.optionalspace
9577 * (V("NoSoftLineBreakEndline")
9578 * parsers.optionalspace)^-1
9579
9580 parsers.html_inline_attribute_value_specification
9581 = parsers.html_spnl
9582 * parsers.equal
9583 * parsers.html_spnl
9584 * parsers.html_inline_attribute_value
9585
9586 parsers.html_attribute
9587 = parsers.html_attribute_spacing
9588 * parsers.html_attribute_name
9589 * parsers.html_inline_attribute_value_specification^-1
9590
9591 parsers.html_non_newline_attribute
9592 = parsers.spacechar^1
9593 * parsers.html_attribute_name
9594 * parsers.html_attribute_value_specification^-1
9595
9596 parsers.nested_breaking_blank = parsers.newline
9597 * parsers.check_minimal_blank_indent
9598 * parsers.blankline
9599
9600 parsers.html_comment_start = P("<!--")
9601
9602 parsers.html_comment_end = P("-->")
9603
9604 parsers.html_comment
9605 = Cs( parsers.html_comment_start
9606 * parsers.html_until_end(parsers.html_comment_end))
9607
9608 parsers.html_inline_comment = (parsers.html_comment_start / "")
9609 * -P(">") * -P("-->")
9610 * Cs(( V("NoSoftLineBreakEndline")
9611 + parsers.any
9612 - parsers.nested_breaking_blank
9613 - parsers.html_comment_end)^0)
9614 * (parsers.html_comment_end / "")
9615
9616 parsers.html_cdatasection_start = P("<![CDATA[")
9617
9618 parsers.html_cdatasection_end = P("]]>")
9619

```



```

9620 parsers.html_cdatasection
9621     = Cs( parsers.html_cdatasection_start
9622           * parsers.html_until_end(parsers.html_cdatasection_end))
9623
9624 parsers.html_inline_cdatasection
9625     = parsers.html_cdatasection_start
9626       * Cs(V("NoSoftLineBreakEndline") + parsers.any
9627           - parsers.nested_breaking_blank - parsers.html_cdatasection_end)^0
9628       * parsers.html_cdatasection_end
9629
9630 parsers.html_declaration_start = P("<!") * parsers.letter
9631
9632 parsers.html_declaration_end = P(">")
9633
9634 parsers.html_declaration
9635     = Cs( parsers.html_declaration_start
9636           * parsers.html_until_end(parsers.html_declaration_end))
9637
9638 parsers.html_inline_declaration
9639     = parsers.html_declaration_start
9640       * Cs(V("NoSoftLineBreakEndline") + parsers.any
9641           - parsers.nested_breaking_blank - parsers.html_declaration_end)^0
9642       * parsers.html_declaration_end
9643
9644 parsers.html_instruction_start = P("<?")
9645
9646 parsers.html_instruction_end = P("?>")
9647
9648 parsers.html_instruction
9649     = Cs( parsers.html_instruction_start
9650           * parsers.html_until_end(parsers.html_instruction_end))
9651
9652 parsers.html_inline_instruction = parsers.html_instruction_start
9653                                * Cs( V("NoSoftLineBreakEndline")
9654                                    + parsers.any
9655                                    - parsers.nested_breaking_blank
9656                                    - parsers.html_instruction_end)^0
9657                                * parsers.html_instruction_end
9658
9659 parsers.html_blankline = parsers.newline
9660                        * parsers.optionalspace
9661                        * parsers.newline
9662
9663 parsers.html_tag_start = parsers.less
9664
9665 parsers.html_tag_closing_start = parsers.less
9666                               * parsers.slash

```

```

9667
9668 parsers.html_tag_end = parsers.html_spnl
9669                        * parsers.more
9670
9671 parsers.html_empty_tag_end = parsers.html_spnl
9672                        * parsers.slash
9673                        * parsers.more
9674
9675 -- opening tags
9676 parsers.html_any_open_inline_tag = parsers.html_tag_start
9677                        * parsers.keyword
9678                        * parsers.html_attribute^0
9679                        * parsers.html_tag_end
9680
9681 parsers.html_any_open_tag = parsers.html_tag_start
9682                        * parsers.keyword
9683                        * parsers.html_non_newline_attribute^0
9684                        * parsers.html_tag_end
9685
9686 parsers.html_open_tag = parsers.html_tag_start
9687                        * parsers.block_keyword
9688                        * parsers.html_attribute^0
9689                        * parsers.html_tag_end
9690
9691 parsers.html_open_special_tag = parsers.html_tag_start
9692                        * parsers.special_block_keyword
9693                        * parsers.html_attribute^0
9694                        * parsers.html_tag_end
9695
9696 -- incomplete tags
9697 parsers.incomplete_tag_following = parsers.spacechar
9698                                + parsers.more
9699                                + parsers.slash * parsers.more
9700                                + #(parsers.newline + parsers.eof)
9701
9702 parsers.incomplete_special_tag_following = parsers.spacechar
9703                                + parsers.more
9704                                + #( parsers.newline
9705                                + parsers.eof)
9706
9707 parsers.html_incomplete_open_tag = parsers.html_tag_start
9708                                * parsers.block_keyword
9709                                * parsers.incomplete_tag_following
9710
9711 parsers.html_incomplete_open_special_tag
9712 = parsers.html_tag_start
9713 * parsers.special_block_keyword

```

```

9714 * parsers.incomplete_special_tag_following
9715
9716 parsers.html_incomplete_close_tag = parsers.html_tag_closing_start
9717 * parsers.block_keyword
9718 * parsers.incomplete_tag_following
9719
9720 parsers.html_incomplete_close_special_tag
9721 = parsers.html_tag_closing_start
9722 * parsers.special_block_keyword
9723 * parsers.incomplete_tag_following
9724
9725 -- closing tags
9726 parsers.html_close_tag = parsers.html_tag_closing_start
9727 * parsers.block_keyword
9728 * parsers.html_tag_end
9729
9730 parsers.html_any_close_tag = parsers.html_tag_closing_start
9731 * parsers.keyword
9732 * parsers.html_tag_end
9733
9734 parsers.html_close_special_tag = parsers.html_tag_closing_start
9735 * parsers.special_block_keyword
9736 * parsers.html_tag_end
9737
9738 -- empty tags
9739 parsers.html_any_empty_inline_tag = parsers.html_tag_start
9740 * parsers.keyword
9741 * parsers.html_attribute^0
9742 * parsers.html_empty_tag_end
9743
9744 parsers.html_any_empty_tag = parsers.html_tag_start
9745 * parsers.keyword
9746 * parsers.html_non_newline_attribute^0
9747 * parsers.optionalspace
9748 * parsers.slash
9749 * parsers.more
9750
9751 parsers.html_empty_tag = parsers.html_tag_start
9752 * parsers.block_keyword
9753 * parsers.html_attribute^0
9754 * parsers.html_empty_tag_end
9755
9756 parsers.html_empty_special_tag = parsers.html_tag_start
9757 * parsers.special_block_keyword
9758 * parsers.html_attribute^0
9759 * parsers.html_empty_tag_end
9760

```

```

9761 parsers.html_incomplete_blocks
9762     = parsers.html_incomplete_open_tag
9763     + parsers.html_incomplete_open_special_tag
9764     + parsers.html_incomplete_close_tag
9765
9766 -- parse special html blocks
9767 parsers.html_blankline_ending_special_block_opening
9768     = ( parsers.html_close_special_tag
9769       + parsers.html_empty_special_tag)
9770     * #( parsers.optionalspace
9771         * (parsers.newline + parsers.eof))
9772
9773 parsers.html_blankline_ending_special_block
9774     = parsers.html_blankline_ending_special_block_opening
9775     * parsers.html_blankline_end_condition
9776
9777 parsers.html_special_block_opening
9778     = parsers.html_incomplete_open_special_tag
9779     - parsers.html_empty_special_tag
9780
9781 parsers.html_closing_special_block
9782     = parsers.html_special_block_opening
9783     * parsers.html_until_end(parsers.html_close_special_tag)
9784
9785 parsers.html_special_block
9786     = parsers.html_blankline_ending_special_block
9787     + parsers.html_closing_special_block
9788
9789 -- parse html blocks
9790 parsers.html_block_opening = parsers.html_incomplete_open_tag
9791                          + parsers.html_incomplete_close_tag
9792
9793 parsers.html_block = parsers.html_block_opening
9794                  * parsers.html_blankline_end_condition
9795
9796 -- parse any html blocks
9797 parsers.html_any_block_opening
9798     = ( parsers.html_any_open_tag
9799       + parsers.html_any_close_tag
9800       + parsers.html_any_empty_tag)
9801     * #(parsers.optionalspace * (parsers.newline + parsers.eof))
9802
9803 parsers.html_any_block = parsers.html_any_block_opening
9804                       * parsers.html_blankline_end_condition
9805
9806 parsers.html_inline_comment_full = parsers.html_comment_start
9807                               * -P(">") * -P("->")

```

```

9808             * Cs(( V("NoSoftLineBreakEndline")
9809                 + parsers.any - P("---")
9810                 - parsers.nested_breaking_blank
9811                 - parsers.html_comment_end)^0)
9812             * parsers.html_comment_end
9813
9814 parsers.html_inline_tags = parsers.html_inline_comment_full
9815                 + parsers.html_any_empty_inline_tag
9816                 + parsers.html_inline_instruction
9817                 + parsers.html_inline_cdatasection
9818                 + parsers.html_inline_declaration
9819                 + parsers.html_any_open_inline_tag
9820                 + parsers.html_any_close_tag
9821

```

### 3.1.5.8 Parsers Used for Markdown Tags and Links

```

9822 parsers.urlchar = parsers.anyescaped
9823                 - parsers.newline
9824                 - parsers.more
9825
9826 parsers.auto_link_scheme_part = parsers.alphanumeric
9827                 + parsers.plus
9828                 + parsers.period
9829                 + parsers.dash
9830
9831 parsers.auto_link_scheme = parsers.letter
9832                 * parsers.auto_link_scheme_part
9833                 * parsers.auto_link_scheme_part^-30
9834
9835 parsers.absolute_uri = parsers.auto_link_scheme * parsers.colon
9836                 * ( parsers.any - parsers.spacing
9837                 - parsers.less - parsers.more)^0
9838
9839 parsers.printable_characters = S(" !#$%&'*/=?^_`{|}~-")
9840
9841 parsers.email_address_local_part_char = parsers.alphanumeric
9842                 + parsers.printable_characters
9843
9844 parsers.email_address_local_part
9845     = parsers.email_address_local_part_char^1
9846
9847 parsers.email_address_dns_label = parsers.alphanumeric
9848                 * ( parsers.alphanumeric
9849                 + parsers.dash)^-62
9850                 * B(parsers.alphanumeric)
9851

```

```

9852 parsers.email_address_domain = parsers.email_address_dns_label
9853                                * ( parsers.period
9854                                * parsers.email_address_dns_label)^0
9855
9856 parsers.email_address = parsers.email_address_local_part
9857                        * parsers.at
9858                        * parsers.email_address_domain
9859
9860 parsers.auto_link_url = parsers.less
9861                        * C(parsers.absolute_uri)
9862                        * parsers.more
9863
9864 parsers.auto_link_email = parsers.less
9865                        * C(parsers.email_address)
9866                        * parsers.more
9867
9868 parsers.auto_link_relative_reference = parsers.less
9869                                    * C(parsers.urlchar^1)
9870                                    * parsers.more
9871
9872 parsers.autolink = parsers.auto_link_url
9873                 + parsers.auto_link_email
9874
9875 -- content in balanced brackets, parentheses, or quotes:
9876 parsers.bracketed = P{ parsers.lbracket
9877                        * (( parsers.backslash / "\"" * parsers.rbracket
9878                        + parsers.any - (parsers.lbracket
9879                                    + parsers.rbracket
9880                                    + parsers.blankline^2)
9881                        ) + V(1))^0
9882                        * parsers.rbracket }
9883
9884 parsers.inparens = P{ parsers.lparent
9885                        * ((parsers.anyescaped - (parsers.lparent
9886                                    + parsers.rparent
9887                                    + parsers.blankline^2)
9888                        ) + V(1))^0
9889                        * parsers.rparent }
9890
9891 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
9892                        * ((parsers.anyescaped - (parsers.squote
9893                                    + parsers.blankline^2)
9894                        ) + V(1))^0
9895                        * parsers.squote }
9896
9897 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
9898                        * ((parsers.anyescaped - (parsers.dquote

```

```

9899                                     + parsers.blankline^2)
9900                                     ) + V(1))^0
9901                                     * parsers.dquote }
9902
9903 parsers.link_text = parsers.lbracket
9904                       * Cs((parsers.alphanumeric^1
9905                           + parsers.bracketed
9906                           + parsers.inticks
9907                           + parsers.autolink
9908                           + V("InlineHtml")
9909                           + ( parsers.backslash * parsers.backslash)
9910                           + ( parsers.backslash
9911                               * ( parsers.lbracket
9912                                   + parsers.rbracket)
9913                               + V("Space")
9914                               + V("NoSoftLineBreakEndline")
9915                               + (parsers.any
9916                                   - ( parsers.newline
9917                                       + parsers.lbracket
9918                                       + parsers.rbracket
9919                                       + parsers.blankline^2))))^0)
9920                       * parsers.rbracket
9921
9922 parsers.link_label_body = -(parsers.sp * parsers.rbracket)
9923                       * #( ( parsers.any
9924                           - parsers.rbracket)^~999
9925                           * parsers.rbracket)
9926                       * Cs((parsers.alphanumeric^1
9927                           + parsers.inticks
9928                           + parsers.autolink
9929                           + V("InlineHtml")
9930                           + ( parsers.backslash * parsers.backslash)
9931                           + ( parsers.backslash
9932                               * ( parsers.lbracket
9933                                   + parsers.rbracket)
9934                               + V("Space")
9935                               + V("NoSoftLineBreakEndline")
9936                               + (parsers.any
9937                                   - ( parsers.newline
9938                                       + parsers.lbracket
9939                                       + parsers.rbracket
9940                                       + parsers.blankline^2))))^1)
9941
9942 parsers.link_label = parsers.lbracket
9943                       * parsers.link_label_body
9944                       * parsers.rbracket
9945

```

```

9946 parsers.inparens_url = P{ parsers.lparent
9947                        * ((parsers.anyescaped - (parsers.lparent
9948   + parsers.rparent
9949   + parsers.spacing)
9950                          ) + V(1))^0
9951                        * parsers.rparent }
9952
9953 -- url for markdown links, allowing nested brackets:
9954 parsers.url           = parsers.less * Cs((parsers.anyescaped
9955   - parsers.newline
9956   - parsers.less
9957   - parsers.more)^0)
9958                               * parsers.more
9959 + -parsers.less
9960 * Cs((parsers.inparens_url + (parsers.anyescaped
9961                               - parsers.spacing
9962                               - parsers.lparent
9963                               - parsers.rparent))^1)
9964
9965 -- quoted text:
9966 parsers.title_s       = parsers.squote
9967                        * Cs((parsers.html_entities
9968                              + V("Space")
9969                              + V("NoSoftLineBreakEndline")
9970                              + ( parsers.anyescaped
9971                                - parsers.newline
9972                                - parsers.squote
9973                                - parsers.blankline^2))^0)
9974                        * parsers.squote
9975
9976 parsers.title_d       = parsers.dquote
9977                        * Cs((parsers.html_entities
9978                              + V("Space")
9979                              + V("NoSoftLineBreakEndline")
9980                              + ( parsers.anyescaped
9981                                - parsers.newline
9982                                - parsers.dquote
9983                                - parsers.blankline^2))^0)
9984                        * parsers.dquote
9985
9986 parsers.title_p       = parsers.lparent
9987                        * Cs((parsers.html_entities
9988                              + V("Space")
9989                              + V("NoSoftLineBreakEndline")
9990                              + ( parsers.anyescaped
9991                                - parsers.newline
9992                                - parsers.lparent

```



```

9993             - parsers.rparent
9994             - parsers.blankline^2))^0)
9995         * parsers.rparent
9996
9997 parsers.title
9998     = parsers.title_d + parsers.title_s + parsers.title_p
9999
10000 parsers.optionaltitle
10001     = parsers.spnlc * parsers.title * parsers.spacechar^0 + Cc("")
10002

```

### 3.1.5.9 Helpers for Links and Link Reference Definitions

```

10003 -- parse a reference definition: [foo]: /bar "title"
10004 parsers.define_reference_parser = (parsers.check_trail / "")
10005                                 * parsers.link_label * parsers.colon
10006                                 * parsers.spnlc * parsers.url
10007                                 * ( parsers.spnlc_sep * parsers.title
10008                                   * parsers.only_blank
10009                                   + Cc("") * parsers.only_blank)

```

### 3.1.5.10 Inline Elements

```

10010 parsers.Inline          = V("Inline")
10011
10012 -- parse many p between starter and ender
10013 parsers.between = function(p, starter, ender)
10014     local ender2 = B(parsers.nonspacechar) * ender
10015     return ( starter
10016             * #parsers.nonspacechar
10017             * Ct(p * (p - ender2)^0)
10018             * ender2)
10019 end
10020

```

### 3.1.5.11 Block Elements

```

10021 parsers.lineof = function(c)
10022     return ( parsers.check_trail_no_rem
10023             * (P(c) * parsers.optionalspace)^3
10024             * (parsers.newline + parsers.eof))
10025 end
10026
10027 parsers.thematic_break_lines = parsers.lineof(parsers.asterisk)
10028                               + parsers.lineof(parsers.dash)
10029                               + parsers.lineof(parsers.underscore)

```

### 3.1.5.12 Headings

```
10030 -- parse Atx heading start and return level
10031 parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
10032                      * -parsers.hash / length
10033
10034 -- parse setext header ending and return level
10035 parsers.heading_level
10036   = parsers.nonindentSPACE * parsers.equal^1
10037   * parsers.optionalspace * #parsers.newline * Cc(1)
10038   + parsers.nonindentSPACE * parsers.dash^1
10039   * parsers.optionalspace * #parsers.newline * Cc(2)
10040
10041 local function strip_atx_end(s)
10042   return s:gsub("%s+#+%s*\n$", "")
10043 end
10044
10045 parsers.atx_heading = parsers.check_trail_no_rem
10046                      * Cg(parsers.heading_start, "level")
10047                      * (C( parsers.optionalspace
10048                          * parsers.hash^0
10049                          * parsers.optionalspace
10050                          * parsers.newline)
10051                      + parsers.spacechar^1
10052                      * C(parsers.line))
```

### 3.1.6 Markdown Reader

This section documents the `reader` object, which implements the routines for parsing the markdown input. The object corresponds to the markdown reader object that was located in the `lunamark/reader/markdown.lua` file in the Lunamark Lua module.

The `reader.new` method creates and returns a new T<sub>E</sub>X reader object associated with the Lua interface options (see Section 2.1.3) `options` and with a writer object `writer`. When `options` are unspecified, it is assumed that an empty table was passed to the method.

The objects produced by the `reader.new` method expose instance methods and variables of their own. As a convention, I will refer to these *member*s as `reader->member`.

```
10053 M.reader = {}
10054 function M.reader.new(writer, options)
10055   local self = {}
```

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

```
10056   self.writer = writer
10057   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.

```
10058  self.parsers = {}
10059  (function(parsers)
10060    setmetatable(self.parsers, {
10061      __index = function (_, key)
10062        return parsers[key]
10063      end
10064    })
10065  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.

```
10066  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.

```
10067  function self.normalize_tag(tag)
10068    tag = util.rope_to_string(tag)
10069    tag = tag:gsub("[ \n\r\t]+", " ")
10070    tag = tag:gsub("^ ", ""):gsub(" $", "")
10071    local form = nil
10072    if options.unicodeNormalization then
10073      form = options.unicodeNormalizationForm
10074    end
10075    tag = util.casefold(tag, form)
10076    return tag
10077  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.

```
10078  local function iterlines(s, f)
10079    local rope = lpeg.match(Ct((parsers.line / f)^1), s)
10080    return util.rope_to_string(rope)
10081  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.

```
10082  if options.preserveTabs then
10083    self.expandtabs = function(s) return s end
10084  else
10085    self.expandtabs = function(s)
10086      if s:find("\t") then
```

```

10087             return iterlines(s, util.expand_tabs_in_line)
10088         else
10089             return s
10090         end
10091     end
10092 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.

```

10093 self.parser_functions = {}
10094 self.create_parser = function(name, grammar, topLevel)
10095     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.

```

10096     if topLevel and options.stripIndent then
10097         local min_prefix_length, min_prefix = nil, ''
10098         str = iterlines(str, function(line)
10099             if lpeg.match(parsers.nonemptyline, line) == nil then
10100                 return line
10101             end
10102             line = util.expand_tabs_in_line(line)
10103             local prefix = lpeg.match(C(parsers.optionalspace), line)
10104             local prefix_length = #prefix
10105             local is_shorter = min_prefix_length == nil
10106             if not is_shorter then
10107                 is_shorter = prefix_length < min_prefix_length
10108             end
10109             if is_shorter then
10110                 min_prefix_length, min_prefix = prefix_length, prefix
10111             end
10112             return line
10113         end)
10114         str = str:gsub('^' .. min_prefix, '')
10115     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.

```

10116     if topLevel and (options.texComments or options.hybrid) then

```

```

10117         str = lpeg.match(Ct(parsers.commented_line^1), str)
10118         str = util.ropetostring(str)
10119     end
10120     local res = lpeg.match(grammar(), str)
10121     if res == nil then
10122         return writer.error(
10123             format("Parser `%s` failed to process the input text.", name),
10124             format("Here are the first 20 characters of the remaining "
10125                 .. "unprocessed text: `%s`.", str:sub(1,20))
10126         )
10127     else
10128         return res
10129     end
10130 end
10131 end
10132
10133 self.create_parser("parse_blocks",
10134     function()
10135         return parsers.blocks
10136     end, true)
10137
10138 self.create_parser("parse_blocks_nested",
10139     function()
10140         return parsers.blocks_nested
10141     end, false)
10142
10143 self.create_parser("parse_inlines",
10144     function()
10145         return parsers.inlines
10146     end, false)
10147
10148 self.create_parser("parse_inlines_no_inline_note",
10149     function()
10150         return parsers.inlines_no_inline_note
10151     end, false)
10152
10153 self.create_parser("parse_inlines_no_html",
10154     function()
10155         return parsers.inlines_no_html
10156     end, false)
10157
10158 self.create_parser("parse_inlines_nbsp",
10159     function()
10160         return parsers.inlines_nbsp
10161     end, false)
10162
10163 self.create_parser("parse_inlines_no_link_or_emphasis",

```

```

10164         function()
10165             return parsers.inlines_no_link_or_emphasis
10166         end, false)
10167
10168     self.create_parser("parse_inlines_identity",
10169         function()
10170             return parsers.inlines_identity
10171         end, false)
10172
10173     self.create_parser("parse_inlines_string",
10174         function()
10175             return parsers.inlines_string
10176         end, false)
10177
10178     self.create_parser("parse_inlines_math",
10179         function()
10180             return parsers.inlines_math
10181         end, false)

```

### 3.1.6.3 Parsers Used for Indentation (local)

The following patterns represent basic building blocks of indented content.

```

10182     parsers.minimally_indented_blankline
10183         = parsers.check_minimal_indent * (parsers.blankline / "")
10184
10185     parsers.minimally_indented_block
10186         = parsers.check_minimal_indent * V("Block")
10187
10188     parsers.minimally_indented_block_or_paragraph
10189         = parsers.check_minimal_indent * V("BlockOrParagraph")
10190
10191     parsers.minimally_indented_paragraph
10192         = parsers.check_minimal_indent * V("Paragraph")
10193
10194     parsers.minimally_indented_plain
10195         = parsers.check_minimal_indent * V("Plain")
10196
10197     parsers.minimally_indented_par_or_plain
10198         = parsers.minimally_indented_paragraph
10199         + parsers.minimally_indented_plain
10200
10201     parsers.minimally_indented_par_or_plain_no_blank
10202         = parsers.minimally_indented_par_or_plain
10203         - parsers.minimally_indented_blankline
10204
10205     parsers.minimally_indented_ref
10206         = parsers.check_minimal_indent * V("Reference")

```

```

10207
10208 parsers.minimally_indented_blank
10209     = parsers.check_minimal_indent * V("Blank")
10210
10211 parsers.conditionally_indented_blankline
10212     = parsers.check_minimal_blank_indent * (parsers.blankline / "")
10213
10214 parsers.minimally_indented_ref_or_block
10215     = parsers.minimally_indented_ref
10216     + parsers.minimally_indented_block
10217     - parsers.minimally_indented_blankline
10218
10219 parsers.minimally_indented_ref_or_block_or_par
10220     = parsers.minimally_indented_ref
10221     + parsers.minimally_indented_block_or_paragraph
10222     - parsers.minimally_indented_blankline
10223

```

The following pattern parses the properly indented content that follows the initial container start.

```

10224
10225 function parsers.separator_loop(separated_block, paragraph,
10226                                block_separator, paragraph_separator)
10227     return separated_block
10228         + block_separator
10229         * paragraph
10230         * separated_block
10231         + paragraph_separator
10232         * paragraph
10233 end
10234
10235 function parsers.create_loop_body_pair(separated_block, paragraph,
10236                                       block_separator,
10237                                       paragraph_separator)
10238     return {
10239         block = parsers.separator_loop(separated_block, paragraph,
10240                                       block_separator, block_separator),
10241         par = parsers.separator_loop(separated_block, paragraph,
10242                                     block_separator, paragraph_separator)
10243     }
10244 end
10245
10246 parsers.block_sep_group = function(blank)
10247     return blank^0 * parsers.eof
10248         + ( blank^2 / writer.paragraphsep
10249           + blank^0 / writer.interblocksep
10250           )

```

```

10251 end
10252
10253 parsers.par_sep_group = function(blank)
10254     return blank^0 * parsers.eof
10255         + blank^0 / writer.paragraphsep
10256 end
10257
10258 parsers.sep_group_no_output = function(blank)
10259     return blank^0 * parsers.eof
10260         + blank^0
10261 end
10262
10263 parsers.content_blank = parsers.minimally_indented_blankline
10264
10265 parsers.ref_or_block_separated
10266     = parsers.sep_group_no_output(parsers.content_blank)
10267     * ( parsers.minimally_indented_ref
10268         - parsers.content_blank)
10269     + parsers.block_sep_group(parsers.content_blank)
10270     * ( parsers.minimally_indented_block
10271         - parsers.content_blank)
10272
10273 parsers.loop_body_pair =
10274     parsers.create_loop_body_pair(
10275         parsers.ref_or_block_separated,
10276         parsers.minimally_indented_par_or_plain_no_blank,
10277         parsers.block_sep_group(parsers.content_blank),
10278         parsers.par_sep_group(parsers.content_blank))
10279
10280 parsers.content_loop = ( V("Block")
10281                         * parsers.loop_body_pair.block^0
10282                         + (V("Paragraph") + V("Plain")))
10283                         * parsers.ref_or_block_separated
10284                         * parsers.loop_body_pair.block^0
10285                         + (V("Paragraph") + V("Plain")))
10286                         * parsers.loop_body_pair.par^0)
10287                         * parsers.content_blank^0
10288
10289 parsers.indented_content = function()
10290     return Ct( (V("Reference") + (parsers.blankline / ""))
10291               * parsers.content_blank^0
10292               * parsers.check_minimal_indent
10293               * parsers.content_loop
10294               + (V("Reference") + (parsers.blankline / ""))
10295               * parsers.content_blank^0
10296               + parsers.content_loop)
10297 end

```



```

10298
10299 parsers.add_indent = function(pattern, name, breakable)
10300     return Cg(Cmt( Cb("indent_info")
10301         * Ct(pattern)
10302         * ( #parsers.linechar -- check if starter is blank
10303         * Cc(false) + Cc(true))
10304         * Cc(name)
10305         * Cc(breakable),
10306         process_starter_indent), "indent_info")
10307 end
10308

```

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

```

10309 if options.hashEnumerators then
10310     parsers.dig = parsers.digit + parsers.hash
10311 else
10312     parsers.dig = parsers.digit
10313 end
10314
10315 parsers.enumerator = function(delimiter_type, interrupting)
10316     local delimiter_range
10317     local allowed_end
10318     if interrupting then
10319         delimiter_range = P("1")
10320         allowed_end = C(parsers.spacechar^1) * #parsers.linechar
10321     else
10322         delimiter_range = parsers.dig * parsers.dig^-8
10323         allowed_end = C(parsers.spacechar^1)
10324             + #(parsers.newline + parsers.eof)
10325     end
10326
10327     return parsers.check_trail
10328         * Ct(C(delimiter_range) * C(delimiter_type))
10329         * allowed_end
10330 end
10331
10332 parsers.starter = parsers.bullet(parsers.dash)
10333     + parsers.bullet(parsers.asterisk)
10334     + parsers.bullet(parsers.plus)
10335     + parsers.enumerator(parsers.period)
10336     + parsers.enumerator(parsers.rparent)
10337

```

#### 3.1.6.5 Parsers Used for Blockquotes (local)

```

10338 parsers.blockquote_start
10339     = parsers.check_trail

```

```

10340     * C(parsers.more)
10341     * C(parsers.spacechar^0)
10342
10343     parsers.blockquote_body
10344     = parsers.add_indent(parsers.blockquote_start, "bq", true)
10345     * parsers.indented_content()
10346     * remove_indent("bq")
10347
10348     if not options.breakableBlockquotes then
10349         parsers.blockquote_body
10350         = parsers.add_indent(parsers.blockquote_start, "bq", false)
10351         * parsers.indented_content()
10352         * remove_indent("bq")
10353     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.

```

10354     local function parse_content_part(content_part)
10355         local rope = util.rope_to_string(content_part)
10356         local parsed
10357         = self.parser_functions.parse_inlines_no_link_or_emphasis(rope)
10358         parsed.indent_info = nil
10359         return parsed
10360     end
10361

```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

10362     local collect_emphasis_content =
10363     function(t, opening_index, closing_index)
10364         local content = {}
10365
10366         local content_part = {}
10367         for i = opening_index, closing_index do
10368             local value = t[i]
10369
10370             if value.rendered ~= nil then
10371                 content[#content + 1] = parse_content_part(content_part)
10372                 content_part = {}
10373                 content[#content + 1] = value.rendered
10374                 value.rendered = nil
10375             else
10376                 if value.warning ~= nil then
10377                     if next(content_part) ~= nil then
10378                         content[#content + 1] = parse_content_part(content_part)
10379                     end

```

```

10380         content_part = {}
10381
10382         content[#content + 1] = value.warning
10383         value.warning = nil
10384     end
10385     if value.type == "delimiter"
10386         and value.element == "emphasis" then
10387         if value.is_active then
10388             content_part[#content_part + 1]
10389                 = string.rep(value.character, value.current_count)
10390         end
10391     else
10392         content_part[#content_part + 1] = value.content
10393     end
10394     value.content = ''
10395     value.is_active = false
10396 end
10397 end
10398
10399 if next(content_part) ~= nil then
10400     content[#content + 1] = parse_content_part(content_part)
10401 end
10402
10403 return content
10404 end
10405

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as emphasis.

```

10406 local function fill_emph(t, opening_index, closing_index)
10407     local content
10408     = collect_emphasis_content(t, opening_index + 1,
10409                               closing_index - 1)
10410     t[opening_index + 1].is_active = true
10411     t[opening_index + 1].rendered = writer.emphasis(content)
10412 end
10413

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as strong emphasis.

```

10414 local function fill_strong(t, opening_index, closing_index)
10415     local content
10416     = collect_emphasis_content(t, opening_index + 1,
10417                               closing_index - 1)
10418     t[opening_index + 1].is_active = true
10419     t[opening_index + 1].rendered = writer.strong(content)
10420 end
10421

```

Check whether the opening delimiter `opening_delimiter` and closing delimiter `closing_delimiter` break rule three together.

```

10422 local function breaks_three_rule(opening_delimiter, closing_delimiter)
10423     return ( opening_delimiter.is_closing
10424             or closing_delimiter.is_opening)
10425             and (( opening_delimiter.original_count
10426                   + closing_delimiter.original_count) % 3 == 0)
10427             and ( opening_delimiter.original_count % 3 ~= 0
10428                 or closing_delimiter.original_count % 3 ~= 0)
10429     end
10430

```

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`.

```

10431 local find_emphasis_opener = function(t, bottom_index, latest_index,
10432                                     character, closing_delimiter)
10433     for i = latest_index, bottom_index, -1 do
10434         local value = t[i]
10435         if value.is_active and
10436            value.is_opening and
10437            value.type == "delimiter" and
10438            value.element == "emphasis" and
10439            (value.character == character) and
10440            (value.current_count > 0) then
10441             if not breaks_three_rule(value, closing_delimiter) then
10442                 return i
10443             end
10444         end
10445     end
10446 end
10447

```

Iterate over the delimiters in the delimiter table `t`, producing emphasis or strong emphasis macros.

```

10448 local function process_emphasis(t, opening_index, closing_index)
10449     for i = opening_index, closing_index do
10450         local value = t[i]
10451         if value.type == "delimiter" and value.element == "emphasis" then
10452             local delimiter_length = string.len(value.content)
10453             value.character = string.sub(value.content, 1, 1)
10454             value.current_count = delimiter_length
10455             value.original_count = delimiter_length
10456         end
10457     end
10458
10459     local openers_bottom = {

```

```

10460     ['*'] = {
10461         [true] = {opening_index, opening_index, opening_index},
10462         [false] = {opening_index, opening_index, opening_index}
10463     },
10464     ['_'] = {
10465         [true] = {opening_index, opening_index, opening_index},
10466         [false] = {opening_index, opening_index, opening_index}
10467     }
10468 }
10469
10470 local current_position = opening_index
10471 local max_position = closing_index
10472
10473 while current_position <= max_position do
10474     local value = t[current_position]
10475
10476     if value.type ~= "delimiter" or
10477        value.element ~= "emphasis" or
10478        not value.is_active or
10479        not value.is_closing or
10480        (value.current_count <= 0) then
10481         current_position = current_position + 1
10482         goto continue
10483     end
10484
10485     local character = value.character
10486     local is_opening = value.is_opening
10487     local closing_length_modulo_three = value.original_count % 3
10488
10489     local current_openers_bottom
10490     = openers_bottom[character][is_opening]
10491       [closing_length_modulo_three + 1]
10492
10493     local opener_position
10494     = find_emphasis_opener(t, current_openers_bottom,
10495                           current_position - 1, character, value)
10496
10497     if (opener_position == nil) then
10498         openers_bottom[character][is_opening]
10499           [closing_length_modulo_three + 1]
10500           = current_position
10501         current_position = current_position + 1
10502         goto continue
10503     end
10504
10505     local opening_delimiter = t[opener_position]
10506

```

```

10507     local current_opening_count = opening_delimiter.current_count
10508     local current_closing_count = t[current_position].current_count
10509
10510     if (current_opening_count >= 2)
10511         and (current_closing_count >= 2) then
10512         opening_delimiter.current_count = current_opening_count - 2
10513         t[current_position].current_count = current_closing_count - 2
10514         fill_strong(t, opener_position, current_position)
10515     else
10516         opening_delimiter.current_count = current_opening_count - 1
10517         t[current_position].current_count = current_closing_count - 1
10518         fill_emph(t, opener_position, current_position)
10519     end
10520
10521     ::continue::
10522 end
10523 end
10524
10525 parsers.delimiter_run = function(character)
10526     return (B(parsers.backslash * character) + -B(character))
10527         * character~1
10528         * -#character
10529 end
10530
10531 parsers.left_flanking_delimiter_run = function(character)
10532     return (B( parsers.any)
10533         * ( parsers.unicode.preceding_punctuation
10534           + parsers.unicode.preceding_whitespace)
10535         + -B(parsers.any))
10536         * parsers.delimiter_run(character)
10537         * #parsers.unicode.punctuation
10538         + parsers.delimiter_run(character)
10539         * -( #parsers.unicode.punctuation
10540           + #parsers.unicode.whitespace
10541           + parsers.eof)
10542 end
10543
10544 parsers.right_flanking_delimiter_run = function(character)
10545     return parsers.unicode.preceding_punctuation
10546         * parsers.delimiter_run(character)
10547         * ( #parsers.unicode.punctuation
10548           + #parsers.unicode.whitespace
10549           + parsers.eof)
10550         + (B(parsers.any)
10551           * -( parsers.unicode.preceding_punctuation
10552             + parsers.unicode.preceding_whitespace))
10553         * parsers.delimiter_run(character)

```

```

10554 end
10555
10556 if options.underscores then
10557     parsers.emph_start
10558         = parsers.left_flanking_delimiter_run(parsers.asterisk)
10559         + ( -#parsers.right_flanking_delimiter_run(parsers.underscore)
10560             + ( parsers.unicode.preceding_punctuation
10561                 * #parsers.right_flanking_delimiter_run(parsers.underscore)))
10562         * parsers.left_flanking_delimiter_run(parsers.underscore)
10563
10564     parsers.emph_end
10565         = parsers.right_flanking_delimiter_run(parsers.asterisk)
10566         + ( -#parsers.left_flanking_delimiter_run(parsers.underscore)
10567             + #( parsers.left_flanking_delimiter_run(parsers.underscore)
10568                 * #parsers.unicode.punctuation))
10569         * parsers.right_flanking_delimiter_run(parsers.underscore)
10570 else
10571     parsers.emph_start
10572         = parsers.left_flanking_delimiter_run(parsers.asterisk)
10573
10574     parsers.emph_end
10575         = parsers.right_flanking_delimiter_run(parsers.asterisk)
10576 end
10577
10578 parsers.emph_capturing_open_and_close
10579     = #parsers.emph_start * #parsers.emph_end
10580     * Ct( Cg(Cc("delimiter"), "type")
10581           * Cg(Cc("emphasis"), "element")
10582           * Cg(C(parsers.emph_start), "content")
10583           * Cg(Cc(true), "is_opening")
10584           * Cg(Cc(true), "is_closing"))
10585
10586 parsers.emph_capturing_open = Ct( Cg(Cc("delimiter"), "type")
10587                                   * Cg(Cc("emphasis"), "element")
10588                                   * Cg(C(parsers.emph_start), "content")
10589                                   * Cg(Cc(true), "is_opening")
10590                                   * Cg(Cc(false), "is_closing"))
10591
10592 parsers.emph_capturing_close = Ct( Cg(Cc("delimiter"), "type")
10593                                    * Cg(Cc("emphasis"), "element")
10594                                    * Cg(C(parsers.emph_end), "content")
10595                                    * Cg(Cc(false), "is_opening")
10596                                    * Cg(Cc(true), "is_closing"))
10597
10598 parsers.emph_open_or_close = parsers.emph_capturing_open_and_close
10599                             + parsers.emph_capturing_open
10600                             + parsers.emph_capturing_close

```

```

10601
10602 parsers.emph_open = parsers.emph_capturing_open_and_close
10603                     + parsers.emph_capturing_open
10604
10605 parsers.emph_close = parsers.emph_capturing_open_and_close
10606                     + parsers.emph_capturing_close
10607

```

### 3.1.6.7 Helpers for Links and Link Reference Definitions (local)

```

10608 -- List of references defined in the document
10609 local references
10610
10611 -- List of note references defined in the document
10612 parsers.rawnotes = {}
10613

```

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.

```

10614 function self.register_link(_, tag, url, title,
10615                             attributes)
10616     local normalized_tag = self.normalize_tag(tag)
10617     if references[normalized_tag] == nil then
10618         references[normalized_tag] = {
10619             url = url,
10620             title = title,
10621             attributes = attributes
10622         }
10623     return ""
10624 else
10625     local text
10626     = string.format('Multiply defined link reference "%s"', tag)
10627     local more = string.format("Look for the text `[%s]: ...`.", tag)
10628     return writer.warning(text, more)
10629 end
10630 end
10631

```

The `reader->lookup_reference` method looks up a reference with link label `tag`.

```

10632 function self.lookup_reference(tag)
10633     return references[self.normalize_tag(tag)]
10634 end
10635

```

The `reader->lookup_note_reference` method looks up a note reference with label `tag`.

```

10636 function self.lookup_note_reference(tag)

```



```

10637     return parsers.rawnotes[self.normalize_tag(tag)]
10638 end
10639
10640 parsers.title_s_direct_ref = parsers.squote
10641     * Cs((parsers.html_entities
10642         + ( parsers.anyescaped
10643           - parsers.squote
10644           - parsers.blankline^2))^0)
10645     * parsers.squote
10646
10647 parsers.title_d_direct_ref = parsers.dquote
10648     * Cs((parsers.html_entities
10649         + ( parsers.anyescaped
10650           - parsers.dquote
10651           - parsers.blankline^2))^0)
10652     * parsers.dquote
10653
10654 parsers.title_p_direct_ref = parsers.lparent
10655     * Cs((parsers.html_entities
10656         + ( parsers.anyescaped
10657           - parsers.lparent
10658           - parsers.rparent
10659           - parsers.blankline^2))^0)
10660     * parsers.rparent
10661
10662 parsers.title_direct_ref = parsers.title_s_direct_ref
10663     + parsers.title_d_direct_ref
10664     + parsers.title_p_direct_ref
10665
10666 parsers.inline_direct_ref_inside = parsers.lparent * parsers.spnl
10667     * Cg(parsers.url + Cc(""), "url")
10668     * parsers.spnl
10669     * Cg( parsers.title_direct_ref
10670         + Cc(""), "title")
10671     * parsers.spnl * parsers.rparent
10672
10673 parsers.inline_direct_ref = parsers.lparent * parsers.spnlc
10674     * Cg(parsers.url + Cc(""), "url")
10675     * parsers.spnlc
10676     * Cg(parsers.title + Cc(""), "title")
10677     * parsers.spnlc * parsers.rparent
10678
10679 parsers.empty_link = parsers.lbracket
10680     * parsers.rbracket
10681
10682 parsers.inline_link = parsers.link_text
10683     * parsers.inline_direct_ref

```

```

10684
10685 parsers.full_link = parsers.link_text
10686                     * parsers.link_label
10687
10688 parsers.shortcut_link = parsers.link_label
10689                     * -(parsers.empty_link + parsers.link_label)
10690
10691 parsers.collapsed_link = parsers.link_label
10692                     * parsers.empty_link
10693
10694 parsers.image_opening = #(parsers.exclamation * parsers.inline_link)
10695                     * Cg(Cc("inline"), "link_type")
10696                     + #(parsers.exclamation * parsers.full_link)
10697                     * Cg(Cc("full"), "link_type")
10698                     + #( parsers.exclamation
10699                     * parsers.collapsed_link)
10700                     * Cg(Cc("collapsed"), "link_type")
10701                     + #(parsers.exclamation * parsers.shortcut_link)
10702                     * Cg(Cc("shortcut"), "link_type")
10703                     + #(parsers.exclamation * parsers.empty_link)
10704                     * Cg(Cc("empty"), "link_type")
10705
10706 parsers.link_opening = #parsers.inline_link
10707                     * Cg(Cc("inline"), "link_type")
10708                     + #parsers.full_link
10709                     * Cg(Cc("full"), "link_type")
10710                     + #parsers.collapsed_link
10711                     * Cg(Cc("collapsed"), "link_type")
10712                     + #parsers.shortcut_link
10713                     * Cg(Cc("shortcut"), "link_type")
10714                     + #parsers.empty_link
10715                     * Cg(Cc("empty_link"), "link_type")
10716                     + #parsers.link_text
10717                     * Cg(Cc("link_text"), "link_type")
10718
10719 parsers.note_opening = #(parsers.circumflex * parsers.link_text)
10720                     * Cg(Cc("note_inline"), "link_type")
10721
10722 parsers.raw_note_opening = #( parsers.lbracket
10723                     * parsers.circumflex
10724                     * parsers.link_label_body
10725                     * parsers.rbracket)
10726                     * Cg(Cc("raw_note"), "link_type")
10727
10728 local inline_note_element = Cg(Cc("note"), "element")
10729                     * parsers.note_opening
10730                     * Cg( parsers.circumflex

```

```

10731             * parsers.lbracket, "content")
10732
10733     local image_element = Cg(Cc("image"), "element")
10734         * parsers.image_opening
10735         * Cg( parsers.exclamation
10736             * parsers.lbracket, "content")
10737
10738     local note_element  = Cg(Cc("note"), "element")
10739         * parsers.raw_note_opening
10740         * Cg( parsers.lbracket
10741             * parsers.circumflex, "content")
10742
10743     local link_element  = Cg(Cc("link"), "element")
10744         * parsers.link_opening
10745         * Cg(parsers.lbracket, "content")
10746
10747     local opening_elements = parsers.fail
10748
10749     if options.inlineNotes then
10750         opening_elements = opening_elements + inline_note_element
10751     end
10752
10753     opening_elements = opening_elements + image_element
10754
10755     if options.notes then
10756         opening_elements = opening_elements + note_element
10757     end
10758
10759     opening_elements = opening_elements + link_element
10760
10761     parsers.link_image_opening = Ct( Cg(Cc("delimiter"), "type")
10762         * Cg(Cc(true), "is_opening")
10763         * Cg(Cc(false), "is_closing")
10764         * opening_elements)
10765
10766     parsers.link_image_closing = Ct( Cg(Cc("delimiter"), "type")
10767         * Cg(Cc("link"), "element")
10768         * Cg(Cc(false), "is_opening")
10769         * Cg(Cc(true), "is_closing")
10770         * ( Cg(Cc(true), "is_direct")
10771             * Cg( parsers.rbracket
10772                 * #parsers.inline_direct_ref,
10773                 "content")
10774             + Cg(Cc(false), "is_direct")
10775             * Cg(parsers.rbracket, "content")))
10776
10777     parsers.link_image_open_or_close = parsers.link_image_opening

```

```

10778                                     + parsers.link_image_closing
10779
10780 if options.html then
10781     parsers.link_emph_precedence = parsers.inticks
10782                                     + parsers.autolink
10783                                     + parsers.html_inline_tags
10784 else
10785     parsers.link_emph_precedence = parsers.inticks
10786                                     + parsers.autolink
10787 end
10788
10789 parsers.link_and_emph_endline = parsers.newline
10790                                 * ((parsers.check_minimal_indent
10791                                     * -V("EndlineExceptions")
10792                                     + parsers.check_optional_indent
10793                                     * -V("EndlineExceptions")
10794                                     * -V("ListStarter")) / "")
10795                                 * parsers.spacechar^0 / "\n"
10796
10797 parsers.link_and_emph_content
10798   = Ct( Cg(Cc("content"), "type")
10799         * Cg(Cs(( parsers.link_emph_precedence
10800                   + parsers.backslash * parsers.linechar
10801                   + parsers.link_and_emph_endline
10802                   + (parsers.linechar
10803                     - parsers.blankline^2
10804                     - parsers.link_image_open_or_close
10805                     - parsers.emph_open_or_close))^0), "content"))
10806
10807 parsers.link_and_emph_table
10808   = (parsers.link_image_opening + parsers.emph_open)
10809     * parsers.link_and_emph_content
10810     * ((parsers.link_image_open_or_close + parsers.emph_open_or_close)
10811         * parsers.link_and_emph_content)^1
10812

```

Collect the content between the [opening\\_index](#) and [closing\\_index](#) in the delimiter table [t](#).

```

10813 local function collect_link_content(t, opening_index, closing_index)
10814     local content = {}
10815     for i = opening_index, closing_index do
10816         content[#content + 1] = t[i].content
10817     end
10818     return util.rope_to_string(content)
10819 end
10820

```

Look for the closest potential link opener in the delimiter table `t` in the range from `bottom_index` to `latest_index`.

```
10821 local function find_link_opener(t, bottom_index, latest_index)
10822   for i = latest_index, bottom_index, -1 do
10823     local value = t[i]
10824     if value.type == "delimiter" and
10825       value.is_opening and
10826       ( value.element == "link"
10827       or value.element == "image"
10828       or value.element == "note")
10829       and not value.removed then
10830       if value.is_active then
10831         return i
10832       end
10833       value.removed = true
10834       return nil
10835     end
10836   end
10837 end
10838
```

Find the position of a delimiter that closes a full link after an an index `latest_index` in the delimiter table `t`.

```
10839 local function find_next_link_closing_index(t, latest_index)
10840   for i = latest_index, #t do
10841     local value = t[i]
10842     if value.is_closing and
10843       value.element == "link" and
10844       not value.removed then
10845       return i
10846     end
10847   end
10848 end
10849
```

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.

```
10850 local function disable_previous_link_openers(t, opening_index)
10851   if t[opening_index].element == "image" then
10852     return
10853   end
10854
10855   for i = opening_index, 1, -1 do
10856     local value = t[i]
10857     if value.is_active and
10858       value.type == "delimiter" and
10859       value.is_opening and
```

```

10860         value.element == "link" then
10861         value.is_active = false
10862     end
10863 end
10864 end
10865

```

Disable the delimiters between the `opening_index` and `closing_index` in the delimiter table `t` by marking them inactive with the `is_active` property.

```

10866 local function disable_range(t, opening_index, closing_index)
10867     for i = opening_index, closing_index do
10868         local value = t[i]
10869         if value.is_active then
10870             value.is_active = false
10871             if value.type == "delimiter" then
10872                 value.removed = true
10873             end
10874         end
10875     end
10876 end
10877

```

Clear the parsed content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

10878 local delete_parsed_content_in_range =
10879     function(t, opening_index, closing_index)
10880         for i = opening_index, closing_index do
10881             t[i].rendered = nil
10882         end
10883     end
10884

```

Clear the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

10885 local function empty_content_in_range(t, opening_index, closing_index)
10886     for i = opening_index, closing_index do
10887         t[i].content = ''
10888     end
10889 end
10890

```

Join the attributes from the link reference definition `reference_attributes` with the link's own attributes `own_attributes`.

```

10891 local function join_attributes(reference_attributes, own_attributes)
10892     local merged_attributes = {}
10893     for _, attribute in ipairs(reference_attributes or {}) do
10894         table.insert(merged_attributes, attribute)
10895     end

```

```

10896     for _, attribute in ipairs(own_attributes or {}) do
10897         table.insert(merged_attributes, attribute)
10898     end
10899     if next(merged_attributes) == nil then
10900         merged_attributes = nil
10901     end
10902     return merged_attributes
10903 end
10904

```

Parse content between two delimiters in the delimiter table `t`. Produce the respective link and image macros.

```

10905     local render_link_or_image =
10906         function(t, opening_index, closing_index, content_end_index,
10907             reference)
10908             process_emphasis(t, opening_index, content_end_index)
10909             local mapped = collect_emphasis_content(t, opening_index + 1,
10910                 content_end_index - 1)
10911
10912             local rendered = {}
10913             if (t[opening_index].element == "link") then
10914                 rendered = writer.link(mapped, reference.url,
10915                     reference.title, reference.attributes)
10916             end
10917
10918             if (t[opening_index].element == "image") then
10919                 rendered = writer.image(mapped, reference.url,
10920                     self.parser_functions.parse_inlines_string(reference.title),
10921                     reference.attributes)
10922             end
10923
10924             if (t[opening_index].element == "note") then
10925                 if (t[opening_index].link_type == "note_inline") then
10926                     rendered = writer.note(mapped)
10927                 end
10928                 if (t[opening_index].link_type == "raw_note") then
10929                     rendered = writer.note(reference)
10930                 end
10931             end
10932
10933             t[opening_index].rendered = rendered
10934             delete_parsed_content_in_range(t, opening_index + 1,
10935                 closing_index)
10936             empty_content_in_range(t, opening_index, closing_index)
10937             disable_previous_link_openers(t, opening_index)
10938             disable_range(t, opening_index, closing_index)
10939         end

```

10940

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.

```
10941  local resolve_inline_following_content =
10942      function(t, closing_index, match_reference, match_link_attributes)
10943          local content = ""
10944          for i = closing_index + 1, #t do
10945              content = content .. t[i].content
10946          end
10947
10948          local matching_content = parsers.succeed
10949
10950          if match_reference then
10951              matching_content = matching_content
10952                  * parsers.inline_direct_ref_inside
10953          end
10954
10955          if match_link_attributes then
10956              matching_content = matching_content
10957                  * Cg(Ct(parsers.attributes^-1), "attributes")
10958          end
10959
10960          local matched = lpeg.match(Ct( matching_content
10961  * Cg(Cp(), "end_position")), content)
10962
10963          local matched_count = matched.end_position - 1
10964          for i = closing_index + 1, #t do
10965              local value = t[i]
10966
10967              local chars_left = matched_count
10968              matched_count = matched_count - #value.content
10969
10970              if matched_count <= 0 then
10971                  value.content = value.content:sub(chars_left + 1)
10972                  break
10973              end
10974
10975              value.content = ''
10976              value.is_active = false
10977          end
10978
10979          local attributes = matched.attributes
10980          if attributes == nil or next(attributes) == nil then
10981              attributes = nil
10982          end
```



```

10983
10984     return {
10985         url = matched.url or "",
10986         title = matched.title or "",
10987         attributes = attributes
10988     }
10989 end
10990

```

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.

```

10991 local function resolve_inline_link(t, opening_index, closing_index)
10992     local inline_content
10993     = resolve_inline_following_content(t, closing_index, true,
10994                                       t.match_link_attributes)
10995     render_link_or_image(t, opening_index, closing_index,
10996                         closing_index, inline_content)
10997 end
10998

```

Resolve an inline note `^[a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`.

```

10999 local resolve_note_inline_link =
11000     function(t, opening_index, closing_index)
11001         local inline_content
11002         = resolve_inline_following_content(t, closing_index,
11003   false, false)
11004         render_link_or_image(t, opening_index, closing_index,
11005                             closing_index, inline_content)
11006     end
11007

```

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.

```

11008 local function resolve_shortcut_link(t, opening_index, closing_index)
11009     local content
11010     = collect_link_content(t, opening_index + 1, closing_index - 1)
11011     local r = self.lookup_reference(content)
11012
11013     if r then
11014         local inline_content
11015         = resolve_inline_following_content(t, closing_index, false,
11016   t.match_link_attributes)
11017         r.attributes
11018         = join_attributes(r.attributes, inline_content.attributes)
11019         render_link_or_image(t, opening_index, closing_index,

```

```

11020                                     closing_index, r)
11021     end
11022 end
11023

```

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.

```

11024 local function resolve_raw_note_link(t, opening_index, closing_index)
11025     local content
11026     = collect_link_content(t, opening_index + 1, closing_index - 1)
11027     local r = self.lookup_note_reference(content)
11028
11029     if r then
11030         local parsed_ref = self.parser_functions.parse_blocks_nested(r)
11031         render_link_or_image(t, opening_index, closing_index,
11032                             closing_index, parsed_ref)
11033     end
11034 end
11035

```

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.

```

11036 local function resolve_full_link(t, opening_index, closing_index)
11037     local next_link_closing_index
11038     = find_next_link_closing_index(t, closing_index + 4)
11039     local next_link_content
11040     = collect_link_content(t, closing_index + 3,
11041                           next_link_closing_index - 1)
11042     local r = self.lookup_reference(next_link_content)
11043
11044     if r then
11045         local inline_content
11046         = resolve_inline_following_content(t, next_link_closing_index,
11047   false,
11048   t.match_link_attributes)
11049         r.attributes
11050         = join_attributes(r.attributes, inline_content.attributes)
11051         render_link_or_image(t, opening_index, next_link_closing_index,
11052                             closing_index, r)
11053     else
11054         local text = string.format('Undefined link reference "%s"',
11055                                   next_link_content)
11056         local more = string.format("Look for the text `[...] [%s]`.",
11057                                   next_link_content)
11058         t[opening_index].warning = writer.warning(text, more)
11059     end
11060 end
11061

```

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.

```

11062 local function resolve_collapsed_link(t, opening_index, closing_index)
11063     local next_link_closing_index
11064     = find_next_link_closing_index(t, closing_index + 4)
11065     local content
11066     = collect_link_content(t, opening_index + 1, closing_index - 1)
11067     local r = self.lookup_reference(content)
11068
11069     if r then
11070         local inline_content
11071         = resolve_inline_following_content(t, closing_index, false,
11072   t.match_link_attributes)
11073         r.attributes
11074         = join_attributes(r.attributes, inline_content.attributes)
11075         render_link_or_image(t, opening_index, next_link_closing_index,
11076                             closing_index, r)
11077     else
11078         local text = string.format('Undefined link reference "%s"',
11079                                   content)
11080         local more = string.format("Look for the text `[%s][ ]`.",
11081                                   content)
11082         t[opening_index].warning = writer.warning(text, more)
11083     end
11084 end
11085

```

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.

```

11086 local function process_links_and_emphasis(t)
11087     for _,value in ipairs(t) do
11088         value.is_active = true
11089     end
11090
11091     for i,value in ipairs(t) do
11092         if not value.is_closing
11093             or value.type ~= "delimiter"
11094             or not ( value.element == "link"
11095                     or value.element == "image"
11096                     or value.element == "note")
11097             or value.removed then
11098             goto continue
11099         end
11100

```

```

11101     local opener_position = find_link_opener(t, 1, i - 1)
11102     if (opener_position == nil) then
11103         goto continue
11104     end
11105
11106     local opening_delimiter = t[opener_position]
11107     opening_delimiter.removed = true
11108
11109     local link_type = opening_delimiter.link_type
11110
11111     if (link_type == "inline") then
11112         resolve_inline_link(t, opener_position, i)
11113     end
11114     if (link_type == "shortcut") then
11115         resolve_shortcut_link(t, opener_position, i)
11116     end
11117     if (link_type == "full") then
11118         resolve_full_link(t, opener_position, i)
11119     end
11120     if (link_type == "collapsed") then
11121         resolve_collapsed_link(t, opener_position, i)
11122     end
11123     if (link_type == "note_inline") then
11124         resolve_note_inline_link(t, opener_position, i)
11125     end
11126     if (link_type == "raw_note") then
11127         resolve_raw_note_link(t, opener_position, i)
11128     end
11129
11130     ::continue::
11131 end
11132
11133 t[#t].content = t[#t].content:gsub("%s*$","")
11134
11135 process_emphasis(t, 1, #t)
11136 local final_result = collect_emphasis_content(t, 1, #t)
11137 return final_result
11138 end
11139
11140 function self.defer_link_and_emphasis_processing(delimiter_table)
11141     return writer.defer_call(function()
11142         return process_links_and_emphasis(delimiter_table)
11143     end)
11144 end
11145

```

### 3.1.6.8 Inline Elements (local)

```
11146  parsers.Str      = ( parsers.normalchar
11147                      * (parsers.normalchar + parsers.at)^0)
11148                      / writer.string
11149
11150  parsers.Symbol    = (parsers.backtick^1 + V("SpecialChar"))
11151                      / writer.string
11152
11153  parsers.Ellipsis  = P("...") / writer.ellipsis
11154
11155  parsers.Smart     = parsers.Ellipsis
11156
11157  parsers.Code      = parsers.inticks / writer.code
11158
11159  if options.blankBeforeBlockquote then
11160    parsers.bqstart = parsers.fail
11161  else
11162    parsers.bqstart = parsers.blockquote_start
11163  end
11164
11165  if options.blankBeforeHeading then
11166    parsers.headerstart = parsers.fail
11167  else
11168    parsers.headerstart = parsers.atx_heading
11169  end
11170
11171  if options.blankBeforeList then
11172    parsers.interrupting_bullets = parsers.fail
11173    parsers.interrupting_enumerators = parsers.fail
11174  else
11175    parsers.interrupting_bullets
11176      = parsers.bullet(parsers.dash, true)
11177      + parsers.bullet(parsers.asterisk, true)
11178      + parsers.bullet(parsers.plus, true)
11179
11180    parsers.interrupting_enumerators
11181      = parsers.enumerator(parsers.period, true)
11182      + parsers.enumerator(parsers.rparent, true)
11183  end
11184
11185  if options.html then
11186    parsers.html_interrupting
11187      = parsers.check_trail
11188      * ( parsers.html_incomplete_open_tag
11189          + parsers.html_incomplete_close_tag
11189          + parsers.html_incomplete_open_special_tag
11190          + parsers.html_comment_start
```

```

11192         + parsers.html_cdatasection_start
11193         + parsers.html_declaration_start
11194         + parsers.html_instruction_start
11195         - parsers.html_close_special_tag
11196         - parsers.html_empty_special_tag)
11197     else
11198         parsers.html_interrupting = parsers.fail
11199     end
11200
11201     if options.blankBeforeHtmlBlock then
11202         parsers.html_interrupting = parsers.fail
11203     end
11204
11205     parsers.ListStarter = parsers.starter
11206
11207     parsers.EndlineExceptions
11208         = parsers.blankline -- paragraph break
11209         + parsers.eof       -- end of document
11210         + parsers.bqstart
11211         + parsers.thematic_break_lines
11212         + parsers.interrupting_bullets
11213         + parsers.interrupting_enumerators
11214         + parsers.headerstart
11215         + parsers.html_interrupting
11216
11217     parsers.NoSoftLineBreakEndlineExceptions = parsers.EndlineExceptions
11218
11219     parsers.endline = parsers.newline
11220         * (parsers.check_minimal_indent
11221         * -V("EndlineExceptions")
11222         + parsers.check_optional_indent
11223         * -V("EndlineExceptions")
11224         * -V("ListStarter")) / function(_) return end
11225         * parsers.spacechar~0
11226
11227     parsers.Endline = parsers.endline
11228         / writer.soft_line_break
11229
11230     parsers.EndlineNoSub = parsers.endline
11231
11232     parsers.NoSoftLineBreakEndline
11233         = parsers.newline
11234         * (parsers.check_minimal_indent
11235         * -V("NoSoftLineBreakEndlineExceptions")
11236         + parsers.check_optional_indent
11237         * -V("NoSoftLineBreakEndlineExceptions")
11238         * -V("ListStarter"))

```

```

11239             * parsers.spacechar^0
11240             / writer.space
11241
11242 parsers.EndlineBreak = parsers.backslash * parsers.endline
11243                       / writer.hard_line_break
11244
11245 parsers.OptionalIndent
11246     = parsers.spacechar^1 / writer.space
11247
11248 parsers.Space        = parsers.spacechar^2 * parsers.endline
11249                       / writer.hard_line_break
11250                       + parsers.spacechar^1
11251                       * parsers.endline~-1
11252                       * parsers.eof / self.expandtabs
11253                       + parsers.spacechar^1 * parsers.endline
11254                       / writer.soft_line_break
11255                       + parsers.spacechar^1
11256                       * -parsers.newline / self.expandtabs
11257                       + parsers.spacechar^1
11258                       + ( parsers.spacechar
11259                         + parsers.html_space_entity_char)^1
11260                       * parsers.endline / writer.soft_line_break
11261
11262 parsers.NonbreakingEndline
11263     = parsers.endline
11264     / writer.nbsp
11265
11266 parsers.NonbreakingSpace
11267     = parsers.spacechar^2 * parsers.endline
11268     / writer.nbsp
11269     + parsers.spacechar^1
11270     * parsers.endline~-1 * parsers.eof / ""
11271     + parsers.spacechar^1 * parsers.endline
11272     * parsers.optionalspace
11273     / writer.nbsp
11274     + parsers.spacechar^1 * parsers.optionalspace
11275     / writer.nbsp
11276

```

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.

```

11277 function self.auto_link_url(url, attributes)
11278     return writer.link(writer.escape(url),
11279                       url, nil, attributes)
11280 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.

```
11281 function self.auto_link_email(email, attributes)
11282   return writer.link(writer.escape(email),
11283                     "mailto: "..email,
11284                     nil, attributes)
11285 end
11286
11287 parsers.AutoLinkUrl = parsers.auto_link_url
11288                       / self.auto_link_url
11289
11290 parsers.AutoLinkEmail
11291                       = parsers.auto_link_email
11292                       / self.auto_link_email
11293
11294 parsers.AutoLinkRelativeReference
11295                       = parsers.auto_link_relative_reference
11296                       / self.auto_link_url
11297
11298 parsers.LinkAndEmph = Ct(parsers.link_and_emph_table)
11299                       / self.defer_link_and_emphasis_processing
11300
11301 parsers.EscapedChar = parsers.backslash
11302                       * C(parsers.escapable) / writer.string
11303
11304 parsers.InlineHtml = Cs(parsers.html_inline_comment)
11305                       / writer.inline_html_comment
11306                       + Cs(parsers.html_any_empty_inline_tag
11307                           + parsers.html_inline_instruction
11308                           + parsers.html_inline_cdatasection
11309                           + parsers.html_inline_declaration
11310                           + parsers.html_any_open_inline_tag
11311                           + parsers.html_any_close_tag)
11312                       / writer.inline_html_tag
11313
11314 parsers.HtmlEntity = parsers.html_entities / writer.string
```

### 3.1.6.9 Block Elements (local)

```
11315 parsers.DisplayHtml = Cs(parsers.check_trail
11316                           * ( parsers.html_comment
11317                               + parsers.html_special_block
11318                               + parsers.html_block
11319                               + parsers.html_any_block
11320                               + parsers.html_instruction
11321                               + parsers.html_cdatasection
11322                               + parsers.html_declaration))
```



```

11323                                     / writer.block_html_element
11324
11325 parsers.indented_non_blank_line = parsers.indentedline
11326                                 - parsers.blankline
11327
11328 parsers.Verbatim
11329   = Cs( parsers.check_code_trail
11330         * (parsers.line - parsers.blankline)
11331         * (( parsers.check_minimal_blank_indent_and_full_code_trail
11332             * parsers.blankline)^0
11333           * ( (parsers.check_minimal_indent / "")
11334             * parsers.check_code_trail
11335             * (parsers.line - parsers.blankline))^1)^0)
11336   / self.expandtabs / writer.verbatim
11337
11338 parsers.Blockquote   = parsers.blockquote_body
11339                       / writer.blockquote
11340
11341 parsers.ThematicBreak = parsers.thematic_break_lines
11342                       / writer.thematic_break
11343
11344 parsers.Reference    = parsers.define_reference_parser
11345                       / self.register_link
11346
11347 parsers.Paragraph    = parsers.freeze_trail
11348                       * (Ct((parsers.Inline)^1)
11349                       * (parsers.newline + parsers.eof)
11350                       * parsers.unfreeze_trail
11351                       / writer.paragraph)
11352
11353 parsers.Plain        = parsers.nonindentspace * Ct(parsers.Inline^1)
11354                       / writer.plain

```

### 3.1.6.10 Lists (local)

```

11355
11356 if options.taskLists then
11357   parsers.tickbox = ( parsers.ticked_box
11358                     + parsers.halfticked_box
11359                     + parsers.unticked_box
11360                     ) / writer.tickbox
11361 else
11362   parsers.tickbox = parsers.fail
11363 end
11364
11365 parsers.list_blank = parsers.conditionally_indented_blankline
11366

```

```

11367 parsers.ref_or_block_list_separated
11368     = parsers.sep_group_no_output(parsers.list_blank)
11369     * parsers.minimally_indented_ref
11370     + parsers.block_sep_group(parsers.list_blank)
11371     * parsers.minimally_indented_block
11372
11373 parsers.ref_or_block_non_separated
11374     = parsers.minimally_indented_ref
11375     + (parsers.succeed / writer.interblocksep)
11376     * parsers.minimally_indented_block
11377     - parsers.minimally_indented_blankline
11378
11379 parsers.tight_list_loop_body_pair =
11380     parsers.create_loop_body_pair(
11381         parsers.ref_or_block_non_separated,
11382         parsers.minimally_indented_par_or_plain_no_blank,
11383         (parsers.succeed / writer.interblocksep),
11384         (parsers.succeed / writer.paragraphsep))
11385
11386 parsers.loose_list_loop_body_pair =
11387     parsers.create_loop_body_pair(
11388         parsers.ref_or_block_list_separated,
11389         parsers.minimally_indented_par_or_plain,
11390         parsers.block_sep_group(parsers.list_blank),
11391         parsers.par_sep_group(parsers.list_blank))
11392
11393 parsers.tight_list_content_loop
11394     = V("Block")
11395     * parsers.tight_list_loop_body_pair.block^0
11396     + (V("Paragraph") + V("Plain"))
11397     * parsers.ref_or_block_non_separated
11398     * parsers.tight_list_loop_body_pair.block^0
11399     + (V("Paragraph") + V("Plain"))
11400     * parsers.tight_list_loop_body_pair.par^0
11401
11402 parsers.loose_list_content_loop
11403     = V("Block")
11404     * parsers.loose_list_loop_body_pair.block^0
11405     + (V("Paragraph") + V("Plain"))
11406     * parsers.ref_or_block_list_separated
11407     * parsers.loose_list_loop_body_pair.block^0
11408     + (V("Paragraph") + V("Plain"))
11409     * parsers.loose_list_loop_body_pair.par^0
11410
11411 parsers.list_item_tightness_condition
11412     = -( parsers.list_blank^0
11413         * parsers.minimally_indented_ref_or_block_or_par)

```

```

11414     * remove_indent("li")
11415     + remove_indent("li")
11416     * parsers.fail
11417
11418 parsers.indented_content_tight
11419   = Ct( (parsers.blankline / "")
11420         * #parsers.list_blank
11421         * remove_indent("li")
11422         + ( (V("Reference") + (parsers.blankline / ""))
11423             * parsers.check_minimal_indent
11424             * parsers.tight_list_content_loop
11425             + (V("Reference") + (parsers.blankline / ""))
11426             + (parsers.tickbox~-1 / writer.escape)
11427             * parsers.tight_list_content_loop
11428             )
11429         * parsers.list_item_tightness_condition)
11430
11431 parsers.indented_content_loose
11432   = Ct( (parsers.blankline / "")
11433         * #parsers.list_blank
11434         + ( (V("Reference") + (parsers.blankline / ""))
11435             * parsers.check_minimal_indent
11436             * parsers.loose_list_content_loop
11437             + (V("Reference") + (parsers.blankline / ""))
11438             + (parsers.tickbox~-1 / writer.escape)
11439             * parsers.loose_list_content_loop))
11440
11441 parsers.TightListItem = function(starter)
11442   return -parsers.ThematicBreak
11443         * parsers.add_indent(starter, "li")
11444         * parsers.indented_content_tight
11445 end
11446
11447 parsers.LooseListItem = function(starter)
11448   return -parsers.ThematicBreak
11449         * parsers.add_indent(starter, "li")
11450         * parsers.indented_content_loose
11451         * remove_indent("li")
11452 end
11453
11454 parsers.BulletListOfType = function(bullet_type)
11455   local bullet = parsers.bullet(bullet_type)
11456   return ( Ct( parsers.TightListItem(bullet)
11457               * ( (parsers.check_minimal_indent / "")
11458                   * parsers.TightListItem(bullet)
11459                   )~0
11460               )

```

```

11461         * Cc(true)
11462         * -#( (parsers.list_blank^0 / "")
11463             * parsers.check_minimal_indent
11464             * (bullet - parsers.ThematicBreak)
11465         )
11466         + Ct( parsers.LooseListItem(bullet)
11467             * ( (parsers.list_blank^0 / "")
11468                 * (parsers.check_minimal_indent / "")
11469                 * parsers.LooseListItem(bullet)
11470             )^0
11471         )
11472         * Cc(false)
11473     ) / writer.bulletlist
11474 end
11475
11476 parsers.BulletList = parsers.BulletListOfType(parsers.dash)
11477                     + parsers.BulletListOfType(parsers.asterisk)
11478                     + parsers.BulletListOfType(parsers.plus)
11479
11480 local function ordered_list(items,tight,starter)
11481     local startnum = starter[2][1]
11482     if options.startNumber then
11483         startnum = tonumber(startnum) or 1 -- fallback for '#'
11484         if startnum ~= nil then
11485             startnum = math.floor(startnum)
11486         end
11487     else
11488         startnum = nil
11489     end
11490     return writer.orderedlist(items,tight,startnum)
11491 end
11492
11493 parsers.OrderedListOfTypes = function(delimiter_type)
11494     local enumerator = parsers.enumerator(delimiter_type)
11495     return Cg(enumerator, "listtype")
11496         * (Ct( parsers.TightListItem(Cb("listtype"))
11497             * ( (parsers.check_minimal_indent / "")
11498                 * parsers.TightListItem(enumerator))^0)
11499         * Cc(true)
11500         * -#((parsers.list_blank^0 / "")
11501             * parsers.check_minimal_indent * enumerator)
11502     + Ct( parsers.LooseListItem(Cb("listtype"))
11503         * ((parsers.list_blank^0 / "")
11504             * (parsers.check_minimal_indent / "")
11505             * parsers.LooseListItem(enumerator))^0)
11506     * Cc(false)
11507     ) * Ct(Cb("listtype")) / ordered_list

```

```

11508 end
11509
11510 parsers.OrderedList = parsers.OrderedListOfType(parsers.period)
11511 + parsers.OrderedListOfType(parsers.rparent)

```

### 3.1.6.11 Blank (local)

```

11512 parsers.Blank = parsers.blankline / ""
11513 + V("Reference")

```

### 3.1.6.12 Headings (local)

```

11514 function parsers.parse_heading_text(s)
11515     local inlines = self.parser_functions.parse_inlines(s)
11516     local flatten_inlines = self.writer.flatten_inlines
11517     self.writer.flatten_inlines = true
11518     local flat_text = self.parser_functions.parse_inlines(s)
11519     flat_text = util.ropetostring(flat_text)
11520     self.writer.flatten_inlines = flatten_inlines
11521     return {flat_text, inlines}
11522 end
11523
11524 -- parse atx header
11525 parsers.AtxHeading = parsers.check_trail_no_rem
11526 * Cg(parsers.heading_start, "level")
11527 * ((C( parsers.optionalspace
11528     * parsers.hash^0
11529     * parsers.optionalspace
11530     * parsers.newline)
11531 + parsers.spacechar^1
11532 * C(parsers.line))
11533 / strip_atx_end
11534 / parsers.parse_heading_text)
11535 * Cb("level")
11536 / writer.heading
11537
11538 parsers.heading_line = parsers.linechar^1
11539 - parsers.thematic_break_lines
11540
11541 parsers.heading_text = parsers.heading_line
11542 * ( (V("Endline") / "\n")
11543 * ( parsers.heading_line
11544     - parsers.heading_level))^0
11545 * parsers.newline^-1
11546
11547 parsers.SettextHeading = parsers.freeze_trail
11548 * parsers.check_trail_no_rem
11549 * #( parsers.heading_text

```

```

11550             * parsers.check_minimal_indent
11551             * parsers.check_trail
11552             * parsers.heading_level)
11553         * Cs(parsers.heading_text)
11554         / parsers.parse_heading_text
11555         * parsers.check_minimal_indent_and_trail
11556         * parsers.heading_level
11557         * parsers.newline
11558         * parsers.unfreeze_trail
11559         / writer.heading
11560
11561     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 `TeX` output.

```

11562     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.

```

11563         local walkable_syntax = (function(global_walkable_syntax)
11564             local local_walkable_syntax = {}
11565             for lhs, rule in pairs(global_walkable_syntax) do
11566                 local_walkable_syntax[lhs] = util.table_copy(rule)
11567             end
11568             return local_walkable_syntax
11569         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.

```

11570         local current_extension_name = nil
11571         self.insert_pattern = function(selector, pattern, pattern_name)
11572             assert(pattern_name == nil or type(pattern_name) == "string")
11573             local _, _, lhs, pos, rhs
11574             = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
11575             assert(lhs ~= nil,
11576                 [[Expected selector in form ]]
11577                 .. [[ "LHS (before|after|instead of) RHS", not "]]
11578                 .. selector .. [[ "]] )
11579             assert(walkable_syntax[lhs] ~= nil,
11580                 [[Rule ]] .. lhs

```

```

11581     .. [[ -> ... does not exist in markdown grammar]])
11582     assert(pos == "before" or pos == "after" or pos == "instead of",
11583           [[Expected positional specifier "before", "after", ]]
11584           .. [[or "instead of", not "]]
11585           .. pos .. [["]])
11586     local rule = walkable_syntax[lhs]
11587     local index = nil
11588     for current_index, current_rhs in ipairs(rule) do
11589         if type(current_rhs) == "string" and current_rhs == rhs then
11590             index = current_index
11591             if pos == "after" then
11592                 index = index + 1
11593             end
11594             break
11595         end
11596     end
11597     assert(index ~= nil,
11598           [[Rule ]] .. lhs .. [[ -> ]] .. rhs
11599           .. [[ does not exist in markdown grammar]])
11600     local accountable_pattern
11601     if current_extension_name then
11602         accountable_pattern
11603         = {pattern, current_extension_name, pattern_name}
11604     else
11605         assert(type(pattern) == "string",
11606               [[reader->insert_pattern() was called outside ]]
11607               .. [[an extension with ]]
11608               .. [[a PEG pattern instead of a rule name]])
11609         accountable_pattern = pattern
11610     end
11611     if pos == "instead of" then
11612         rule[index] = accountable_pattern
11613     else
11614         table.insert(rule, index, accountable_pattern)
11615     end
11616     -- TODO: Remove all occurrences of `pattern` after `index`
11617     --         to improve speed?
11618 end
11619 if options.htmlOverLinks then
11620     self.insert_pattern("Inline before AutoLinkUrl", "InlineHtml")
11621 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.

```

11622     local syntax =
11623         { "Blocks",
11624

```

```

11625     Blocks = V("InitializeState")
11626           * V("ExpectedJekyllData")
11627           * 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.

```

11628         * ( V("Block")
11629             * ( V("Blank")^0 * parsers.eof
11630               + ( V("Blank")^2 / writer.paragraphsep
11631                 + V("Blank")^0 / writer.interblocksep
11632               )
11633             )
11634         + ( V("Paragraph") + V("Plain") )
11635         * ( V("Blank")^0 * parsers.eof
11636           + ( V("Blank")^2 / writer.paragraphsep
11637             + V("Blank")^0 / writer.interblocksep
11638           )
11639         )
11640         * V("Block")
11641         * ( V("Blank")^0 * parsers.eof
11642           + ( V("Blank")^2 / writer.paragraphsep
11643             + V("Blank")^0 / writer.interblocksep
11644           )
11645         )
11646         + ( V("Paragraph") + V("Plain") )
11647         * ( V("Blank")^0 * parsers.eof
11648           + V("Blank")^0 / writer.paragraphsep
11649         )
11650       )^0,
11651
11652     ExpectedJekyllData = parsers.succeed,
11653
11654     Blank                = parsers.Blank,
11655     Reference            = parsers.Reference,
11656
11657     Blockquote           = parsers.Blockquote,
11658     Verbatim             = parsers.Verbatim,
11659     ThematicBreak        = parsers.ThematicBreak,
11660     BulletList           = parsers.BulletList,
11661     OrderedList          = parsers.OrderedList,
11662     DisplayHtml          = parsers.DisplayHtml,
11663     Heading              = parsers.Heading,
11664     Paragraph            = parsers.Paragraph,
11665     Plain                = parsers.Plain,
11666
11667     ListStarter          = parsers.ListStarter,

```



```

11668      EndlineExceptions = parsers.EndlineExceptions,
11669      NoSoftLineBreakEndlineExceptions
11670                          = parsers.NoSoftLineBreakEndlineExceptions,
11671
11672      Str                  = parsers.Str,
11673      Space                = parsers.Space,
11674      OptionalIndent      = parsers.OptionalIndent,
11675      Endline              = parsers.Endline,
11676      EndlineNoSub        = parsers.EndlineNoSub,
11677      NoSoftLineBreakEndline
11678                          = parsers.NoSoftLineBreakEndline,
11679      EndlineBreak        = parsers.EndlineBreak,
11680      LinkAndEmph         = parsers.LinkAndEmph,
11681      Code                 = parsers.Code,
11682      AutoLinkUrl         = parsers.AutoLinkUrl,
11683      AutoLinkEmail       = parsers.AutoLinkEmail,
11684      AutoLinkRelativeReference
11685                          = parsers.AutoLinkRelativeReference,
11686      InlineHtml          = parsers.InlineHtml,
11687      HtmlEntity          = parsers.HtmlEntity,
11688      EscapedChar         = parsers.EscapedChar,
11689      Smart                = parsers.Smart,
11690      Symbol               = parsers.Symbol,
11691      SpecialChar         = parsers.fail,
11692      InitializeState     = parsers.succeed,
11693  }

```

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]`.

```

11694      self.update_rule = function(rule_name, get_pattern)
11695          assert(current_extension_name ~= nil)
11696          assert(syntax[rule_name] ~= nil,
11697              [[Rule ]] .. rule_name
11698              .. [[ -> ... does not exist in markdown grammar]])
11699          local previous_pattern
11700          local extension_name
11701          if walkable_syntax[rule_name] then
11702              local previous_accountable_pattern
11703              = walkable_syntax[rule_name][1]
11704              previous_pattern = previous_accountable_pattern[1]
11705              extension_name
11706              = previous_accountable_pattern[2]
11707              .. ", " .. current_extension_name
11708          else

```

```

11709         previous_pattern = nil
11710         extension_name = current_extension_name
11711     end
11712     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

```

```

11713         if type(get_pattern) == "function" then
11714             pattern = get_pattern(previous_pattern)
11715         else
11716             assert(previous_pattern == nil,
11717                 [[Rule ]] .. rule_name ..
11718                 [[ has already been updated by ]] .. extension_name)
11719             pattern = get_pattern
11720         end
11721         local accountable_pattern = { pattern, extension_name, rule_name }
11722         walkable_syntax[rule_name] = { accountable_pattern }
11723     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.

```

11724     local special_characters = {}
11725     self.add_special_character = function(c)
11726         table.insert(special_characters, c)
11727         syntax.SpecialChar = S(table.concat(special_characters, ""))
11728     end
11729
11730     self.add_special_character("*")
11731     self.add_special_character("[")
11732     self.add_special_character("]")
11733     self.add_special_character("<")
11734     self.add_special_character("!")
11735     self.add_special_character("\\")

```

Add method `reader->initialize_named_group` that defines named groups with a default capture value.

```

11736     self.initialize_named_group = function(name, value)
11737         local pattern = Ct("")

```

```

11738     if value ~= nil then
11739         pattern = pattern / value
11740     end
11741     syntax.InitializeState = syntax.InitializeState
11742                             * Cg(pattern, name)
11743 end

```

Add a named group for indentation.

```

11744     self.initialize_named_group("indent_info")

```

Apply syntax extensions.

```

11745     for _, extension in ipairs(extensions) do
11746         current_extension_name = extension.name
11747         extension.extend_writer(writer)
11748         extension.extend_reader(self)
11749     end
11750     current_extension_name = nil

```

If the `debugExtensions` option is enabled, serialize `walkable_syntax` to a JSON for debugging purposes.

```

11751     if options.debugExtensions then
11752         local sorted_lhs = {}
11753         for lhs, _ in pairs(walkable_syntax) do
11754             table.insert(sorted_lhs, lhs)
11755         end
11756         table.sort(sorted_lhs)
11757
11758         local output_lines = {"{"}
11759         for lhs_index, lhs in ipairs(sorted_lhs) do
11760             local encoded_lhs = util.encode_json_string(lhs)
11761             table.insert(output_lines, [[    ]] .. encoded_lhs .. [[[: []]])
11762             local rule = walkable_syntax[lhs]
11763             for rhs_index, rhs in ipairs(rule) do
11764                 local human_readable_rhs
11765                 if type(rhs) == "string" then
11766                     human_readable_rhs = rhs
11767                 else
11768                     local pattern_name
11769                     if rhs[3] then
11770                         pattern_name = rhs[3]
11771                     else
11772                         pattern_name = "Anonymous Pattern"
11773                     end
11774                     local extension_name = rhs[2]
11775                     human_readable_rhs = pattern_name .. [[ (]]
11776                                     .. extension_name .. [[)]]
11777                 end
11778                 local encoded_rhs

```

```

11779         = util.encode_json_string(human_readable_rhs)
11780         local output_line = [[          ]] .. encoded_rhs
11781         if rhs_index < #rule then
11782             output_line = output_line .. ","
11783         end
11784         table.insert(output_lines, output_line)
11785     end
11786     local output_line = "    ]"
11787     if lhs_index < #sorted_lhs then
11788         output_line = output_line .. ","
11789     end
11790     table.insert(output_lines, output_line)
11791 end
11792 table.insert(output_lines, "}")
11793
11794 local output = table.concat(output_lines, "\n")
11795 local output_filename = options.debugExtensionsFileName
11796 local output_file = assert(io.open(output_filename, "w"),
11797     [[Could not open file ]] .. output_filename
11798     .. [[ for writing]])
11799 assert(output_file:write(output))
11800 assert(output_file:close())
11801 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](#).

```

11802     for lhs, rule in pairs(walkable_syntax) do
11803         syntax[lhs] = parsers.fail
11804         for _, rhs in ipairs(rule) do
11805             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.

```

11806         if type(rhs) == "string" then
11807             pattern = V(rhs)
11808         else
11809             pattern = rhs[1]
11810             if type(pattern) == "string" then
11811                 pattern = V(pattern)
11812             end
11813         end
11814         syntax[lhs] = syntax[lhs] + pattern
11815     end
11816 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.

```
11817     if options.underscores then
11818         self.add_special_character("_")
11819     end
11820
11821     if not options.codeSpans then
11822         syntax.Code = parsers.fail
11823     else
11824         self.add_special_character("`")
11825     end
11826
11827     if not options.html then
11828         syntax.DisplayHtml = parsers.fail
11829         syntax.InlineHtml = parsers.fail
11830         syntax.HtmlEntity = parsers.fail
11831     else
11832         self.add_special_character("&")
11833     end
11834
11835     if options.preserveTabs then
11836         options.stripIndent = false
11837     end
11838
11839     if not options.smartEllipses then
11840         syntax.Smart = parsers.fail
11841     else
11842         self.add_special_character(".")
11843     end
11844
11845     if not options.relativeReferences then
11846         syntax.AutoLinkRelativeReference = parsers.fail
11847     end
11848
11849     if options.contentLevel == "inline" then
11850         syntax[1] = "Inlines"
11851         syntax.Inlines = V("InitializeState")
11852             * parsers.Inline^0
11853             * ( parsers.spacing^0
11854               * parsers.eof / "")
11855         syntax.Space = parsers.Space + parsers.blankline / writer.space
11856     end
11857
11858     local blocks_nested_t = util.table_copy(syntax)
11859     blocks_nested_t.ExpectedJekyllData = parsers.succeed
11860     parsers.blocks_nested = Ct(blocks_nested_t)
```

```

11861
11862     parsers.blocks = Ct(syntax)
11863
11864     local inlines_t = util.table_copy(syntax)
11865     inlines_t[1] = "Inlines"
11866     inlines_t.Inlines = V("InitializeState")
11867         * parsers.Inline^0
11868         * ( parsers.spacing^0
11869             * parsers.eof / "")
11870     parsers.inlines = Ct(inlines_t)
11871
11872     local inlines_no_inline_note_t = util.table_copy(inlines_t)
11873     inlines_no_inline_note_t.InlineNote = parsers.fail
11874     parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
11875
11876     local inlines_no_html_t = util.table_copy(inlines_t)
11877     inlines_no_html_t.DisplayHtml = parsers.fail
11878     inlines_no_html_t.InlineHtml = parsers.fail
11879     inlines_no_html_t.HtmlEntity = parsers.fail
11880     parsers.inlines_no_html = Ct(inlines_no_html_t)
11881
11882     local inlines_nbsp_t = util.table_copy(inlines_t)
11883     inlines_nbsp_t.Endline = parsers.NonbreakingEndline
11884     inlines_nbsp_t.Space = parsers.NonbreakingSpace
11885     parsers.inlines_nbsp = Ct(inlines_nbsp_t)
11886
11887     local inlines_identity_t = util.table_copy(inlines_t)
11888     inlines_identity_t.Str = parsers.any / function(s) return s end
11889     parsers.inlines_identity = Ct(inlines_identity_t)
11890
11891     local inlines_string_t = util.table_copy(inlines_t)
11892     inlines_string_t.Str = parsers.any / writer.string
11893     parsers.inlines_string = Ct(inlines_string_t)
11894
11895     local inlines_math_t = util.table_copy(inlines_t)
11896     inlines_math_t.Str = parsers.any / writer.math
11897     parsers.inlines_math = Ct(inlines_math_t)
11898
11899     local inlines_no_link_or_emphasis_t = util.table_copy(inlines_t)
11900     inlines_no_link_or_emphasis_t.LinkAndEmph = parsers.fail
11901     inlines_no_link_or_emphasis_t.EndlineExceptions
11902         = parsers.EndlineExceptions - parsers.eof
11903     parsers.inlines_no_link_or_emphasis
11904         = Ct(inlines_no_link_or_emphasis_t)

```

Return a function that converts markdown string `input` into a plain  $\text{\TeX}$  output and returns it..

```
11905     return function(input)
```

Normalize the input.

```
11906         if options.unicodeNormalization then
11907             local form = options.unicodeNormalizationForm
11908             input = util.normalize(input, form)
11909         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.

```
11910         input = input:gsub("\r\n?", "\n")
11911         if input:sub(-1) ~= "\n" then
11912             input = input .. "\n"
11913         end
```

Clear the table of references.

```
11914         references = {}
11915         local document = self.parser_functions.parse_blocks(input)
11916         local output = util.rope_to_string(writer.document(document))
```

Remove block element / paragraph separators immediately followed by the output of `writer->undosep`, possibly interleaved by section ends. Then, remove any leftover output of `writer->undosep`.

```
11917         local undosep_start, undosep_end
11918         local potential_secend_start, secend_start
11919         local potential_sep_start, sep_start
11920         while true do
11921             -- find a `writer->undosep`
11922             undosep_start, undosep_end
11923             = output:find(writer.undosep_text, 1, true)
11924             if undosep_start == nil then break end
11925             -- skip any preceding section ends
11926             secend_start = undosep_start
11927             while true do
11928                 potential_secend_start = secend_start - #writer.secend_text
11929                 if potential_secend_start < 1
11930                     or output:sub(potential_secend_start,
11931                                   secend_start - 1) ~= writer.secend_text
11932                 then
11933                     break
11934                 end
11935                 secend_start = potential_secend_start
11936             end
11937             -- find an immediately preceding
11938             -- block element / paragraph separator
11939             sep_start = secend_start
11940             potential_sep_start = sep_start - #writer.interblocksep_text
11941             if potential_sep_start >= 1
```

```

11942         and output:sub(potential_sep_start,
11943                         sep_start - 1) == writer.interblocksep_text
11944         then
11945             sep_start = potential_sep_start
11946         else
11947             potential_sep_start = sep_start - #writer.paragraphsep_text
11948             if potential_sep_start >= 1
11949                 and output:sub(potential_sep_start,
11950                             sep_start - 1) == writer.paragraphsep_text
11951             then
11952                 sep_start = potential_sep_start
11953             end
11954         end
11955         -- remove `writer->undosep` and immediately preceding
11956         -- block element / paragraph separator
11957         output = output:sub(1, sep_start - 1)
11958             .. output:sub(secend_start, undosep_start - 1)
11959             .. output:sub(undosep_end + 1)
11960         end
11961         return output
11962     end
11963 end
11964 return self
11965 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.

```

11966 M.extensions = {}

```

#### 3.1.7.1 Acronyms

The `extensions.acronyms` function recognizes acronyms, initialisms, and other all-caps sequences in markdown text.

```

11967 M.extensions.acronyms = function(acronyms)
11968     assert(#acronyms > 0)
11969     return {
11970         name = "built-in acronyms syntax extension",
11971         extend_writer = function(self)

```

Define `writer->acronym` as a function that will transform an input acronym, initialism, or another all-caps sequence to the output format.

```

11972         function self.acronym(s)

```



```

11973         if self.flatten_inlines then return s end
11974         return {"\\markdownRendererAcronym{" ,s,"}"}
11975     end
11976 end, extend_reader = function(self)
11977     local parsers = self.parsers
11978     local options = self.options
11979     local writer = self.writer

```

In order to minimize the size and speed of the parser, we will first construct prefix trees of all registered acronyms, initialisms, and other all-caps sequences.

```

11980     local prefix_tree = {_type = "intermediate"}
11981     for _, acronym in ipairs(acronyms) do
11982         local node = prefix_tree

```

Normalize each acronym, initialism, or another all-caps sequence.

```

11983         if options.unicodeNormalization then
11984             local form = options.unicodeNormalizationForm
11985             acronym = util.normalize(acronym, form)
11986         end
11987         for i = 1, #acronym do
11988             local acronym_byte = acronym:sub(i, i)
11989             if i < #acronym then
11990                 if node[acronym_byte] == nil then
11991                     node[acronym_byte] = {_type = "intermediate"}
11992                 end
11993                 node = node[acronym_byte]
11994             else
11995                 table.insert(node, {acronym_byte, _type = "leaf"})
11996             end
11997         end
11998     end

```

Next, we will construct a parser out of the prefix tree.

```

11999     local subparsers = {}
12000     depth_first_search(prefix_tree, "", function(node, path)
12001         if node._type == "leaf" then
12002             local acronym_byte = table.unpack(node)
12003             if subparsers[path] == nil then
12004                 subparsers[path] = parsers.fail
12005             end
12006             subparsers[path] = subparsers[path] + P(acronym_byte)
12007         end
12008     end, function(_, path)
12009         if #path > 0 then
12010             local byte = path:sub(#path, #path)
12011             local parent_path = path:sub(1, #path-1)
12012             local prefix = P(byte)
12013             local suffix = prefix * subparsers[path]

```

```

12014         if subparsers[parent_path] == nil then
12015             subparsers[parent_path] = parsers.fail
12016         end
12017         subparsers[parent_path] = subparsers[parent_path] + suffix
12018     end
12019 end)
12020 assert(subparsers[""] ~= nil)

```

Only match acronyms at word boundaries.

```

12021     local Acronym = ( parsers.unicode.punctuation
12022                       - V("SpecialChar")
12023                       + parsers.unicode.whitespace
12024                       - V("EndlineExceptions"))^0
12025     / writer.string
12026     * (
12027         subparsers[""]
12028         * #(-parsers.normalchar
12029           + parsers.unicode.punctuation
12030           + parsers.unicode.whitespace
12031           + parsers.eof)
12032         / writer.acronym)
12033     self.insert_pattern("Inline before Str", Acronym, "Acronym")
12034 end
12035 }
12036 end
12037 %
12038 %#### Bracketed Spans
12039 %
12040 % The \luamdef{extensions.bracketed_spans} function implements the Pandoc
12041 % bracketed span syntax extension.
12042 %
12043 % \end{markdown}
12044 % \begin{macrocode}
12045 M.extensions.bracketed_spans = function()
12046     return {
12047         name = "built-in bracketed_spans syntax extension",
12048         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.

```

12049     function self.span(s, attr)
12050         if self.flatten_inlines then return s end
12051         return {"\\markdownRendererBracketedSpanAttributeContextBegin",
12052               self.attributes(attr),
12053               "\\markdownRendererBracketedSpan{"..s.."}",
12054               "\\markdownRendererBracketedSpanAttributeContextEnd{}}"}
12055     end
12056 end, extend_reader = function(self)

```

```

12057     local parsers = self.parsers
12058     local writer = self.writer
12059
12060     local span_label = parsers.lbracket
12061                       * (Cs((parsers.alphanumeric^1
12062                             + parsers.inticks
12063                             + parsers.autolink
12064                             + V("InlineHtml")
12065                             + ( parsers.backslash * parsers.backslash)
12066                             + ( parsers.backslash
12067                               * (parsers.lbracket + parsers.rbracket)
12068                               + V("Space") + V("Endline")
12069                               + (parsers.any
12070                                 - ( parsers.newline
12071                                   + parsers.lbracket
12072                                   + parsers.rbracket
12073                                   + parsers.blankline^2))))^1)
12074                       / self.parser_functions.parse_inlines)
12075                       * parsers.rbracket
12076
12077     local Span = span_label
12078                 * Ct(parsers.attributes)
12079                 / writer.span
12080
12081     self.insert_pattern("Inline before LinkAndEmph",
12082                        Span, "Span")
12083 end
12084 }
12085 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.

```

12086 M.extensions.citations = function(citation_nbsps)
12087   return {
12088     name = "built-in citations syntax extension",
12089     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.

```

12090     function self.citations(text_cites, cites)
12091         local buffer = {}
12092         if self.flatten_inlines then
12093             for _,cite in ipairs(cites) do
12094                 if cite.prenote then
12095                     table.insert(buffer, {cite.prenote, " "})
12096                 end
12097                 table.insert(buffer, cite.name)
12098                 if cite.postnote then
12099                     table.insert(buffer, {" ", cite.postnote})
12100                 end
12101             end
12102         else
12103             table.insert(buffer,
12104                 {"\\markdownRenderer",
12105                     text_cites and "TextCite" or "Cite",
12106                     "{", #cites, "}"})
12107             for _,cite in ipairs(cites) do
12108                 table.insert(buffer,
12109                     {cite.suppress_author and "-" or "+", "{",
12110                     cite.prenote or "", "}{" ,
12111                     cite.postnote or "", "}{" , cite.name, "}"})
12112             end
12113         end
12114         return buffer
12115     end
12116 end, extend_reader = function(self)
12117     local parsers = self.parsers
12118     local writer = self.writer
12119
12120     local citation_chars
12121         = parsers.alphanumeric
12122         + S("#$%&-+<>~/_")
12123
12124     local citation_name
12125         = Cs(parsers.dash^-1) * parsers.at
12126         * Cs(citation_chars
12127             * ((( citation_chars
12128                 + parsers.internal_punctuation
12129                 - parsers.comma - parsers.semicolon)

```

```

12130         * -#(( parsers.internal_punctuation
12131             - parsers.comma
12132             - parsers.semicolon)^0
12133         * -( citation_chars
12134             + parsers.internal_punctuation
12135             - parsers.comma
12136             - parsers.semicolon)))^0
12137     * citation_chars)^-1)
12138
12139     local citation_body_prenote
12140         = Cs((parsers.alphanumeric^1
12141             + parsers.bracketed
12142             + parsers.inticks
12143             + parsers.autolink
12144             + V("InlineHtml")
12145             + V("Space") + V("EndlineNoSub")
12146             + (parsers.anyescaped
12147                 - ( parsers.newline
12148                     + parsers.rbracket
12149                     + parsers.blankline^2))
12150             - ( parsers.spnl
12151                 * parsers.dash^-1
12152                 * parsers.at))^1)
12153
12154     local citation_body_postnote
12155         = Cs((parsers.alphanumeric^1
12156             + parsers.bracketed
12157             + parsers.inticks
12158             + parsers.autolink
12159             + V("InlineHtml")
12160             + V("Space") + V("EndlineNoSub")
12161             + (parsers.anyescaped
12162                 - ( parsers.newline
12163                     + parsers.rbracket
12164                     + parsers.semicolon
12165                     + parsers.blankline^2))
12166             - (parsers.spnl * parsers.rbracket))^1)
12167
12168     local citation_body_chunk
12169         = ( citation_body_prenote
12170             * parsers.spnlc_sep
12171             + Cc("")
12172             * parsers.spnlc
12173         )
12174     * citation_name
12175     * ( parsers.internal_punctuation
12176         - parsers.semicolon)^-1

```

```

12177         * ( parsers.spnlc / function(_) return end
12178         * citation_body_postnote
12179         + Cc("")
12180         * parsers.spnlc
12181     )
12182
12183     local citation_body
12184         = citation_body_chunk
12185         * ( parsers.semicolon
12186         * parsers.spnlc
12187         * citation_body_chunk
12188         )^0
12189
12190     local citation_headless_body_postnote
12191         = Cs((parsers.alphanumeric^1
12192             + parsers.bracketed
12193             + parsers.inticks
12194             + parsers.autolink
12195             + V("InlineHtml")
12196             + V("Space") + V("Endline")
12197             + (parsers.anyescaped
12198                 - ( parsers.newline
12199                     + parsers.rbracket
12200                     + parsers.at
12201                     + parsers.semicolon + parsers.blankline^2))
12202             - (parsers.spnl * parsers.rbracket))^0)
12203
12204     local citation_headless_body
12205         = citation_headless_body_postnote
12206         * ( parsers.semicolon
12207         * parsers.spnlc
12208         * citation_body_chunk
12209         )^0
12210
12211     local citations
12212         = function(text_cites, raw_cites)
12213         local function normalize(str)
12214             if str == "" then
12215                 str = nil
12216             else
12217                 str = (citation_nbsps and
12218                     self.parser_functions.parse_inlines_nbsp or
12219                     self.parser_functions.parse_inlines)(str)
12220             end
12221             return str
12222         end
12223

```

```

12224         local cites = {}
12225         for i = 1,#raw_cites,4 do
12226             cites[#cites+1] = {
12227                 prenote = normalize(raw_cites[i]),
12228                 suppress_author = raw_cites[i+1] == "-",
12229                 name = writer.identifier(raw_cites[i+2]),
12230                 postnote = normalize(raw_cites[i+3]),
12231             }
12232         end
12233         return writer.citations(text_cites, cites)
12234     end
12235
12236     local TextCitations
12237         = Ct((parsers.spnlc
12238             * Cc("")
12239             * citation_name
12240             * ((parsers.spnlc
12241                 * parsers.lbracket
12242                 * citation_headless_body
12243                 * parsers.rbracket) + Cc("")))~1)
12244         / function(raw_cites)
12245             return citations(true, raw_cites)
12246         end
12247
12248     local ParenthesizedCitations
12249         = Ct((parsers.spnlc
12250             * parsers.lbracket
12251             * citation_body
12252             * parsers.rbracket)^1)
12253         / function(raw_cites)
12254             return citations(false, raw_cites)
12255         end
12256
12257     local Citations = TextCitations + ParenthesizedCitations
12258
12259     self.insert_pattern("Inline before LinkAndEmph",
12260         Citations, "Citations")
12261
12262     self.add_special_character("@")
12263     self.add_special_character("-")
12264 end
12265 }
12266 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.

```
12267 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.

```
12268   local languages_json = (function()
12269     local base, prev, curr
12270     for _, pathname in ipairs(util.find_files(language_map)) do
12271       local file = io.open(pathname, "r")
12272       if not file then goto continue end
12273       local input = assert(file:read("*a"))
12274       assert(file:close())
12275       local json = input:gsub('("[^\n]-"):', '[%1]=')
12276       curr = load("_ENV = {}; return "..json")()
12277       if type(curr) == "table" then
12278         if base == nil then
12279           base = curr
12280         else
12281           setmetatable(prev, { __index = curr })
12282           end
12283           prev = curr
12284         end
12285         ::continue::
12286       end
12287       return base or {}
12288     end)()
12289
12290   return {
12291     name = "built-in content_blocks syntax extension",
12292     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.

```
12293     function self.contentblock(src,suf,type,tit)
12294       if not self.is_writing then return "" end
12295       src = src..".."..suf
12296       suf = suf:lower()
12297       if type == "onlineimage" then
12298         return {"\\markdownRendererContentBlockOnlineImage{",suf,"}",
12299           "{",self.string(src),"}",
12300           "{",self.uri(src),"}",
12301           "{",self.string(tit or ""),"}"}
```



```

12302     elseif languages_json[suf] then
12303         return {"\\markdownRendererContentBlockCode{" , suf, "}" ,
12304             "{" , self.string(languages_json[suf]) , "}" ,
12305             "{" , self.string(src) , "}" ,
12306             "{" , self.uri(src) , "}" ,
12307             "{" , self.string(tit or "") , "}" }
12308     else
12309         return {"\\markdownRendererContentBlock{" , suf, "}" ,
12310             "{" , self.string(src) , "}" ,
12311             "{" , self.uri(src) , "}" ,
12312             "{" , self.string(tit or "") , "}" }
12313     end
12314 end
12315 end, extend_reader = function(self)
12316     local parsers = self.parsers
12317     local writer = self.writer
12318
12319     local contentblock_tail
12320         = parsers.optionalttitle
12321         * (parsers.newline + parsers.eof)
12322
12323     -- case insensitive online image suffix:
12324     local onlineimagesuffix
12325         = (function(...)
12326             local parser = nil
12327             for _, suffix in ipairs({...}) do
12328                 local pattern=nil
12329                 for i=1,#suffix do
12330                     local char=suffix:sub(i,i)
12331                     char = S(char:lower()..char:upper())
12332                     if pattern == nil then
12333                         pattern = char
12334                     else
12335                         pattern = pattern * char
12336                     end
12337                 end
12338                 if parser == nil then
12339                     parser = pattern
12340                 else
12341                     parser = parser + pattern
12342                 end
12343             end
12344             return parser
12345         end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
12346
12347     -- online image url for iA Writer content blocks with
12348     -- mandatory suffix, allowing nested brackets:

```

```

12349     local onlineimageurl
12350         = (parsers.less
12351             * Cs((parsers.anyescaped
12352                 - parsers.more
12353                 - parsers.spacing
12354                 - #(parsers.period
12355                     * onlineimagesuffix
12356                     * parsers.more
12357                     * contentblock_tail)))^0)
12358             * parsers.period
12359             * Cs(onlineimagesuffix)
12360             * parsers.more
12361             + (Cs((parsers.inparens
12362                 + (parsers.anyescaped
12363                     - parsers.spacing
12364                     - parsers.rparent
12365                     - #(parsers.period
12366                         * onlineimagesuffix
12367                         * contentblock_tail))))^0)
12368                 * parsers.period
12369                 * Cs(onlineimagesuffix))
12370             ) * Cc("onlineimage")
12371
12372     -- filename for iA Writer content blocks with mandatory suffix:
12373     local localfilepath
12374         = parsers.slash
12375         * Cs((parsers.anyescaped
12376             - parsers.tab
12377             - parsers.newline
12378             - #(parsers.period
12379                 * parsers.alphanumeric^1
12380                 * contentblock_tail))^1)
12381         * parsers.period
12382         * Cs(parsers.alphanumeric^1)
12383         * Cc("localfile")
12384
12385     local ContentBlock
12386         = parsers.check_trail_no_rem
12387         * (localfilepath + onlineimageurl)
12388         * contentblock_tail
12389         / writer.contentblock
12390
12391     self.insert_pattern("Block before Blockquote",
12392                         ContentBlock, "ContentBlock")
12393 end
12394 }
12395 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.

```
12396 M.extensions.definition_lists = function(tight_lists)
12397   return {
12398     name = "built-in definition_lists syntax extension",
12399     extend_writer = function(self)
12400       local function dlistem(term, defs)
12401         local retVal = {"\\markdownRendererDlItem{",term,""}
12402         for _, def in ipairs(defs) do
12403           retVal[#retVal+1]
12404             = {"\\markdownRendererDlDefinitionBegin ",def,
12405               "\\markdownRendererDlDefinitionEnd "}
12406         end
12407         retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
12408         return retVal
12409       end
12410
12411       function self.definitionlist(items,tight)
12412         if not self.is_writing then return "" end
12413         local buffer = {}
12414         for _,item in ipairs(items) do
12415           buffer[#buffer + 1] = dlistem(item.term, item.definitions)
12416         end
12417         if tight and tight_lists then
12418           return {"\\markdownRendererDlBeginTight\\n", buffer,
12419                 "\\n\\markdownRendererDlEndTight"}
12420         else
12421           return {"\\markdownRendererDlBegin\\n", buffer,
12422                 "\\n\\markdownRendererDlEnd"}
12423         end
12424       end
12425     end, extend_reader = function(self)
12426       local parsers = self.parsers
12427       local writer = self.writer
12428
12429       local defstartchar = S("~:")
12430
12431       local defstart
12432       = parsers.check_trail_length(0) * defstartchar
12433       * #parsers.spacing
```

```

12434      * (parsers.tab + parsers.space^-3)
12435      + parsers.check_trail_length(1)
12436      * defstartchar * #parsers.spacing
12437      * (parsers.tab + parsers.space^-2)
12438      + parsers.check_trail_length(2)
12439      * defstartchar * #parsers.spacing
12440      * (parsers.tab + parsers.space^-1)
12441      + parsers.check_trail_length(3)
12442      * defstartchar * #parsers.spacing
12443
12444      local indented_line
12445      = (parsers.check_minimal_indent / "")
12446      * parsers.check_code_trail * parsers.line
12447
12448      local blank
12449      = parsers.check_minimal_blank_indent_and_any_trail
12450      * parsers.optionalspace * parsers.newline
12451
12452      local dlchunk = Cs(parsers.line * (indented_line - blank)^0)
12453
12454      local indented_blocks = function(bl)
12455      return Cs( bl
12456      * (blank^1 * (parsers.check_minimal_indent / "")
12457      * parsers.check_code_trail * -parsers.blankline * bl)^0
12458      * (blank^1 + parsers.eof))
12459      end
12460
12461      local function definition_list_item(term, defs, _)
12462      return { term = self.parser_functions.parse_inlines(term),
12463      definitions = defs }
12464      end
12465
12466      local DefinitionListItemLoose
12467      = C(parsers.line) * blank^0
12468      * Ct((parsers.check_minimal_indent * (defstart
12469      * indented_blocks(dlchunk)
12470      / self.parser_functions.parse_blocks_nested))^1)
12471      * Cc(false) / definition_list_item
12472
12473      local DefinitionListItemTight
12474      = C(parsers.line)
12475      * Ct((parsers.check_minimal_indent * (defstart * dlchunk
12476      / self.parser_functions.parse_blocks_nested))^1)
12477      * Cc(true) / definition_list_item
12478
12479      local DefinitionList
12480      = ( Ct(DefinitionListItemLoose^1) * Cc(false)

```

```

12481         + Ct(DefinitionListItemTight~1)
12482         * (blank~0
12483         * -DefinitionListItemLoose * Cc(true))
12484         ) / writer.definitionlist
12485
12486         self.insert_pattern("Block after Heading",
12487                             DefinitionList, "DefinitionList")
12488     end
12489 }
12490 end

```

### 3.1.7.5 Fancy Lists

The `extensions.fancy_lists` function implements the Pandoc fancy list syntax extension.

```

12491 M.extensions.fancy_lists = function()
12492   return {
12493     name = "built-in fancy_lists syntax extension",
12494     extend_writer = function(self)
12495       local options = self.options
12496

```

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.

```

12497     function self.fancylist(items,tight,startnum,numstyle,numdelim)
12498       if not self.is_writing then return "" end
12499       local buffer = {}
12500       local num = startnum

```

```

12501     for _,item in ipairs(items) do
12502         if item ~= "" then
12503             buffer[#buffer + 1] = self.fancyitem(item,num)
12504         end
12505         if num ~= nil and item ~= "" then
12506             num = num + 1
12507         end
12508     end
12509     local contents = util.intersperse(buffer,"\n")
12510     if tight and options.tightLists then
12511         return {"\\markdownRendererFancyOlBeginTight{",
12512             numstyle,"}{",numdelim,"}",contents,
12513             "\\n\\markdownRendererFancyOlEndTight "}
12514     else
12515         return {"\\markdownRendererFancyOlBegin{",
12516             numstyle,"}{",numdelim,"}",contents,
12517             "\\n\\markdownRendererFancyOlEnd "}
12518     end
12519 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.

```

12520     function self.fancyitem(s,num)
12521         if num ~= nil then
12522             return {"\\markdownRendererFancyOlItemWithNumber{",num,"}",s,
12523                 "\\markdownRendererFancyOlItemEnd "}
12524         else
12525             return {"\\markdownRendererFancyOlItem ",s,
12526                 "\\markdownRendererFancyOlItemEnd "}
12527         end
12528     end
12529 end, extend_reader = function(self)
12530     local parsers = self.parsers
12531     local options = self.options
12532     local writer = self.writer
12533
12534     local function combine_markers_and_delims(markers, delims)
12535         local markers_table = {}
12536         for _,marker in ipairs(markers) do
12537             local start_marker
12538             local continuation_marker
12539             if type(marker) == "table" then
12540                 start_marker = marker[1]
12541                 continuation_marker = marker[2]
12542             else
12543                 start_marker = marker

```

```

12544         continuation_marker = marker
12545     end
12546     for _,delim in ipairs(delims) do
12547         table.insert(markers_table,
12548             {start_marker, continuation_marker, delim})
12549     end
12550 end
12551 return markers_table
12552 end
12553
12554 local function join_table_with_func(func, markers_table)
12555     local pattern = func(table.unpack(markers_table[1]))
12556     for i = 2, #markers_table do
12557         pattern = pattern + func(table.unpack(markers_table[i]))
12558     end
12559     return pattern
12560 end
12561
12562 local lowercase_letter_marker = R("az")
12563 local uppercase_letter_marker = R("AZ")
12564
12565 local roman_marker = function(chars)
12566     local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])
12567     local l, x, v, i
12568         = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])
12569     return  m^-3
12570           * (c*m + c*d + d^-1 * c^-3)
12571           * (x*c + x*l + l^-1 * x^-3)
12572           * (i*x + i*v + v^-1 * i^-3)
12573 end
12574
12575 local lowercase_roman_marker
12576     = roman_marker({"m", "d", "c", "l", "x", "v", "i"})
12577 local uppercase_roman_marker
12578     = roman_marker({"M", "D", "C", "L", "X", "V", "I"})
12579
12580 local lowercase_opening_roman_marker = P("i")
12581 local uppercase_opening_roman_marker = P("I")
12582
12583 local digit_marker = parsers.dig * parsers.dig^-8
12584
12585 local markers = {
12586     {lowercase_opening_roman_marker, lowercase_roman_marker},
12587     {uppercase_opening_roman_marker, uppercase_roman_marker},
12588     lowercase_letter_marker,
12589     uppercase_letter_marker,
12590     lowercase_roman_marker,

```

```

12591     uppercase_roman_marker,
12592     digit_marker
12593 }
12594
12595 local delims = {
12596     parsers.period,
12597     parsers.rparent
12598 }
12599
12600 local markers_table = combine_markers_and_delims(markers, delims)
12601
12602 local function enumerator(start_marker, _,
12603     delimiter_type, interrupting)
12604     local delimiter_range
12605     local allowed_end
12606     if interrupting then
12607         delimiter_range = P("1")
12608         allowed_end = C(parsers.spacechar^1) * #parsers.linechar
12609     else
12610         delimiter_range = start_marker
12611         allowed_end = C(parsers.spacechar^1)
12612             + #(parsers.newline + parsers.eof)
12613     end
12614
12615     return parsers.check_trail
12616         * Ct(C(delimiter_range) * C(delimiter_type))
12617         * allowed_end
12618 end
12619
12620 local starter = join_table_with_func(enumerator, markers_table)
12621
12622 local TightListItem = function(starter)
12623     return parsers.add_indent(starter, "li")
12624         * parsers.indented_content_tight
12625 end
12626
12627 local LooseListItem = function(starter)
12628     return parsers.add_indent(starter, "li")
12629         * parsers.indented_content_loose
12630         * remove_indent("li")
12631 end
12632
12633 local function roman2number(roman)
12634     local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100,
12635         ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
12636     local numeral = 0
12637

```



```

12638     local i = 1
12639     local len = string.len(roman)
12640     while i < len do
12641         local z1, z2 = romans[ string.sub(roman, i, i) ],
12642             romans[ string.sub(roman, i+1, i+1) ]
12643         if z1 < z2 then
12644             numeral = numeral + (z2 - z1)
12645             i = i + 2
12646         else
12647             numeral = numeral + z1
12648             i = i + 1
12649         end
12650     end
12651     if i <= len then
12652         numeral = numeral + romans[ string.sub(roman,i,i) ]
12653     end
12654     return numeral
12655 end
12656
12657 local function sniffstyle(numstr, delimend)
12658     local numdelim
12659     if delimend == ")" then
12660         numdelim = "OneParen"
12661     elseif delimend == "." then
12662         numdelim = "Period"
12663     else
12664         numdelim = "Default"
12665     end
12666
12667     local num
12668     num = numstr:match("^([I])$")
12669     if num then
12670         return roman2number(num), "UpperRoman", numdelim
12671     end
12672     num = numstr:match("^([i])$")
12673     if num then
12674         return roman2number(string.upper(num)), "LowerRoman", numdelim
12675     end
12676     num = numstr:match("^([A-Z])$")
12677     if num then
12678         return string.byte(num) - string.byte("A") + 1,
12679             "UpperAlpha", numdelim
12680     end
12681     num = numstr:match("^([a-z])$")
12682     if num then
12683         return string.byte(num) - string.byte("a") + 1,
12684             "LowerAlpha", numdelim

```

```

12685         end
12686         num = numstr:match("^([IVXLCDM]+)")
12687         if num then
12688             return roman2number(num), "UpperRoman", numdelim
12689         end
12690         num = numstr:match("^([ivxlcdm]+)")
12691         if num then
12692             return roman2number(string.upper(num)), "LowerRoman", numdelim
12693         end
12694         return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
12695     end
12696
12697     local function fancylist(items,tight,start)
12698         local startnum, numstyle, numdelim
12699         = sniffstyle(start[2][1], start[2][2])
12700         return writer.fancylist(items,tight,
12701                                 options.startNumber and startnum or 1,
12702                                 numstyle or "Decimal",
12703                                 numdelim or "Default")
12704     end
12705
12706     local FancyListOfType
12707     = function(start_marker, continuation_marker, delimiter_type)
12708         local enumerator_start
12709         = enumerator(start_marker, continuation_marker,
12710                     delimiter_type)
12711         local enumerator_cont
12712         = enumerator(continuation_marker, continuation_marker,
12713                     delimiter_type)
12714         return Cg(enumerator_start, "listtype")
12715             * (Ct( TightListItem(Cb("listtype"))
12716                 * ((parsers.check_minimal_indent / "")
12717                   * TightListItem(enumerator_cont))^0)
12718             * Cc(true)
12719             * -#((parsers.conditionally_indented_blankline^0 / "")
12720                 * parsers.check_minimal_indent * enumerator_cont)
12721             + Ct( LooseListItem(Cb("listtype"))
12722                 * ((parsers.conditionally_indented_blankline^0 / "")
12723                   * (parsers.check_minimal_indent / "")
12724                   * LooseListItem(enumerator_cont))^0)
12725             * Cc(false)
12726             ) * Ct(Cb("listtype")) / fancylist
12727     end
12728
12729     local FancyList
12730     = join_table_with_func(FancyListOfType, markers_table)
12731

```

```

12732     local ListStarter = starter
12733
12734     self.update_rule("OrderedList", FancyList)
12735     self.update_rule("ListStarter", ListStarter)
12736 end
12737 }
12738 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.

```

12739 M.extensions.fenced_code = function(blank_before_code_fence,
12740                                     allow_attributes,
12741                                     allow_raw_blocks)
12742   return {
12743     name = "built-in fenced_code syntax extension",
12744     extend_writer = function(self)
12745       local options = self.options
12746

```

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.

```

12747   function self.fencedCode(s, i, attr)
12748     if not self.is_writing then return "" end
12749     s = s:gsub("\n$", "")
12750     local buf = {}
12751     if attr ~= nil then
12752       table.insert(buf,
12753         {"\\markdownRendererFencedCodeAttributeContextBegin",
12754          self.attributes(attr)})
12755     end
12756     local name = util.cache_verbatim(options.cacheDir, s)
12757     table.insert(buf,
12758       {"\\markdownRendererInputFencedCode{",
12759        name,"}{"",self.string(i),"}{"",self.infostring(i),""}")
12760     if attr ~= nil then
12761       table.insert(buf,
12762         "\\markdownRendererFencedCodeAttributeContextEnd{")
12763     end
12764     return buf

```

```

12765         end
12766

```

Define `writer->rawBlock` as a function that will transform an input raw block `s` with the raw attribute `attr` to the output format.

```

12767         if allow_raw_blocks then
12768             function self.rawBlock(s, attr)
12769                 if not self.is_writing then return "" end
12770                 s = s:gsub("\n$", "")
12771                 local name = util.cache_verbatim(options.cacheDir, s)
12772                 return {"\\markdownRendererInputRawBlock{",
12773                     name,"}{", self.string(attr),""}
12774             end
12775         end
12776     end, extend_reader = function(self)
12777         local parsers = self.parsers
12778         local writer = self.writer
12779
12780         local function captures_geq_length(_,i,a,b)
12781             return #a >= #b and i
12782         end
12783
12784         local function strip_enclosing_whitespaces(str)
12785             return str:gsub("^%s*(.)%s*$", "%1")
12786         end
12787
12788         local tilde_infostring = Cs(Cs((V("HtmlEntity")
12789             + parsers.anyescaped
12790             - parsers.newline)^0)
12791             / strip_enclosing_whitespaces)
12792
12793         local backtick_infostring
12794             = Cs( Cs((V("HtmlEntity")
12795                 + ( -#(parsers.backslash * parsers.backtick)
12796                     * parsers.anyescaped)
12797                 - parsers.newline
12798                 - parsers.backtick)^0)
12799                 / strip_enclosing_whitespaces)
12800
12801         local fenceindent
12802
12803         local function has_trail(indent_table)
12804             return indent_table ~= nil and
12805                 indent_table.trail ~= nil and
12806                 next(indent_table.trail) ~= nil
12807         end
12808

```

```

12809     local function has_indents(indent_table)
12810         return indent_table ~= nil and
12811             indent_table.indents ~= nil and
12812             next(indent_table.indents) ~= nil
12813     end
12814
12815     local function get_last_indent_name(indent_table)
12816         if has_indents(indent_table) then
12817             return indent_table.indents[#indent_table.indents].name
12818         end
12819     end
12820
12821     local count_fenced_start_indent =
12822     function(_, _, indent_table, trail)
12823         local last_indent_name = get_last_indent_name(indent_table)
12824         fenceindent = 0
12825         if last_indent_name ~= "li" then
12826             fenceindent = #trail
12827         end
12828         return true
12829     end
12830
12831     local fencehead = function(char, infostring)
12832         return Cmt( Cb("indent_info")
12833             * parsers.check_trail, count_fenced_start_indent)
12834             * Cg(char^3, "fencelength")
12835             * parsers.optionalspace
12836             * infostring
12837             * (parsers.newline + parsers.eof)
12838     end
12839
12840     local fencetail = function(char)
12841         return parsers.check_trail_no_rem
12842             * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
12843             * parsers.optionalspace * (parsers.newline + parsers.eof)
12844             + parsers.eof
12845     end
12846
12847     local process_fenced_line =
12848     function(s, i, -- luacheck: ignore s i
12849         indent_table, line_content, is_blank)
12850         local remainder = ""
12851         if has_trail(indent_table) then
12852             remainder = indent_table.trail.internal_remainder
12853         end
12854
12855         if is_blank

```

```

12856         and get_last_indent_name(indent_table) == "li" then
12857             remainder = ""
12858         end
12859
12860         local str = remainder .. line_content
12861         local index = 1
12862         local remaining = fenceindent
12863
12864         while true do
12865             local c = str:sub(index, index)
12866             if c == " " and remaining > 0 then
12867                 remaining = remaining - 1
12868                 index = index + 1
12869             elseif c == "\t" and remaining > 3 then
12870                 remaining = remaining - 4
12871                 index = index + 1
12872             else
12873                 break
12874             end
12875         end
12876
12877         return true, str:sub(index)
12878     end
12879
12880     local fencedline = function(char)
12881         return Cmt( Cb("indent_info")
12882             * C(parsers.line - fencetail(char))
12883             * Cc(false), process_fenced_line)
12884     end
12885
12886     local blankfencedline
12887     = Cmt( Cb("indent_info")
12888         * C(parsers.blankline)
12889         * Cc(true), process_fenced_line)
12890
12891     local TildeFencedCode
12892     = fencehead(parsers.tilde, tilde_infostring)
12893     * Cs(( parsers.check_minimal_blank_indent / "")
12894         * blankfencedline
12895         + ( parsers.check_minimal_indent / "")
12896         * fencedline(parsers.tilde))^0)
12897     * ( (parsers.check_minimal_indent / "")
12898         * fencetail(parsers.tilde) + parsers.succeed)
12899
12900     local BacktickFencedCode
12901     = fencehead(parsers.backtick, backtick_infostring)
12902     * Cs(( parsers.check_minimal_blank_indent / "")

```

```

12903         * blankfencedline
12904         + (parsers.check_minimal_indent / "")
12905         * fencedline(parsers.backtick))^0)
12906     * ( (parsers.check_minimal_indent / "")
12907         * fencetail(parsers.backtick) + parsers.succeed)
12908
12909     local infostring_with_attributes
12910         = Ct(C((parsers.linechar
12911             - ( parsers.optionalspace
12912               * parsers.attributes))^0)
12913             * parsers.optionalspace
12914             * Ct(parsers.attributes))
12915
12916     local FencedCode
12917         = ((TildeFencedCode + BacktickFencedCode)
12918         / function(infostring, code)
12919             local expanded_code = self.expandtabs(code)
12920
12921             if allow_raw_blocks then
12922                 local raw_attr = lpeg.match(parsers.raw_attribute,
12923   infostring)
12924
12925                 if raw_attr then
12926                     return writer.rawBlock(expanded_code, raw_attr)
12927                 end
12928             end
12929
12930             local attr = nil
12931             if allow_attributes then
12932                 local match = lpeg.match(infostring_with_attributes,
12933   infostring)
12934
12935                 if match then
12936                     infostring, attr = table.unpack(match)
12937                 end
12938             end
12939             return writer.fencedCode(expanded_code, infostring, attr)
12940         end)
12941
12942     self.insert_pattern("Block after Verbatim",
12943                       FencedCode, "FencedCode")
12944
12945     local fencestart
12946     if blank_before_code_fence then
12947         fencestart = parsers.fail
12948     else
12949         fencestart = fencehead(parsers.backtick, backtick_infostring)
12950             + fencehead(parsers.tilde, tilde_infostring)
12951     end

```

```

12950
12951     self.update_rule("EndlineExceptions", function(previous_pattern)
12952         if previous_pattern == nil then
12953             previous_pattern = parsers.EndlineExceptions
12954         end
12955         return previous_pattern + fencestart
12956     end)
12957
12958     self.add_special_character("`")
12959     self.add_special_character("~")
12960 end
12961 }
12962 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.

```

12963 M.extensions.fenced_divs = function(blank_before_div_fence)
12964     return {
12965         name = "built-in fenced_divs syntax extension",
12966         extend_writer = function(self)

```

Define `writer->div_begin` as a function that will transform the beginning of an input fenced div with attributes `attributes` to the output format.

```

12967         function self.div_begin(attributes)
12968             local start_output
12969             = {"\\markdownRendererFencedDivAttributeContextBegin\\n",
12970                 self.attributes(attributes)}
12971             local end_output
12972             = {"\\markdownRendererFencedDivAttributeContextEnd{}}"}
12973             return self.push_attributes(
12974                 "div", attributes, start_output, end_output)
12975         end

```

Define `writer->div_end` as a function that will produce the end of a fenced div in the output format.

```

12976         function self.div_end()
12977             return self.pop_attributes("div")
12978         end
12979     end, extend_reader = function(self)
12980         local parsers = self.parsers
12981         local writer = self.writer

```

Define basic patterns for matching the opening and the closing tag of a div.

```

12982         local fenced_div_infostring

```



```

12983             = Ct(parsers.attributes)
12984             + (
12985                 C( parsers.attribute_classname
12986                   - parsers.colon)^1
12987                 / function (infostring)
12988                     return {"." .. infostring}
12989                 end
12990             )
12991
12992     local fenced_div_begin = parsers.nonindentspace
12993     * parsers.colon^3
12994     * parsers.optionalspace
12995     * fenced_div_infostring
12996     * ( parsers.spacechar^1
12997       * parsers.colon^1)^0
12998     * parsers.optionalspace
12999     * (parsers.newline + parsers.eof)
13000
13001     local fenced_div_end = parsers.nonindentspace
13002     * parsers.colon^3
13003     * parsers.optionalspace
13004     * (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.

```

13005     self.initialize_named_group("fenced_div_level", "0")
13006     self.initialize_named_group("fenced_div_num_opening_indents")
13007
13008     local function increment_div_level()
13009         local push_indent_table =
13010             function(s, i, indent_table, -- luacheck: ignore s i
13011                   fenced_div_num_opening_indents, fenced_div_level)
13012                 fenced_div_level = tonumber(fenced_div_level) + 1
13013                 local num_opening_indents = 0
13014                 if indent_table.indents ~= nil then
13015                     num_opening_indents = #indent_table.indents
13016                 end
13017                 fenced_div_num_opening_indents[fenced_div_level]
13018                     = num_opening_indents
13019                 return true, fenced_div_num_opening_indents
13020             end
13021
13022     local increment_level =

```

```

13023         function(s, i, fenced_div_level) -- luacheck: ignore s i
13024             fenced_div_level = tonumber(fenced_div_level) + 1
13025             return true, tostring(fenced_div_level)
13026         end
13027
13028         return Cg( Cmt( Cb("indent_info")
13029             * Cb("fenced_div_num_opening_indents")
13030             * Cb("fenced_div_level"), push_indent_table)
13031             , "fenced_div_num_opening_indents")
13032         * Cg( Cmt( Cb("fenced_div_level"), increment_level)
13033             , "fenced_div_level")
13034     end
13035
13036     local function decrement_div_level()
13037         local pop_indent_table =
13038             function(s, i, -- luacheck: ignore s i
13039                 fenced_div_indent_table, fenced_div_level)
13040                 fenced_div_level = tonumber(fenced_div_level)
13041                 fenced_div_indent_table[fenced_div_level] = nil
13042                 return true, tostring(fenced_div_level - 1)
13043             end
13044
13045             return Cg( Cmt( Cb("fenced_div_num_opening_indents")
13046                 * Cb("fenced_div_level"), pop_indent_table)
13047                 , "fenced_div_level")
13048     end
13049
13050
13051     local non_fenced_div_block
13052     = parsers.check_minimal_indent * V("Block")
13053     - parsers.check_minimal_indent_and_trail * fenced_div_end
13054
13055     local non_fenced_div_paragraph
13056     = parsers.check_minimal_indent * V("Paragraph")
13057     - parsers.check_minimal_indent_and_trail * fenced_div_end
13058
13059     local blank = parsers.minimally_indented_blank
13060
13061     local block_separated = parsers.block_sep_group(blank)
13062         * non_fenced_div_block
13063
13064     local loop_body_pair
13065     = parsers.create_loop_body_pair(block_separated,
13066                                     non_fenced_div_paragraph,
13067                                     parsers.block_sep_group(blank),
13068                                     parsers.par_sep_group(blank))
13069

```

```

13070     local content_loop = ( non_fenced_div_block
13071                           * loop_body_pair.block^0
13072                           + non_fenced_div_paragraph
13073                           * block_separated
13074                           * loop_body_pair.block^0
13075                           + non_fenced_div_paragraph
13076                           * loop_body_pair.par^0)
13077                           * blank^0
13078
13079     local FencedDiv = fenced_div_begin
13080                       / writer.div_begin
13081                       * increment_div_level()
13082                       * parsers.skipblanklines
13083                       * Ct(content_loop)
13084                       * parsers.minimally_indented_blank^0
13085                       * parsers.check_minimal_indent_and_trail
13086                       * fenced_div_end
13087                       * decrement_div_level()
13088                       * (Cc("") / writer.div_end)
13089
13090     self.insert_pattern("Block after Verbatim",
13091                       FencedDiv, "FencedDiv")
13092
13093     self.add_special_character(":")
13094

```

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.

```

13095     local function is_inside_div()
13096       local check_div_level =
13097         function(s, i, fenced_div_level) -- luacheck: ignore s i
13098           fenced_div_level = tonumber(fenced_div_level)
13099           return fenced_div_level > 0
13100         end
13101
13102       return Cmt(Cb("fenced_div_level"), check_div_level)
13103     end
13104
13105     local function check_indent()
13106       local compare_indent =
13107         function(s, i, indent_table, -- luacheck: ignore s i
13108           fenced_div_num_opening_indents, fenced_div_level)
13109           fenced_div_level = tonumber(fenced_div_level)
13110           local num_current_indents
13111             = ( indent_table.current_line_indents ~= nil and
13112               #indent_table.current_line_indents) or 0

```

```

13113         local num_opening_indents
13114             = fenced_div_num_opening_indents[fenced_div_level]
13115         return num_current_indents == num_opening_indents
13116     end
13117
13118     return Cmt( Cb("indent_info")
13119                 * Cb("fenced_div_num_opening_indents")
13120                 * Cb("fenced_div_level"), compare_indent)
13121 end
13122
13123 local fencestart = is_inside_div()
13124                 * fenced_div_end
13125                 * check_indent()
13126
13127 if not blank_before_div_fence then
13128     self.update_rule("EndlineExceptions", function(previous_pattern)
13129         if previous_pattern == nil then
13130             previous_pattern = parsers.EndlineExceptions
13131         end
13132         return previous_pattern + fencestart
13133     end)
13134 end
13135 end
13136 }
13137 end

```

### 3.1.7.8 Header Attributes

The `extensions.header_attributes` function implements the Pandoc header attribute syntax extension.

```

13138 M.extensions.header_attributes = function()
13139     return {
13140         name = "built-in header_attributes syntax extension",
13141         extend_writer = function()
13142             end, extend_reader = function(self)
13143                 local parsers = self.parsers
13144                 local writer = self.writer
13145
13146                 local function strip_atx_end(s)
13147                     return s:gsub("%s+##%s*$", "")
13148                 end
13149
13150                 local AtxHeading = Cg(parsers.heading_start, "level")
13151                     * parsers.optionalspace
13152                     * (C(((parsers.linechar
13153                         - (parsers.attributes
13154                             * parsers.optionalspace

```

```

13155             * parsers.newline))
13156         * (parsers.linechar
13157           - parsers.lbrace)^0)^1)
13158         / strip_atx_end
13159         / parsers.parse_heading_text)
13160     * Cg(Ct(parsers.newline
13161         + (parsers.attributes
13162           * parsers.optionalspace
13163           * parsers.newline)), "attributes")
13164     * Cb("level")
13165     * Cb("attributes")
13166     / writer.heading
13167
13168     local function strip_trailing_spaces(s)
13169         return s:gsub("%s*$", "")
13170     end
13171
13172     local heading_line = (parsers.linechar
13173                         - (parsers.attributes
13174                           * parsers.optionalspace
13175                           * parsers.newline))^1
13176                         - parsers.thematic_break_lines
13177
13178     local heading_text
13179     = heading_line
13180     * ( (V("Endline") / "\n")
13181       * (heading_line - parsers.heading_level))^0
13182     * parsers.newline^-1
13183
13184     local SettextHeading
13185     = parsers.freeze_trail * parsers.check_trail_no_rem
13186     * #(heading_text
13187       * (parsers.attributes
13188         * parsers.optionalspace
13189         * parsers.newline)^-1
13190       * parsers.check_minimal_indent
13191       * parsers.check_trail
13192       * parsers.heading_level)
13193     * Cs(heading_text) / strip_trailing_spaces
13194     / parsers.parse_heading_text
13195     * Cg(Ct((parsers.attributes
13196         * parsers.optionalspace
13197         * parsers.newline)^-1), "attributes")
13198     * parsers.check_minimal_indent_and_trail * parsers.heading_level
13199     * Cb("attributes")
13200     * parsers.newline
13201     * parsers.unfreeze_trail

```

```

13202         / writer.heading
13203
13204     local Heading = AtxHeading + SetextHeading
13205     self.update_rule("Heading", Heading)
13206 end
13207 }
13208 end

```

### 3.1.7.9 Inline Code Attributes

The `extensions.inline_code_attributes` function implements the Pandoc inline code attribute syntax extension.

```

13209 M.extensions.inline_code_attributes = function()
13210   return {
13211     name = "built-in inline_code_attributes syntax extension",
13212     extend_writer = function()
13213     end, extend_reader = function(self)
13214       local writer = self.writer
13215
13216       local CodeWithAttributes = parsers.inticks
13217         * Ct(parsers.attributes)
13218         / writer.code
13219
13220       self.insert_pattern("Inline before Code",
13221         CodeWithAttributes,
13222         "CodeWithAttributes")
13223     end
13224   }
13225 end

```

### 3.1.7.10 Line Blocks

The `extensions.line_blocks` function implements the Pandoc line block syntax extension.

```

13226 M.extensions.line_blocks = function()
13227   return {
13228     name = "built-in line_blocks syntax extension",
13229     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.

```

13230     function self.lineblock(lines)
13231       if not self.is_writing then return "" end
13232       local buffer = {}
13233       for i = 1, #lines - 1 do
13234         buffer[#buffer + 1] = { lines[i], self.hard_line_break }
13235       end
13236       buffer[#buffer + 1] = lines[#lines]

```

```

13237
13238         return {"\\markdownRendererLineBlockBegin\\n"
13239                ,buffer,
13240                "\\n\\markdownRendererLineBlockEnd "}
13241     end
13242 end, extend_reader = function(self)
13243     local parsers = self.parsers
13244     local writer = self.writer
13245
13246     local LineBlock
13247     = Ct((Cs(( (parsers.pipe * parsers.space) / ""
13248                * ((parsers.space)/entities.char_entity("nbsp"))^0
13249                * parsers.linechar^0 * (parsers.newline/""))
13250                * (-parsers.pipe
13251                  * (parsers.space^1/" ")
13252                  * parsers.linechar^1
13253                  * (parsers.newline/""))
13254                  ^0)
13255                / self.parser_functions.parse_inlines)^1)
13256     / writer.lineblock
13257
13258     self.insert_pattern("Block after Blockquote",
13259                        LineBlock, "LineBlock")
13260 end
13261 }
13262 end

```

### 3.1.7.11 Marked text

The `extensions.mark` function implements the Pandoc mark syntax extension.

```

13263 M.extensions.mark = function()
13264     return {
13265         name = "built-in mark syntax extension",
13266         extend_writer = function(self)

```

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

```

13267         function self.mark(s)
13268             if self.flatten_inlines then return s end
13269             return {"\\markdownRendererMark{" , s, "}" }
13270         end
13271     end, extend_reader = function(self)
13272         local parsers = self.parsers
13273         local writer = self.writer
13274
13275         local doubleequals = P("==")
13276
13277         local Mark

```

```

13278         = parsers.between(V("Inline"), doubleequals, doubleequals)
13279         / function (inlines) return writer.mark(inlines) end
13280
13281         self.add_special_character("=")
13282         self.insert_pattern("Inline before LinkAndEmph",
13283                             Mark, "Mark")
13284     end
13285 }
13286 end

```

### 3.1.7.12 Link Attributes

The `extensions.link_attributes` function implements the Pandoc link attribute syntax extension.

```

13287 M.extensions.link_attributes = function()
13288     return {
13289         name = "built-in link_attributes syntax extension",
13290         extend_writer = function()
13291         end, extend_reader = function(self)
13292             local parsers = self.parsers
13293             local options = self.options
13294

```

The following patterns define link reference definitions with attributes.

```

13295         local define_reference_parser
13296         = (parsers.check_trail / "")
13297         * parsers.link_label
13298         * parsers.colon
13299         * parsers.spnlc * parsers.url
13300         * ( parsers.spnlc_sep * parsers.title
13301           * (parsers.spnlc * Ct(parsers.attributes))
13302           * parsers.only_blank
13303           + parsers.spnlc_sep * parsers.title * parsers.only_blank
13304           + Cc("") * (parsers.spnlc * Ct(parsers.attributes))
13305           * parsers.only_blank
13306           + Cc("") * parsers.only_blank)
13307
13308         local ReferenceWithAttributes = define_reference_parser
13309         / self.register_link
13310
13311         self.update_rule("Reference", ReferenceWithAttributes)
13312

```

The following patterns define direct and indirect links with attributes.

```

13313
13314         local LinkWithAttributesAndEmph
13315         = Ct(parsers.link_and_emph_table * Cg(Cc(true),
13316         "match_link_attributes"))

```



```

13317         / self.defer_link_and_emphasis_processing
13318
13319     self.update_rule("LinkAndEmph", LinkWithAttributesAndEmph)
13320

```

The following patterns define autolinks with attributes.

```

13321     local AutoLinkUrlWithAttributes
13322         = parsers.auto_link_url
13323         * Ct(parsers.attributes)
13324         / self.auto_link_url
13325
13326     self.insert_pattern("Inline before AutoLinkUrl",
13327         AutoLinkUrlWithAttributes,
13328         "AutoLinkUrlWithAttributes")
13329
13330     local AutoLinkEmailWithAttributes
13331         = parsers.auto_link_email
13332         * Ct(parsers.attributes)
13333         / self.auto_link_email
13334
13335     self.insert_pattern("Inline before AutoLinkEmail",
13336         AutoLinkEmailWithAttributes,
13337         "AutoLinkEmailWithAttributes")
13338
13339     if options.relativeReferences then
13340
13341         local AutoLinkRelativeReferenceWithAttributes
13342             = parsers.auto_link_relative_reference
13343             * Ct(parsers.attributes)
13344             / self.auto_link_url
13345
13346         self.insert_pattern(
13347             "Inline before AutoLinkRelativeReference",
13348             AutoLinkRelativeReferenceWithAttributes,
13349             "AutoLinkRelativeReferenceWithAttributes")
13350
13351     end
13352
13353 end
13354 }
13355 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.

```
13356 M.extensions.notes = function(notes, inline_notes)
13357   assert(notes or inline_notes)
13358   return {
13359     name = "built-in notes syntax extension",
13360     extend_writer = function(self)
```

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

```
13361     function self.note(s)
13362       if self.flatten_inlines then return "" end
13363       return {"\\markdownRendererNote{",s,""}
13364     end
13365   end, extend_reader = function(self)
13366     local parsers = self.parsers
13367     local writer = self.writer
13368
13369     local rawnotes = parsers.rawnotes
13370
13371     if inline_notes then
13372       local InlineNote
13373       = parsers.circumflex
13374       * ( parsers.link_label
13375         / self.parser_functions.parse_inlines_no_inline_note)
13376       / writer.note
13377
13378       self.insert_pattern("Inline after LinkAndEmph",
13379         InlineNote, "InlineNote")
13380     end
13381     if notes then
13382       local function strip_first_char(s)
13383         return s:sub(2)
13384       end
13385
13386       local RawNoteRef
13387       = #(parsers.lbracket * parsers.circumflex)
13388       * parsers.link_label / strip_first_char
13389
13390       -- like indirect_link
13391       local function lookup_note(ref)
13392         return writer.defer_call(function()
13393           local found = rawnotes[self.normalize_tag(ref)]
13394           if found then
13395             return writer.note(
13396               self.parser_functions.parse_blocks_nested(found))
13397           else
```

```

13398         local text = string.format(
13399             'Undefined note reference "%s"', ref)
13400         local more = string.format(
13401             "Look for the text `[~%s]`.", ref)
13402         return {writer.warning(text, more), "[",
13403             self.parser_functions.parse_inlines("^" .. ref), "]" }
13404     end
13405 end)
13406 end
13407
13408 local function register_note(ref, rawnote)
13409     local normalized_tag = self.normalize_tag(ref)
13410     if rawnotes[normalized_tag] == nil then
13411         rawnotes[normalized_tag] = rawnote
13412         return ""
13413     else
13414         local text
13415         = string.format('Multiply defined note reference "%s"',
13416             ref)
13417         local more
13418         = string.format("Look for the text `[~%s]: ...`.", ref)
13419         return writer.warning(text, more)
13420     end
13421 end
13422
13423 local NoteRef = RawNoteRef / lookup_note
13424
13425 local optionally_indented_line
13426     = parsers.check_optional_indent_and_any_trail * parsers.line
13427
13428 local blank
13429     = parsers.check_optional_blank_indent_and_any_trail
13430     * parsers.optionalspace * parsers.newline
13431
13432 local chunk
13433     = Cs(parsers.line
13434         * (optionally_indented_line - blank)^0)
13435
13436 local indented_blocks = function(bl)
13437     return Cs( bl
13438         * ( blank^1 * (parsers.check_optional_indent / "")
13439             * parsers.check_code_trail
13440             * -parsers.blankline * bl)^0)
13441 end
13442
13443 local NoteBlock
13444     = parsers.check_trail_no_rem

```

```

13445             * RawNoteRef * parsers.colon
13446             * parsers.spnlc * indented_blocks(chunk)
13447             / register_note
13448
13449         self.update_rule("Reference", function(previous_pattern)
13450             if previous_pattern == nil then
13451                 previous_pattern = parsers.Reference
13452             end
13453             return NoteBlock + previous_pattern
13454         end)
13455
13456         self.insert_pattern("Inline before LinkAndEmph",
13457                             NoteRef, "NoteRef")
13458     end
13459
13460     self.add_special_character("^")
13461 end
13462 }
13463 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.

```

13464 M.extensions.pipe_tables = function(table_captions, table_attributes)
13465
13466     local function make_pipe_table_rectangular(rows)
13467         local num_columns = #rows[2]
13468         local rectangular_rows = {}
13469         for i = 1, #rows do
13470             local row = rows[i]
13471             local rectangular_row = {}
13472             for j = 1, num_columns do
13473                 rectangular_row[j] = row[j] or ""
13474             end
13475             table.insert(rectangular_rows, rectangular_row)
13476         end
13477         return rectangular_rows
13478     end
13479
13480     local function pipe_table_row(allow_empty_first_column
13481                                   , nonempty_column
13482                                   , column_separator

```

```

13483                                     , column)
13484     local row_beginning
13485     if allow_empty_first_column then
13486         row_beginning = -- empty first column
13487                         #(parsers.spacechar^4
13488                           * column_separator)
13489                         * parsers.optionalspace
13490                         * column
13491                         * parsers.optionalspace
13492                         -- non-empty first column
13493                         + parsers.nonindentspace
13494                         * nonempty_column~-1
13495                         * parsers.optionalspace
13496     else
13497         row_beginning = parsers.nonindentspace
13498                         * nonempty_column~-1
13499                         * parsers.optionalspace
13500     end
13501
13502     return Ct(row_beginning
13503               * (-- single column with no leading pipes
13504                 #(column_separator
13505                   * parsers.optionalspace
13506                   * parsers.newline)
13507                 * column_separator
13508                 * parsers.optionalspace
13509                 -- single column with leading pipes or
13510                 -- more than a single column
13511                 + (column_separator
13512                   * parsers.optionalspace
13513                   * column
13514                   * parsers.optionalspace)^1
13515                 * (column_separator
13516                   * parsers.optionalspace)^-1))
13517 end
13518
13519 return {
13520     name = "built-in pipe_tables syntax extension",
13521     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.

```

13522     function self.table(rows, caption, attributes)
13523         if not self.is_writing then return "" end
13524         local buffer = {}
13525         if attributes ~= nil then
13526             table.insert(buffer,

```

```

13527         "\\markdownRendererTableAttributeContextBegin\n")
13528     table.insert(buffer, self.attributes(attributes))
13529 end
13530 table.insert(buffer,
13531     {"\\markdownRendererTable{",
13532         caption or "", "}{" , #rows - 1, "}{" ,
13533         #rows[1], "}")
13534     local temp = rows[2] -- put alignments on the first row
13535     rows[2] = rows[1]
13536     rows[1] = temp
13537     for i, row in ipairs(rows) do
13538         table.insert(buffer, "{")
13539         for _, column in ipairs(row) do
13540             if i > 1 then -- do not use braces for alignments
13541                 table.insert(buffer, "{")
13542             end
13543             table.insert(buffer, column)
13544             if i > 1 then
13545                 table.insert(buffer, "}")
13546             end
13547         end
13548         table.insert(buffer, "}")
13549     end
13550     if attributes ~= nil then
13551         table.insert(buffer,
13552             "\\markdownRendererTableAttributeContextEnd{")
13553     end
13554     return buffer
13555 end
13556 end, extend_reader = function(self)
13557     local parsers = self.parsers
13558     local writer = self.writer
13559
13560     local table_hline_separator = parsers.pipe + parsers.plus
13561
13562     local table_hline_column = (parsers.dash
13563         - #(parsers.dash
13564             * (parsers.spacechar
13565                 + table_hline_separator
13566                 + parsers.newline)))^1
13567         * (parsers.colon * Cc("r")
13568             + parsers.dash * Cc("d"))
13569         + parsers.colon
13570         * (parsers.dash
13571             - #(parsers.dash
13572                 * (parsers.spacechar
13573                     + table_hline_separator

```

```

13574             + parsers.newline)))^1
13575         * (parsers.colon * Cc("c")
13576           + parsers.dash * Cc("l"))
13577
13578     local table_hline = pipe_table_row(false
13579                                       , table_hline_column
13580                                       , table_hline_separator
13581                                       , table_hline_column)
13582
13583     local table_caption_beginning
13584     = ( parsers.check_minimal_blank_indent_and_any_trail_no_rem
13585       * parsers.optionalspace * parsers.newline)^0
13586       * parsers.check_minimal_indent_and_trail
13587       * (P("Table")^-1 * parsers.colon)
13588       * parsers.optionalspace
13589
13590     local function strip_trailing_spaces(s)
13591       return s:gsub("%s*$","")
13592     end
13593
13594     local table_row
13595     = pipe_table_row(true
13596                     , (C((parsers.linechar - parsers.pipe)^1)
13597                       / strip_trailing_spaces
13598                       / self.parser_functions.parse_inlines)
13599                     , parsers.pipe
13600                     , (C((parsers.linechar - parsers.pipe)^0)
13601                       / strip_trailing_spaces
13602                       / self.parser_functions.parse_inlines))
13603
13604     local table_caption
13605     if table_captions then
13606       table_caption = #table_caption_beginning
13607                     * table_caption_beginning
13608       if table_attributes then
13609         table_caption = table_caption
13610                       * (C(((( parsers.linechar
13611                               - (parsers.attributes
13612                                 * parsers.optionalspace
13613                                 * parsers.newline
13614                                 * -( parsers.optionalspace
13615                                   * parsers.linechar))))
13616                       + ( parsers.newline
13617                         * #( parsers.optionalspace
13618                           * parsers.linechar)
13619                         * C(parsers.optionalspace)
13620                         / writer.space))

```

```

13621             * (parsers.linechar
13622               - parsers.lbrace)^0)^1)
13623             / strip_trailing_spaces
13624             / self.parser_functions.parse_inlines)
13625         * (parsers.newline
13626           + ( Ct(parsers.attributes)
13627             * parsers.optionalspace
13628             * parsers.newline))
13629     else
13630         table_caption = table_caption
13631         * C(( parsers.linechar
13632           + ( parsers.newline
13633             * #( parsers.optionalspace
13634               * parsers.linechar)
13635             * C(parsers.optionalspace)
13636             / writer.space))^1)
13637         / self.parser_functions.parse_inlines
13638         * parsers.newline
13639     end
13640 else
13641     table_caption = parsers.fail
13642 end
13643
13644 local PipeTable
13645 = Ct( table_row * parsers.newline
13646       * (parsers.check_minimal_indent_and_trail / {})
13647       * table_hline * parsers.newline
13648       * ( (parsers.check_minimal_indent / {})
13649         * table_row * parsers.newline)^0)
13650 / make_pipe_table_rectangular
13651 * table_caption^-1
13652 / writer.table
13653
13654 self.insert_pattern("Block after Blockquote",
13655                   PipeTable, "PipeTable")
13656 end
13657 }
13658 end

```

### 3.1.7.15 Raw Attributes

The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```

13659 M.extensions.raw_inline = function()
13660   return {
13661     name = "built-in raw_inline syntax extension",
13662     extend_writer = function(self)

```



```

13663     local options = self.options
13664

```

Define `writer->rawInline` as a function that will transform an input inline raw span `s` with the raw attribute `attr` to the output format.

```

13665     function self.rawInline(s, attr)
13666         if not self.is_writing then return "" end
13667         if self.flatten_inlines then return s end
13668         local name = util.cache_verbatim(options.cacheDir, s)
13669         return {"\\markdownRendererInputRawInline{",
13670             name,"}{", self.string(attr),""}
13671     end
13672 end, extend_reader = function(self)
13673     local writer = self.writer
13674
13675     local RawInline = parsers.inticks
13676                     * parsers.raw_attribute
13677                     / writer.rawInline
13678
13679     self.insert_pattern("Inline before Code",
13680                         RawInline, "RawInline")
13681 end
13682 }
13683 end

```

### 3.1.7.16 Strike-Through

The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```

13684 M.extensions.strike_through = function()
13685     return {
13686         name = "built-in strike_through syntax extension",
13687         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.

```

13688         function self.strike_through(s)
13689             if self.flatten_inlines then return s end
13690             return {"\\markdownRendererStrikeThrough{",s,""}
13691         end
13692     end, extend_reader = function(self)
13693         local parsers = self.parsers
13694         local writer = self.writer
13695
13696         local StrikeThrough = (
13697             parsers.between(parsers.Inline, parsers.doubletildes,
13698                             parsers.doubletildes)
13699         ) / writer.strike_through

```

```

13700
13701     self.insert_pattern("Inline after LinkAndEmph",
13702                         StrikeThrough, "StrikeThrough")
13703
13704     self.add_special_character("~")
13705 end
13706 }
13707 end

```

### 3.1.7.17 Subscripts

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

```

13708 M.extensions.subscripts = function()
13709   return {
13710     name = "built-in subscripts syntax extension",
13711     extend_writer = function(self)

```

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

```

13712     function self.subscript(s, parsed)
13713       if self.flatten_inlines then return s end
13714       return {"\\markdownRendererSubscript{" ,parsed,""} }
13715     end
13716   end, extend_reader = function(self)
13717     local parsers = self.parsers
13718     local writer = self.writer
13719
13720     local Subscript = (
13721       parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
13722     ) / function(s)
13723       return writer.subscript(
13724         s, self.parser_functions.parse_inlines_identity(s[1])
13725       )
13726     end
13727
13728     self.insert_pattern("Inline after LinkAndEmph",
13729                         Subscript, "Subscript")
13730
13731     self.add_special_character("~")
13732   end
13733 }
13734 end

```

### 3.1.7.18 Superscripts

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

```

13735 M.extensions.superscripts = function()
13736   return {
13737     name = "built-in superscripts syntax extension",
13738     extend_writer = function(self)

```

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

```

13739     function self.superscript(s, parsed)
13740       if self.flatten_inlines then return s end
13741       return {"\\markdownRendererSuperscript{" ,parsed,""} }
13742     end
13743   end, extend_reader = function(self)
13744     local parsers = self.parsers
13745     local writer = self.writer
13746
13747     local Superscript = (
13748       parsers.between(parsers.Str, parsers.circumflex,
13749         parsers.circumflex)
13750     ) / function(s)
13751       return writer.superscript(
13752         s, self.parser_functions.parse_inlines_identity(s[1])
13753       )
13754     end
13755
13756     self.insert_pattern("Inline after LinkAndEmph",
13757       Superscript, "Superscript")
13758
13759     self.add_special_character("^")
13760   end
13761 }
13762 end

```

### 3.1.7.19 T<sub>E</sub>X Math

The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```

13763 M.extensions.tex_math = function(tex_math_dollars,
13764                                   tex_math_single_backslash,
13765                                   tex_math_double_backslash)
13766   return {
13767     name = "built-in tex_math syntax extension",
13768     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.

```

13769     function self.display_math(s, parsed)
13770       if self.flatten_inlines then return s end
13771       return {"\\markdownRendererDisplayMath{" ,parsed,""} }

```

13772           end

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

```
13773       function self.inline_math(s, parsed)
13774       if self.flatten_inlines then return s end
13775       return {"\\markdownRendererInlineMath{" ,parsed,""} }
13776       end
13777 end, extend_reader = function(self)
13778   local parsers = self.parsers
13779   local writer = self.writer
13780
13781   local function between(p, starter, ender)
13782     return (starter * Cs(p * (p - ender)^0) * ender)
13783   end
13784
13785   local function strip_preceding_spaces(str)
13786     return str:gsub("^%s*(.-)$", "%1")
13787   end
13788
13789   local allowed_before_closing
13790     = B( parsers.backslash * parsers.any
13791         + parsers.any * (parsers.any - parsers.backslash))
13792
13793   local allowed_before_closing_no_space
13794     = B( parsers.backslash * parsers.any
13795         + parsers.any * (parsers.nonspacechar - parsers.backslash))
13796
```

The following patterns implement the Pandoc dollar math syntax extension.

```
13797   local dollar_math_content
13798     = (parsers.newline * (parsers.check_optional_indent / "")
13799       + parsers.backslash^-1
13800       * parsers.linechar)
13801       - parsers.blankline^2
13802       - parsers.dollar
13803
13804   local inline_math_opening_dollars = parsers.dollar
13805                                    * #(parsers.nonspacechar)
13806
13807   local inline_math_closing_dollars
13808     = allowed_before_closing_no_space
13809       * parsers.dollar
13810       * -#(parsers.digit)
13811
13812   local inline_math_dollars = between(Cs( dollar_math_content),
13813                                       inline_math_opening_dollars,
13814                                       inline_math_closing_dollars)
```

```

13815
13816     local display_math_opening_dollars = parsers.dollar
13817                                     * parsers.dollar
13818
13819     local display_math_closing_dollars = parsers.dollar
13820                                     * parsers.dollar
13821
13822     local display_math_dollars = between(Cs( dollar_math_content),
13823   display_math_opening_dollars,
13824   display_math_closing_dollars)

```

The following patterns implement the Pandoc single and double backslash math syntax extensions.

```

13825     local backslash_math_content
13826     = (parsers.newline * (parsers.check_optional_indent / ""))
13827       + parsers.linechar)
13828       - parsers.blankline^2

```

The following patterns implement the Pandoc double backslash math syntax extension.

```

13829     local inline_math_opening_double = parsers.backslash
13830                                     * parsers.backslash
13831                                     * parsers.lparent
13832
13833     local inline_math_closing_double = allowed_before_closing
13834                                     * parsers.spacechar^0
13835                                     * parsers.backslash
13836                                     * parsers.backslash
13837                                     * parsers.rparent
13838
13839     local inline_math_double = between(Cs( backslash_math_content),
13840                                       inline_math_opening_double,
13841                                       inline_math_closing_double)
13842                                   / strip_preceding_whitespaces
13843
13844     local display_math_opening_double = parsers.backslash
13845                                     * parsers.backslash
13846                                     * parsers.lbracket
13847
13848     local display_math_closing_double = allowed_before_closing
13849                                     * parsers.spacechar^0
13850                                     * parsers.backslash
13851                                     * parsers.backslash
13852                                     * parsers.rbracket
13853
13854     local display_math_double = between(Cs( backslash_math_content),
13855                                       display_math_opening_double,
13856                                       display_math_closing_double)

```

```

13857                                     / strip_preceding_whitespaces
The following patterns implement the Pandoc single backslash math syntax extension.
13858     local inline_math_opening_single = parsers.backslash
13859   * parsers.lparent
13860
13861     local inline_math_closing_single = allowed_before_closing
13862   * parsers.spacechar^0
13863   * parsers.backslash
13864   * parsers.rparent
13865
13866     local inline_math_single = between(Cs( backslash_math_content),
13867   inline_math_opening_single,
13868   inline_math_closing_single)
13869                                     / strip_preceding_whitespaces
13870
13871     local display_math_opening_single = parsers.backslash
13872   * parsers.lbracket
13873
13874     local display_math_closing_single = allowed_before_closing
13875   * parsers.spacechar^0
13876   * parsers.backslash
13877   * parsers.rbracket
13878
13879     local display_math_single = between(Cs( backslash_math_content),
13880   display_math_opening_single,
13881   display_math_closing_single)
13882                                     / strip_preceding_whitespaces
13883
13884     local display_math = parsers.fail
13885
13886     local inline_math = parsers.fail
13887
13888     if tex_math_dollars then
13889         display_math = display_math + display_math_dollars
13890         inline_math = inline_math + inline_math_dollars
13891     end
13892
13893     if tex_math_double_backslash then
13894         display_math = display_math + display_math_double
13895         inline_math = inline_math + inline_math_double
13896     end
13897
13898     if tex_math_single_backslash then
13899         display_math = display_math + display_math_single
13900         inline_math = inline_math + inline_math_single
13901     end
13902

```

```

13903     local TexMath = display_math
13904         / function(s)
13905             return writer.display_math(
13906                 s, self.parser_functions.parse_inlines_math(s)
13907             )
13908         end
13909     + inline_math
13910     / function(s)
13911         return writer.inline_math(
13912             s, self.parser_functions.parse_inlines_math(s)
13913         )
13914     end
13915
13916     self.insert_pattern("Inline after LinkAndEmph",
13917         TexMath, "TexMath")
13918
13919     if tex_math_dollars then
13920         self.add_special_character("$")
13921     end
13922
13923     if tex_math_single_backslash or tex_math_double_backslash then
13924         self.add_special_character("\\")
13925         self.add_special_character("[")
13926         self.add_special_character("]")
13927         self.add_special_character("(")
13928         self.add_special_character("(")
13929     end
13930 end
13931 }
13932 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.

```

13933 M.extensions.jekyll_data = function(expect_jekyll_data,
13934                                     ensure_jekyll_data)
13935     return {
13936         name = "built-in jekyll_data syntax extension",
13937         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.

```

13938     function self.jekyllData(d, t, p)
13939         if not self.is_writing then return "" end
13940
13941         local buf = {}
13942
13943         local keys = {}
13944         for k, _ in pairs(d) do
13945             table.insert(keys, k)
13946         end

```

For reproducibility, sort the keys. For mixed string-and-numeric keys, sort numeric keys before string keys.

```

13947         table.sort(keys, function(first, second)
13948             if type(first) ~= type(second) then
13949                 return type(first) < type(second)
13950             else
13951                 return first < second
13952             end
13953         end)
13954
13955         if not p then
13956             table.insert(buf, "\\markdownRendererJekyllDataBegin")
13957         end
13958
13959         local is_sequence = false
13960         if #d > 0 and #d == #keys then
13961             for i=1, #d do
13962                 if d[i] == nil then
13963                     goto not_a_sequence
13964                 end
13965             end
13966             is_sequence = true
13967         end
13968         ::not_a_sequence::
13969
13970         if is_sequence then
13971             table.insert(buf,
13972                 "\\markdownRendererJekyllDataSequenceBegin{")
13973             table.insert(buf, self.identifier(p or "null"))
13974             table.insert(buf, "{")
13975             table.insert(buf, #keys)
13976             table.insert(buf, "}")
13977         else

```



```

13978         table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
13979         table.insert(buf, self.identifier(p or "null"))
13980         table.insert(buf, "{")
13981         table.insert(buf, #keys)
13982         table.insert(buf, "}")
13983     end
13984
13985     for _, k in ipairs(keys) do
13986         local v = d[k]
13987         local typ = type(v)
13988         k = tostring(k or "null")
13989         if typ == "table" and next(v) ~= nil then
13990             table.insert(
13991                 buf,
13992                 self.jekyllData(v, t, k)
13993             )
13994         else
13995             k = self.identifier(k)
13996             v = tostring(v)
13997             if typ == "boolean" then
13998                 table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
13999                 table.insert(buf, k)
14000                 table.insert(buf, "{")
14001                 table.insert(buf, v)
14002                 table.insert(buf, "}")
14003             elseif typ == "number" then
14004                 table.insert(buf, "\\markdownRendererJekyllDataNumber{")
14005                 table.insert(buf, k)
14006                 table.insert(buf, "{")
14007                 table.insert(buf, v)
14008                 table.insert(buf, "}")
14009             elseif typ == "string" then
14010                 table.insert(buf,
14011                     "\\markdownRendererJekyllDataProgrammaticString{")
14012                 table.insert(buf, k)
14013                 table.insert(buf, "{")
14014                 table.insert(buf, self.identifier(v))
14015                 table.insert(buf, "}")
14016                 table.insert(buf,
14017                     "\\markdownRendererJekyllDataTypographicString{")
14018                 table.insert(buf, k)
14019                 table.insert(buf, "{")
14020                 table.insert(buf, t(v))
14021                 table.insert(buf, "}")
14022             elseif typ == "table" then
14023                 table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
14024                 table.insert(buf, k)

```

```

14025         table.insert(buf, "}")
14026     else
14027         local error = self.error(format(
14028             "Unexpected type %s for value of "
14029             .. "YAML key %s.", typ, k))
14030         table.insert(buf, error)
14031     end
14032 end
14033 end
14034
14035 if is_sequence then
14036     table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
14037 else
14038     table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
14039 end
14040
14041 if not p then
14042     table.insert(buf, "\\markdownRendererJekyllDataEnd")
14043 end
14044
14045 return buf
14046 end
14047 end, extend_reader = function(self)
14048     local parsers = self.parsers
14049     local writer = self.writer
14050
14051     local JekyllData
14052     = Cmt( C((parsers.line - P("---") - P("..."))~0)
14053         , function(s, i, text) -- luacheck: ignore s i
14054             local data
14055             local ran_ok, _ = pcall(function()
14056                 local tinyyaml = require("tinyyaml")
14057                 data = tinyyaml.parse(text, {timestamps=false})
14058             end)
14059             if ran_ok and data ~= nil then
14060                 return true, writer.jekyllData(data, function(s)
14061                     return self.parser_functions.parse_blocks_nested(s)
14062                 end, nil)
14063             else
14064                 return false
14065             end
14066         end
14067     )
14068
14069     local UnexpectedJekyllData
14070     = P("---")
14071     * parsers.blankline / 0

```

```

14072      -- if followed by blank, it's thematic break
14073      * #(-parsers.blankline)
14074      * JekyllData
14075      * (P("----") + P("..."))
14076
14077      local ExpectedJekyllData
14078      = ( P("----")
14079        * parsers.blankline / 0
14080        -- if followed by blank, it's thematic break
14081        * #(-parsers.blankline)
14082        )^-1
14083      * JekyllData
14084      * (P("----") + P("..."))^-1
14085
14086      if ensure_jekyll_data then
14087        ExpectedJekyllData = ExpectedJekyllData
14088                          * parsers.eof
14089      else
14090        ExpectedJekyllData = ( ExpectedJekyllData
14091                              * (V("Blank")^0 / writer.interblocksep)
14092                              )^-1
14093      end
14094
14095      self.insert_pattern("Block before Blockquote",
14096                        UnexpectedJekyllData, "UnexpectedJekyllData")
14097      if expect_jekyll_data then
14098        self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
14099      end
14100    end
14101  }
14102 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`.

```

14103 function M.new(options)

```

Make the `options` table inherit from the `defaultOptions` and `experimentalOptions` tables.

```

14104   options = options or {}
14105   setmetatable(options, { __index = function (_, key)
14106     return defaultOptions[key] end })
14107   if options.experimental then
14108     setmetatable(options, { __index = function (_, key)

```

```

14109         return experimentalOptions[key] end })
14110     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.

```

14111     local parser_convert = nil
14112     return function(input, include_flat_output)
14113         local function convert(input)
14114             if parser_convert == nil then

```

Lazy-load `markdown-parser.lua` and check that it originates from the same version of the Markdown package.

```

14115                 local parser = require("markdown-parser")
14116                 if metadata.version ~= parser.metadata.version then
14117                     warn("markdown.lua " .. metadata.version .. " used with " ..
14118                         "markdown-parser.lua " .. parser.metadata.version .. ".")
14119                 end
14120                 parser_convert = parser.new(options)
14121             end
14122             return parser_convert(input)
14123         end

```

If we cache markdown documents, produce the cache file and transform its filename to plain TeX output.

When determining the name of the cache file, create salt for the hashing function out of the package version and the passed options recognized by the Lua interface (see Section 2.1.3).

```

14124         local raw_output, flat_output
14125         if options.eagerCache or options.finalizeCache then
14126             local salt = util.salt(options)
14127             local name, result = util.cache(options.cacheDir, input, salt,
14128   convert, ".md.tex")
14129             raw_output = [[\input{}} .. name .. [[}\relax]]
14130             flat_output = function()
14131                 if result == nil then
14132                     local input_file = assert(io.open(name, "r"),
14133                         [[Could not open file "]] .. name .. [[ for reading]])
14134                     result = assert(input_file:read("*a"))
14135                     assert(input_file:close())
14136                 end
14137                 return result
14138             end

```

Otherwise, return the result of the conversion directly.

```

14139         else
14140             raw_output = convert(input)
14141             flat_output = function()

```

```

14142         return raw_output
14143     end
14144 end

```

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

```

14145     if options.finalizeCache then
14146         local file, mode
14147         if options.frozenCacheCounter > 0 then
14148             mode = "a"
14149         else
14150             mode = "w"
14151         end
14152         file = assert(io.open(options.frozenCacheFileName, mode),
14153             [[Could not open file ]] .. options.frozenCacheFileName
14154             .. [[ for writing]])
14155         assert(file:write(
14156             [[\expandafter\global\expandafter\def\csname ]]
14157             .. [[markdownFrozenCache]] .. options.frozenCacheCounter
14158             .. [[\endcsname{}]] .. raw_output .. [[]] .. "\n"))
14159         assert(file:close())
14160     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.

```

14161     if include_flat_output then
14162         return raw_output, flat_output
14163     else
14164         return raw_output
14165     end
14166 end
14167 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.

```

14168 function M.new(options)

```

Make the `options` table inherit from the `defaultOptions` and `experimentalOptions` tables.

```

14169     options = options or {}
14170     setmetatable(options, { __index = function (_, key)
14171         return defaultOptions[key] end })
14172     if options.experimental then
14173         setmetatable(options, { __index = function (_, key)
14174             return experimentalOptions[key] end })
14175     end

```

If the singleton cache contains a conversion function for the same `options`, reuse it.

```
14176   if options.singletonCache and singletonCache.convert then
14177       for k, v in pairs(defaultOptions) do
14178           if type(v) == "table" then
14179               for i = 1, math.max(#singletonCache.options[k], #options[k]) do
14180                   if singletonCache.options[k][i] ~= options[k][i] then
14181                       goto miss
14182                   end
14183               end

```

The `cacheDir` option is disregarded.

```
14184           elseif k ~= "cacheDir"
14185               and singletonCache.options[k] ~= options[k] then
14186               goto miss
14187           end
14188       end
14189       return singletonCache.convert
14190   end
14191   ::miss::

```

Apply built-in syntax extensions based on `options`.

```
14192   local extensions = {}
14193
14194   if #options.acronyms > 0 then
14195       local acronyms_extension = M.extensions.acronyms(options.acronyms)
14196       table.insert(extensions, acronyms_extension)
14197   end
14198
14199   if options.bracketedSpans then
14200       local bracketed_spans_extension = M.extensions.bracketed_spans()
14201       table.insert(extensions, bracketed_spans_extension)
14202   end
14203
14204   if options.contentBlocks then
14205       local content_blocks_extension = M.extensions.content_blocks(
14206           options.contentBlocksLanguageMap)
14207       table.insert(extensions, content_blocks_extension)
14208   end
14209
14210   if options.definitionLists then
14211       local definition_lists_extension = M.extensions.definition_lists(
14212           options.tightLists)
14213       table.insert(extensions, definition_lists_extension)
14214   end
14215
14216   if options.fencedCode then
14217       local fenced_code_extension = M.extensions.fenced_code(
14218           options.blankBeforeCodeFence,

```

```

14219         options.fencedCodeAttributes,
14220         options.rawAttribute)
14221     table.insert(extensions, fenced_code_extension)
14222 end
14223
14224 if options.fencedDivs then
14225     local fenced_div_extension = M.extensions.fenced_divs(
14226         options.blankBeforeDivFence)
14227     table.insert(extensions, fenced_div_extension)
14228 end
14229
14230 if options.headerAttributes then
14231     local header_attributes_extension = M.extensions.header_attributes()
14232     table.insert(extensions, header_attributes_extension)
14233 end
14234
14235 if options.inlineCodeAttributes then
14236     local inline_code_attributes_extension =
14237         M.extensions.inline_code_attributes()
14238     table.insert(extensions, inline_code_attributes_extension)
14239 end
14240
14241 if options.jekyllData then
14242     local jekyll_data_extension = M.extensions.jekyll_data(
14243         options.expectJekyllData, options.ensureJekyllData)
14244     table.insert(extensions, jekyll_data_extension)
14245 end
14246
14247 if options.linkAttributes then
14248     local link_attributes_extension =
14249         M.extensions.link_attributes()
14250     table.insert(extensions, link_attributes_extension)
14251 end
14252
14253 if options.lineBlocks then
14254     local line_block_extension = M.extensions.line_blocks()
14255     table.insert(extensions, line_block_extension)
14256 end
14257
14258 if options.mark then
14259     local mark_extension = M.extensions.mark()
14260     table.insert(extensions, mark_extension)
14261 end
14262
14263 if options.pipeTables then
14264     local pipe_tables_extension = M.extensions.pipe_tables(
14265         options.tableCaptions, options.tableAttributes)

```

```

14266     table.insert(extensions, pipe_tables_extension)
14267 end
14268
14269 if options.rawAttribute then
14270     local raw_inline_extension = M.extensions.raw_inline()
14271     table.insert(extensions, raw_inline_extension)
14272 end
14273
14274 if options.strikeThrough then
14275     local strike_through_extension = M.extensions.strike_through()
14276     table.insert(extensions, strike_through_extension)
14277 end
14278
14279 if options.subscripts then
14280     local subscript_extension = M.extensions.subscripts()
14281     table.insert(extensions, subscript_extension)
14282 end
14283
14284 if options.superscripts then
14285     local superscript_extension = M.extensions.superscripts()
14286     table.insert(extensions, superscript_extension)
14287 end
14288
14289 if options.texMathDollars or
14290     options.texMathSingleBackslash or
14291     options.texMathDoubleBackslash then
14292     local tex_math_extension = M.extensions.tex_math(
14293         options.texMathDollars,
14294         options.texMathSingleBackslash,
14295         options.texMathDoubleBackslash)
14296     table.insert(extensions, tex_math_extension)
14297 end
14298
14299 if options.notes or options.inlineNotes then
14300     local notes_extension = M.extensions.notes(
14301         options.notes, options.inlineNotes)
14302     table.insert(extensions, notes_extension)
14303 end
14304
14305 if options.citations then
14306     local citations_extension
14307         = M.extensions.citations(options.citationNbsps)
14308     table.insert(extensions, citations_extension)
14309 end
14310
14311 if options.fancyLists then
14312     local fancy_lists_extension = M.extensions.fancy_lists()

```



```

14313     table.insert(extensions, fancy_lists_extension)
14314 end

```

Apply user-defined syntax extensions based on `options.extensions`.

```

14315 for _, user_extension_filename in ipairs(options.extensions) do
14316     local user_extension = (function(filename)

```

First, load and compile the contents of the user-defined syntax extension.

```

14317         local pathname = assert(util.find_file(filename),
14318             [[Could not locate user-defined syntax extension "]]
14319             .. filename)
14320         local input_file = assert(io.open(pathname, "r"),
14321             [[Could not open user-defined syntax extension "]]
14322             .. pathname .. [{" for reading}]]
14323         local input = assert(input_file:read("*a"))
14324         assert(input_file:close())
14325         local user_extension, err = load([[
14326             local sandbox = {}
14327             setmetatable(sandbox, {__index = _G})
14328             _ENV = sandbox
14329         ]] .. input)()
14330         assert(user_extension,
14331             [[Failed to compile user-defined syntax extension "]]
14332             .. pathname .. [{": "}] .. (err or [{"}]))

```

Then, validate the user-defined syntax extension.

```

14333         assert(user_extension.api_version ~= nil,
14334             [[User-defined syntax extension "]] .. pathname
14335             .. [{" does not specify mandatory field "api_version"}]])
14336         assert(type(user_extension.api_version) == "number",
14337             [[User-defined syntax extension "]] .. pathname
14338             .. [{" specifies field "api_version" of type "}]
14339             .. type(user_extension.api_version)
14340             .. [{" but "number" was expected}]]
14341         assert(user_extension.api_version > 0
14342             and user_extension.api_version
14343             <= metadata.user_extension_api_version,
14344             [[User-defined syntax extension "]] .. pathname
14345             .. [{" uses syntax extension API version "}]
14346             .. user_extension.api_version .. [{" but markdown.lua "}]
14347             .. metadata.version .. [{" uses API version "}]
14348             .. metadata.user_extension_api_version
14349             .. [{" , which is incompatible}]]
14350
14351         assert(user_extension.grammar_version ~= nil,
14352             [[User-defined syntax extension "]] .. pathname
14353             .. [{" does not specify mandatory field "grammar_version"}]])
14354         assert(type(user_extension.grammar_version) == "number",

```

```

14355     [[User-defined syntax extension "]] .. pathname
14356     .. [[[" specifies field "grammar_version" of type "]]
14357     .. type(user_extension.grammar_version)
14358     .. [[[" but "number" was expected]])
14359     assert(user_extension.grammar_version == metadata.grammar_version,
14360            [[User-defined syntax extension "]] .. pathname
14361            .. [[[" uses grammar version "]]
14362            .. user_extension.grammar_version
14363            .. [[[" but markdown.lua ]] .. metadata.version
14364            .. [[[" uses grammar version ]] .. metadata.grammar_version
14365            .. [[[" which is incompatible]])
14366
14367     assert(user_extension.finalize_grammar ~= nil,
14368            [[User-defined syntax extension "]] .. pathname
14369            .. [[[" does not specify mandatory "finalize_grammar" field]])
14370     assert(type(user_extension.finalize_grammar) == "function",
14371            [[User-defined syntax extension "]] .. pathname
14372            .. [[[" specifies field "finalize_grammar" of type "]]
14373            .. type(user_extension.finalize_grammar)
14374            .. [[[" 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.)

```

14375     local extension = {
14376         name = [[user-defined "]] .. pathname .. [[[" syntax extension]],
14377         extend_reader = user_extension.finalize_grammar,
14378         extend_writer = function() end,
14379     }
14380     return extension
14381 end)(user_extension_filename)
14382 table.insert(extensions, user_extension)
14383 end

```

Produce a conversion function from markdown to plain TeX.

```

14384 local writer = M.writer.new(options)
14385 local reader = M.reader.new(writer, options)
14386 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`.

```

14387 collectgarbage("collect")

```

Update the singleton cache.

```

14388 if options.singletonCache then
14389     local singletonCacheOptions = {}
14390     for k, v in pairs(options) do
14391         singletonCacheOptions[k] = v
14392     end

```

```

14393     setmetatable.singletonCacheOptions,
14394     { __index = function (_, key)
14395       return defaultOptions[key] end })
14396     singletonCache.options = singletonCacheOptions
14397     singletonCache.convert = convert
14398   end

```

Return the conversion function from markdown to plain T<sub>E</sub>X.

```

14399   return convert
14400 end
14401 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.

```

14402
14403 local input
14404 if input_filename then
14405   local input_file = assert(io.open(input_filename, "r"),
14406     [[Could not open file ]] .. input_filename .. [[ for reading]])
14407   input = assert(input_file:read("*a"))
14408   assert(input_file:close())
14409 else
14410   input = assert(io.read("*a"))
14411 end
14412

```

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

```

14413 local-lfs = require("lfs")
14414 if options.cacheDir and not-lfs.isdir(options.cacheDir) then
14415   assert(lfs.mkdir(options["cacheDir"]))
14416 end

```

If Kpathsea has not been loaded before or if LuaT<sub>E</sub>X has not yet been initialized, configure Kpathsea on top of loading it.

```

14417 local kpse
14418 (function()
14419   local should_initialize = package.loaded.kpse == nil
14420   or tex.initialize ~= nil
14421   kpse = require("kpse")
14422   if should_initialize then
14423     kpse.set_program_name("luatex")
14424   end
14425 end)()
14426 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.

```

14427 if metadata.version ~= md.metadata.version then
14428     warn("markdown-cli.lua " .. metadata.version .. " used with " ..
14429         "markdown.lua " .. md.metadata.version .. ".")
14430 end
14431
14432 local convert = md.new(options)
14433 local raw_output, flat_output = convert(input, true)
14434 local output
14435 if flat_output == nil then
14436     if options.eagerCache then
14437         warn("markdown.lua has not produced flat output, so I am using " ..
14438             "backwards-compatible raw output instead. This may cause " ..
14439             "the conversion result to be hidden behind "\\input".')
14440     end
14441     output = raw_output
14442 else
14443     output = flat_output()
14444 end
14445
14446 if output_filename then
14447     local output_file = assert(io.open(output_filename, "w"),
14448         [[Could not open file ]] .. output_filename .. [[ for writing]])
14449     assert(output_file:write(output))
14450     assert(output_file:close())
14451 else
14452     assert(io.write(output))
14453 end
14454
14454 if options.cacheDir then
14455     lfs.rmdir(options.cacheDir)
14456 end

```

Remove the `options.cacheDir` directory if it is empty.

## 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

```

14457 \ExplSyntaxOn
14458 \cs_if_free:NT
14459     \markdownInfo

```

```

14460 {
14461     \cs_new:Npn
14462         \markdownInfo #1
14463     {
14464         \msg_info:nne
14465             { markdown }
14466             { generic-message }
14467             { #1 }
14468     }
14469 }
14470 \cs_if_free:NT
14471     \markdownWarning
14472 {
14473     \cs_new:Npn
14474         \markdownWarning #1
14475     {
14476         \msg_warning:nne
14477             { markdown }
14478             { generic-message }
14479             { #1 }
14480     }
14481 }
14482 \cs_if_free:NT
14483     \markdownError
14484 {
14485     \cs_new:Npn
14486         \markdownError #1 #2
14487     {
14488         \msg_error:nnee
14489             { markdown }
14490             { generic-message-with-help-text }
14491             { #1 }
14492             { #2 }
14493     }
14494 }
14495 \msg_new:nnn
14496     { markdown }
14497     { generic-message }
14498     { #1 }
14499 \msg_new:nnnn
14500     { markdown }
14501     { generic-message-with-help-text }
14502     { #1 }
14503     { #2 }
14504 \cs_generate_variant:Nn
14505     \msg_info:nnn
14506     { nne }

```

```

14507 \cs_generate_variant:Nn
14508   \msg_warning:nnn
14509   { nne }
14510 \cs_generate_variant:Nn
14511   \msg_error:nnnn
14512   { nnee }
14513 \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.

```

14514 \ExplSyntaxOn
14515 \prop_new:N \g_@@_plain_tex_loaded_themes_linenos_prop
14516 \prop_new:N \g_@@_plain_tex_loaded_themes_versions_prop
14517 \cs_new:Nn
14518   \@@_plain_tex_load_theme:nnn
14519   {
14520     \prop_get:NnNTF
14521       \g_@@_plain_tex_loaded_themes_linenos_prop
14522       { #1 }
14523     \l_tmpa_tl
14524     {
14525       \prop_get:NnN
14526         \g_@@_plain_tex_loaded_themes_versions_prop
14527         { #1 }
14528       \l_tmpb_tl
14529       \str_if_eq:nVTF
14530       { #2 }
14531       \l_tmpb_tl
14532       {
14533         \msg_warning:nnnVn
14534           { markdown }
14535           { repeatedly-loaded-plain-tex-theme }
14536           { #1 }
14537         \l_tmpa_tl
14538         { #2 }
14539       }
14540     }
14541     \msg_error:nnnnVV
14542       { markdown }
14543       { different-versions-of-plain-tex-theme }
14544       { #1 }
14545       { #2 }
14546     \l_tmpb_tl
14547     \l_tmpa_tl

```

```

14548     }
14549   }
14550   {
14551     \prop_gput:Nnx
14552     \g_@@_plain_tex_loaded_themes_linenos_prop
14553     { #1 }
14554     { \tex_the:D \tex_inputlineno:D } % noqa: W200
14555     \prop_gput:Nnn
14556     \g_@@_plain_tex_loaded_themes_versions_prop
14557     { #1 }
14558     { #2 }

```

Load built-in plain TeX themes from the prop `\g_@@_plain_tex_built_in_themes_prop` and from the filesystem otherwise.

```

14559     \prop_if_in:NnTF
14560     \g_@@_plain_tex_built_in_themes_prop
14561     { #1 }
14562     {
14563       \msg_info:nnnn
14564       { markdown }
14565       { loading-built-in-plain-tex-theme }
14566       { #1 }
14567       { #2 }
14568       \prop_item:Nn
14569       \g_@@_plain_tex_built_in_themes_prop
14570       { #1 }
14571     }
14572     {
14573       \msg_info:nnnn
14574       { markdown }
14575       { loading-plain-tex-theme }
14576       { #1 }
14577       { #2 }
14578       \file_input:n
14579       { markdown theme #3 }
14580     }
14581   }
14582 }
14583 \msg_new:nnn
14584 { markdown }
14585 { loading-plain-tex-theme }
14586 { Loading-version~#2~of~plain~TeX~Markdown~theme~#1 }
14587 \msg_new:nnn
14588 { markdown }
14589 { loading-built-in-plain-tex-theme }
14590 { Loading-version~#2~of~built-in~plain~TeX~Markdown~theme~#1 }
14591 \msg_new:nnn

```

```

14592 { markdown }
14593 { repeatedly-loaded-plain-tex-theme }
14594 {
14595   Version~#3~of~plain~TeX~Markdown~theme~#1~was~previously~
14596   loaded~on~line~#2,~not~loading~it~again
14597 }
14598 \msg_new:nnn
14599 { markdown }
14600 { different-versions-of-plain-tex-theme }
14601 {
14602   Tried~to~load~version~#2~of~plain~TeX~Markdown~theme~#1~
14603   but~version~#3~has~already~been~loaded~on~line~#4
14604 }
14605 \cs_generate_variant:Nn
14606 \prop_gput:Nnn
14607 { Nnx }
14608 \cs_gset_eq:NN
14609 \@@_load_theme:nnn
14610 \@@_plain_tex_load_theme:nnn
14611 \cs_generate_variant:Nn
14612 \@@_load_theme:nnn
14613 { VeV }
14614 \cs_generate_variant:Nn
14615 \msg_error:nnnnnn
14616 { nnnnVV }
14617 \cs_generate_variant:Nn
14618 \msg_warning:nnnnn
14619 { nnnVn }

```

Developers can use the `\markdownLoadPlainTeXTheme` macro to load a corresponding plain T<sub>E</sub>X theme from within themes for higher-level T<sub>E</sub>X formats such as L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt.

```

14620 \cs_new:Npn
14621 \markdownLoadPlainTeXTheme
14622 {

```

First, we extract the name of the current theme from the `\g_@@_current_theme_tl` macro.

```

14623   \tl_set:NV
14624     \l_tmpa_tl
14625     \g_@@_current_theme_tl
14626   \tl_reverse:N
14627     \l_tmpa_tl
14628   \tl_set:Ne
14629     \l_tmpb_tl
14630   {
14631     \tl_tail:V
14632     \l_tmpa_tl

```



```

14633     }
14634     \tl_reverse:N
14635     \l_tmpb_tl

```

Next, we munge the theme name.

```

14636     \str_set:NV
14637     \l_tmpa_str
14638     \l_tmpb_tl
14639     \str_replace_all:Nnn
14640     \l_tmpa_str
14641     { / }
14642     { _ }

```

Finally, we load the plain  $\TeX$  theme.

```

14643     \@@_plain_tex_load_theme:VeV
14644     \l_tmpb_tl
14645     { \markdownThemeVersion }
14646     \l_tmpa_str
14647   }
14648   \cs_generate_variant:Nn
14649   \tl_set:Nn
14650   { Ne }
14651   \cs_generate_variant:Nn
14652   \@@_plain_tex_load_theme:nnn
14653   { VeV }

```

The [witiko/dot](#) theme nags users that they should use the name [witiko/diagrams@v1](#) instead.

```

14654   \prop_gput:Nnn
14655   \g_@@_plain_tex_built_in_themes_prop
14656   { witiko / dot }
14657   {
14658     \str_if_eq:enF
14659     { \markdownThemeVersion }
14660     { silent }
14661     {
14662       \markdownWarning
14663       {
14664         The~theme~name~"witiko/dot"~has~been~soft-deprecated.
14665         \iow_newline:
14666         Consider~changing~the~name~to~"witiko/diagrams@v1".
14667       }
14668     }

```

We enable the [fencedCode](#) Lua option.

```

14669     \markdownSetup { fencedCode }

```

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

```

14670     \cs_set_eq:NN

```

```

14671 \@@_dot_previous_definition:nnn
14672 \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:

```

14673 \regex_const:Nn
14674 \c_@@_dot_infostring_regex
14675 { ^dot(\s+(.+)?)? }
14676 \seq_new:N
14677 \l_@@_dot_matches_seq
14678 \markdownSetup {
14679   rendererPrototypes = {
14680     inputFencedCode = {
14681       \regex_extract_once:NnNTF
14682       \c_@@_dot_infostring_regex
14683       { #2 }
14684       \l_@@_dot_matches_seq
14685       {
14686         \@@_if_option:nF
14687         { frozenCache }
14688         {
14689           \sys_shell_now:n
14690           {
14691             if~!~test~-e~#1.pdf.source~
14692             ||~!~diff~#1~#1.pdf.source;
14693             then~
14694               dot~-Tpdf~-o~#1.pdf~#1;
14695               cp~#1~#1.pdf.source;
14696             fi
14697           }
14698         }
14699       }
14700     }
14701   }
14702 }

```

We include the typeset image using the image token renderer:

```

14699 \exp_args:NNne
14700 \exp_last_unbraced:No
14701 \markdownRendererImage
14702 {
14703   { Graphviz~image }
14704   { #1.pdf }
14705   { #1.pdf }
14706 }
14707 {
14708   \seq_item:Nn
14709   \l_@@_dot_matches_seq
14710   { 3 }
14711 }

```

```
14712          }
```

If the infostring does not start with `dot ...`, we use the previous definition of the fenced code token renderer prototype:

```
14713      {
14714          \@@_dot_previous_definition:nnn
14715          { #1 }
14716          { #2 }
14717          { #3 }
14718      }
14719  },
14720 },
14721 }
14722 }
```

The theme `witiko/diagrams` loads either the theme `witiko/dot` for version `v1` or the theme `witiko/diagrams/v2` for version `v2`.

```
14723 \prop_gput:Nnn
14724 \g_@@_plain_tex_built_in_themes_prop
14725 { witiko / diagrams }
14726 {
14727     \str_case:enF
14728     { \markdownThemeVersion }
14729     {
14730         { latest }
14731         {
14732             \msg_warning:nnnn
14733             { markdown }
14734             { pin-theme-version }
14735             { witiko/diagrams }
14736             { v2 }
14737             \markdownSetup
14738             {
14739                 import = witiko/diagrams/v2,
14740             }
14741         }
14742         { v2 }
14743         {
14744             \markdownSetup
14745             {
14746                 import = witiko/diagrams/v2,
14747             }
14748         }
14749         { v1 }
14750         {
14751             \markdownSetup
14752             {
14753                 import = witiko/dot@silent,
```

```

14754         }
14755     }
14756 }
14757 {
14758     \msg_error:nnen
14759     { markdown }
14760     { unknown-theme-version }
14761     { witiko/diagrams }
14762     { \markdownThemeVersion }
14763     { v1~and~v2 }
14764 }
14765 }
14766 \cs_generate_variant:Nn
14767   \msg_error:nnnnn
14768   { nnnen }
14769 \msg_new:nnn
14770   { markdown }
14771   { pin-theme-version }
14772   {
14773       Write~"#1@#2"~to~pin-version~"#2"~of~the~theme~"#1".
14774       This~will~keep~your~documents~from~suddenly~breaking
14775       when~we~have~released~future~versions~of~the~theme
14776       with~backwards~incompatible~syntax~and~behavior.
14777   }
14778 \msg_new:nnnn
14779   { markdown }
14780   { unknown-theme-version }
14781   { Unknown~version~"#2"~of~theme~"#1"~has~been~requested. }
14782   { Known~versions~are:~#3 }

```

Next, we implement the theme `witiko/diagrams/v2`.

```

14783 \prop_gput:Nnn
14784   \g_@@_plain_tex_built_in_themes_prop
14785   { witiko / diagrams / v2 }
14786   {

```

We enable the `fencedCode` and `fencedCodeAttributes` Lua option.

```

14787   \@@_setup:n
14788   {
14789       fencedCode = true,
14790       fencedCodeAttributes = true,
14791   }

```

Store the previous fenced code token renderer prototype.

```

14792   \cs_set_eq:NN
14793     \@@_diagrams_previous_fenced_code:nnn
14794     \markdownRendererInputFencedCodePrototype

```

Store the caption, the desired format, and the command to produce the diagram.

```

14795 \tl_new:N
14796 \l_@@_diagrams_caption_tl
14797 \tl_new:N
14798 \l_@@_diagrams_format_tl
14799 \tl_set:Nn
14800 \l_@@_diagrams_format_tl
14801 { pdf }
14802 \tl_new:N
14803 \l_@@_diagrams_command_tl
14804 \@@_setup:n
14805 {
14806     rendererPrototypes = {

```

Route attributes on fenced code blocks to the image attribute renderer prototypes.

```

14807     fencedCodeAttributeContextBegin = {
14808         \group_begin:
14809         \markdownRendererImageAttributeContextBegin
14810         \cs_set_eq:NN
14811         \@@_diagrams_previous_key_value:nn
14812         \markdownRendererAttributeKeyValuePrototype
14813         \@@_setup:n
14814         {
14815             rendererPrototypes = {
14816                 attributeKeyValue = {
14817                     \str_case:nnF
14818                     { ##1 }
14819                     {
14820                         { caption }
14821                         {
14822                             \tl_set:Nn
14823                             \l_@@_diagrams_caption_tl
14824                             { ##2 }
14825                         }
14826                     { format }
14827                     {
14828                         \tl_set:Nn
14829                         \l_@@_diagrams_format_tl
14830                         { ##2 }
14831                     }
14832                     { command }
14833                     {
14834                         \tl_set:Nn
14835                         \l_@@_diagrams_command_tl
14836                         { ##2 }
14837                     }
14838                 }
14839             {
14840                 \@@_diagrams_previous_key_value:nn

```

```

14841             { ##1 }
14842             { ##2 }
14843         }
14844     },
14845 },
14846 }
14847 },
14848 fencedCodeAttributeContextEnd = {
14849     \markdownRendererImageAttributeContextEnd
14850     \group_end:
14851 },
14852 },
14853 }
14854 \cs_new:Nn
14855   \@@_diagrams_render_diagram:nnnn
14856 {
14857   \@@_if_option:nF
14858     { frozenCache }
14859     {
14860       \sys_shell_now:n
14861       {
14862         if~!~test~-e~#2.source~
14863         ||~!~diff~#1~#2.source;
14864         then~
14865           (#3);
14866           cp~#1~#2.source;
14867         fi
14868       }
14869       \exp_args:NNnV
14870       \exp_last_unbraced:No
14871       \markdownRendererImage
14872       {
14873         { #4 }
14874         { #2 }
14875         { #2 }
14876       }
14877       \l_@@_diagrams_caption_tl
14878     }
14879 }

```

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.

```

14880   \prop_new:N
14881   \g_markdown_diagrams_infostrings_prop

```

If we know a processing function for a given infostring, use it.

```

14882   \@@_setup:n

```

```

14883 {
14884   rendererPrototypes = {
14885     inputFencedCode = {
14886       \prop_get:NnNTF
14887       \g_markdown_diagrams_infostrings_prop
14888       { #2 }
14889       \l_tmpa_tl
14890       {
14891         \cs_set:NV
14892         \@@_diagrams_infostrings_current:n
14893         \l_tmpa_tl
14894         \@@_diagrams_infostrings_current:n
14895         { #1 }
14896       }

```

Otherwise, use the previous fenced code token renderer prototype.

```

14897       {
14898         \@@_diagrams_previous_fenced_code:nnn
14899         { #1 }
14900         { #2 }
14901         { #3 }
14902       }
14903     },
14904   },
14905 }
14906 \cs_generate_variant:Nn
14907 \cs_set:Nn
14908 { NV }

```

Typeset fenced code with infostring `dot` using the command `dot` from the package `Graphviz`.

```

14909 \cs_set:Nn
14910 \@@_diagrams_infostrings_current:n
14911 {
14912   \tl_set:Nn
14913   \l_tmpb_tl
14914   { dot~~T }
14915   \tl_put_right:NV
14916   \l_tmpb_tl
14917   \l_@@_diagrams_format_tl
14918   \tl_put_right:Nn
14919   \l_tmpb_tl
14920   { ~-o~#1. }
14921   \tl_put_right:NV
14922   \l_tmpb_tl
14923   \l_@@_diagrams_format_tl
14924   \tl_put_right:Nn
14925   \l_tmpb_tl

```

```
14926      { ~#1 }
```

For the SVG format, use Inkscape to convert the resulting image to PDF.

```
14927      \str_if_eq:VnT
14928      \l_@@_diagrams_format_tl
14929      { svg }
14930      {
14931          \tl_put_right:Nn
14932          \l_tmpb_tl
14933          { ;~inkscape~#1. }
14934          \tl_put_right:NV
14935          \l_tmpb_tl
14936          \l_@@_diagrams_format_tl
14937          \tl_put_right:Nn
14938          \l_tmpb_tl
14939          { ---export-area-drawing---export-dpi=300~-o~#1.pdf }
14940      }
14941      \@@_diagrams_render_diagram:nnVn
14942      { #1 }
14943      { #1.pdf }
14944      \l_tmpb_tl
14945      { Graphviz~image }
14946  }
14947  \cs_generate_variant:Nn
14948  \@@_diagrams_render_diagram:nnnn
14949  { nnVn }
14950  \@@_tl_set_from_cs:NNn
14951  \l_tmpa_tl
14952  \@@_diagrams_infostrings_current:n
14953  { 1 }
14954  \prop_gput:NnV
14955  \g_markdown_diagrams_infostrings_prop
14956  { dot }
14957  \l_tmpa_tl
```

Typeset fenced code with infostring `mermaid` using the command `mmdc` from the npm package `@mermaid-js/mermaid-cli`. The exact command can be configured by redefining or appending to the `\g_@@_diagrams_mmdc_command_tl` token list, by redefining the `\mmdcCommand` macro, or using the HTML attribute `command` on the fenced code.

Unlike the token list, the macro can be redefined even before loading the Markdown package. Unlike the token list and the macro, the HTML attribute is local to a single fenced code.

```
14958      \tl_new:N
14959      \g_@@_diagrams_mmdc_command_tl
14960      \tl_gset:Nn
14961      \g_@@_diagrams_mmdc_command_tl
```



```

14962     {
14963         \tl_if_empty:NTF
14964         \l_@@_diagrams_command_tl
14965         { mmdc }
14966         {
14967             \tl_use:N
14968             \l_@@_diagrams_command_tl
14969         }
14970     }
14971 \cs_if_free:NT
14972 \mmdcCommand
14973 {
14974     \cs_new:Npn
14975     \mmdcCommand
14976     {
14977         \tl_use:N
14978         \g_@@_diagrams_mmdc_command_tl
14979     }
14980 }
14981 \cs_set:Nn
14982 \@@_diagrams_infostrings_current:n
14983 {
14984     \tl_set:Nx
14985     \l_tmpb_tl
14986     { \mmdcCommand }
14987     \tl_put_right:Nn
14988     \l_tmpb_tl
14989     { ---pdfFit~-i~#1~-o~#1.pdf }
14990     \@@_diagrams_render_diagram:nnVn
14991     { #1 }
14992     { #1.pdf }
14993     \l_tmpb_tl
14994     { Mermaid~image }
14995 }
14996 \@@_tl_set_from_cs:NNn
14997 \l_tmpa_tl
14998 \@@_diagrams_infostrings_current:n
14999 { 1 }
15000 \prop_gput:NnV
15001 \g_markdown_diagrams_infostrings_prop
15002 { mermaid }
15003 \l_tmpa_tl

```

Typeset fenced code with infostring `plantuml` using the command `plantuml` from the package PlantUML.

```

15004 \regex_const:Nn
15005 \c_@@_diagrams_filename_suffix_regex

```

```

15006      { \.[^.]*$ }
15007      \cs_set:Nn
15008        \@@_diagrams_infostrings_current:n
15009        {

```

Use the output format provided by the user.

```

15010      \tl_set:Nn
15011        \l_tmpa_tl
15012        { #1 }
15013      \regex_replace_once:NxN
15014        \c_@@_diagrams_filename_suffix_regex
15015        {
15016          .
15017          \tl_use:N
15018            \l_@@_diagrams_format_tl
15019        }
15020      \l_tmpa_tl
15021      \tl_set:Nn
15022        \l_tmpb_tl
15023        { plantuml~-t }
15024      \tl_put_right:NV
15025        \l_tmpb_tl
15026        \l__markdown_diagrams_format_tl
15027      \tl_put_right:Nn
15028        \l_tmpb_tl
15029        { ~#1 }

```

For the SVG format, use Inkscape to convert the resulting image to PDF.

```

15030      \str_if_eq:VnT
15031        \l_@@_diagrams_format_tl
15032        { svg }
15033      {
15034        \tl_put_right:Nn
15035          \l_tmpb_tl
15036          { ;~inkscape~ }
15037        \tl_put_right:NV
15038          \l_tmpb_tl
15039          \l_tmpa_tl
15040        \tl_put_right:Nn
15041          \l_tmpb_tl
15042          { ---export-area-drawing---export-dpi=300~-o~ }
15043        \tl_set:Nn
15044          \l_tmpa_tl
15045          { #1 }
15046        \regex_replace_once:NnN
15047          \c_@@_diagrams_filename_suffix_regex
15048          { .pdf }
15049          \l_tmpa_tl

```

```

15050         \tl_put_right:NV
15051         \l_tmpb_tl
15052         \l_tmpa_tl
15053     }
15054     \@@_diagrams_render_diagram:nVVn
15055     { #1 }
15056     \l_tmpa_tl
15057     \l_tmpb_tl
15058     { PlantUML~image }
15059 }
15060 \cs_generate_variant:Nn
15061   \@@_diagrams_render_diagram:nnnn
15062   { nVVn }
15063 \cs_generate_variant:Nn
15064   \regex_replace_once:NnN
15065   { NxN }
15066 \@@_tl_set_from_cs:NNn
15067   \l_tmpa_tl
15068   \@@_diagrams_infostrings_current:n
15069   { 1 }
15070 \prop_gput:NnV
15071   \g_@@_plain_tex_built_in_themes_prop
15072   { plantuml }
15073   \l_tmpa_tl
15074 }

```

We locally change the category code of percent signs, so that we can use them in the shell code:

```

15075 \group_begin:
15076 \char_set_catcode_other:N \%

```

The [witiko/graphicx/http](https://github.com/witiko/latex-plantuml) theme stores the previous definition of the image token renderer prototype:

```

15077 \prop_gput:Nnn
15078   \g_@@_plain_tex_built_in_themes_prop
15079   { witiko / graphicx / http }
15080   {
15081     \cs_set_eq:NN
15082       \@@_graphicx_http_previous_definition:nnnn
15083       \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.

```

15084   \int_new:N
15085     \g_@@_graphicx_http_image_number_int
15086   \int_gset:Nn
15087     \g_@@_graphicx_http_image_number_int
15088     { 0 }

```

```

15089 \cs_new:Nn
15090 \@@_graphicx_http_filename:
15091 {
15092   \markdownOptionCacheDir
15093   / witiko_graphicx_http .
15094   \int_use:N
15095   \g_@@_graphicx_http_image_number_int
15096 }

```

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.

```

15097 \cs_new:Nn
15098 \@@_graphicx_http_download:nn
15099 {
15100   wget~--0~#2~#1~
15101   ||~curl~--location~--o~#2~#1~
15102   ||~rm~--f~#2
15103 }

```

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

```

15104 \str_new:N
15105 \l_@@_graphicx_http_filename_str
15106 \ior_new:N
15107 \g_@@_graphicx_http_filename_ior
15108 \markdownSetup {
15109   rendererPrototypes = {
15110     image = {
15111       \@@_if_option:nF
15112       { frozenCache }
15113       {

```

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:

```

15114 \sys_shell_now:e
15115 {
15116   mkdir~--p~" \markdownOptionCacheDir ";
15117   if~printf~'%s'~"#3"~|~grep~--q~--E~'^https?:';
15118   then~

```

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

```

15119 OUTPUT_PREFIX=" \markdownOptionCacheDir ";
15120 OUTPUT_BODY="$(printf~'%s'~'#3'
15121   |~md5sum~|~cut~--d'~'~--f1)";
15122 OUTPUT_SUFFIX="$(printf~'%s'~'#3'

```

```

15123             |~sed~'s/.*[.]/')";
15124             OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";

```

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

```

15125             if~!~[~-e~"$OUTPUT"~];
15126             then~
15127                 \@@_graphicx_http_download:nn
15128                 { '#3' }
15129                 { "$OUTPUT" } ;
15130                 printf~'%s'~"$OUTPUT"~
15131                 >~" \@@_graphicx_http_filename: ";
15132             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:

```

15133             else~
15134                 printf~'%s'~'#3'~
15135                 >~" \@@_graphicx_http_filename: ";
15136             fi
15137         }
15138     }

```

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

```

15139         \ior_open:Ne
15140         \g_@@_graphicx_http_filename_ior
15141         { \@@_graphicx_http_filename: }
15142         \ior_str_get:NN
15143         \g_@@_graphicx_http_filename_ior
15144         \l_@@_graphicx_http_filename_str
15145         \ior_close:N
15146         \g_@@_graphicx_http_filename_ior
15147         \@@_graphicx_http_previous_definition:nnVn
15148         { #1 }
15149         { #2 }
15150         \l_@@_graphicx_http_filename_str
15151         { #4 }
15152         \int_gincr:N
15153         \g_@@_graphicx_http_image_number_int
15154     }
15155 }
15156 }
15157 \cs_generate_variant:Nn
15158 \ior_open:Nn
15159 { Ne }
15160 \cs_generate_variant:Nn
15161 \@@_graphicx_http_previous_definition:nnnn
15162 { nnVn }

```

```

15163 }
15164 \group_end:

```

The `witiko/tilde` theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```

15165 \prop_gput:Nnn
15166   \g_@@_plain_tex_built_in_themes_prop
15167   { witiko / tilde }
15168   {
15169     \markdownSetup {
15170       rendererPrototypes = {
15171         tilde = {~},
15172       },
15173     }
15174   }

```

The themes `witiko/example/foo` and `witiko/example/bar` are supposed to be used in code examples. They don't do anything.

```

15175 \clist_map_inline:nn
15176   { foo, bar }
15177   {
15178     \prop_gput:Nnn
15179     \g_@@_plain_tex_built_in_themes_prop
15180     { witiko / example / #1 }
15181     {
15182       \markdownWarning
15183       {
15184         The~theme~witiko/example/#1~is~supposed~to~be~used~in~code~
15185         examples.~Using~it~in~actual~code~has~no~effect,~except~
15186         this~warning~message,~and~is~usually~a~mistake.
15187       }
15188     }
15189   }
15190 \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.

```

15191 \def\markdownRendererInterblockSeparatorPrototype{\par}%
15192 \def\markdownRendererParagraphSeparatorPrototype{%
15193   \markdownRendererInterblockSeparator}%
15194 \def\markdownRendererHardLineBreakPrototype{\hfil\break}%
15195 \def\markdownRendererSoftLineBreakPrototype{ }%
15196 \let\markdownRendererEllipsisPrototype\dots
15197 \def\markdownRendererNbspPrototype{~}%

```

```

15198 \def\markdownRendererLeftBracePrototype{\char`{}\}%
15199 \def\markdownRendererRightBracePrototype{\char`\}\}%
15200 \def\markdownRendererDollarSignPrototype{\char`$}\}%
15201 \def\markdownRendererPercentSignPrototype{\char`\%}\}%
15202 \def\markdownRendererAmpersandPrototype{\&}\}%
15203 \def\markdownRendererUnderscorePrototype{\char`_}\}%
15204 \def\markdownRendererHashPrototype{\char`\#}\}%
15205 \def\markdownRendererCircumflexPrototype{\char`^}\}%
15206 \def\markdownRendererBackslashPrototype{\char`\}\}%
15207 \def\markdownRendererTildePrototype{\char`~}\}%
15208 \def\markdownRendererPipePrototype{\|}\}%
15209 \def\markdownRendererCodeSpanPrototype#1{\tt#1}\}%
15210 \def\markdownRendererLinkPrototype#1#2#3#4{#2}\}%
15211 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
15212   \markdownInput{#3}}\}%
15213 \def\markdownRendererContentBlockOnlineImagePrototype{%
15214   \markdownRendererImage}\}%
15215 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
15216   \markdownRendererInputFencedCode{#3}{#2}{#2}}\}%
15217 \def\markdownRendererImagePrototype#1#2#3#4{#2}\}%
15218 \def\markdownRendererUlBeginPrototype{}\}%
15219 \def\markdownRendererUlBeginTightPrototype{}\}%
15220 \def\markdownRendererUlItemPrototype{}\}%
15221 \def\markdownRendererUlItemEndPrototype{}\}%
15222 \def\markdownRendererUlEndPrototype{}\}%
15223 \def\markdownRendererUlEndTightPrototype{}\}%
15224 \def\markdownRendererOlBeginPrototype{}\}%
15225 \def\markdownRendererOlBeginTightPrototype{}\}%
15226 \def\markdownRendererFancyOlBeginPrototype#1#2{%
15227   \markdownRendererOlBegin}\}%
15228 \def\markdownRendererFancyOlBeginTightPrototype#1#2{%
15229   \markdownRendererOlBeginTight}\}%
15230 \def\markdownRendererOlItemPrototype{}\}%
15231 \def\markdownRendererOlItemWithNumberPrototype#1{}\}%
15232 \def\markdownRendererOlItemEndPrototype{}\}%
15233 \def\markdownRendererFancyOlItemPrototype{\markdownRendererOlItem}\}%
15234 \def\markdownRendererFancyOlItemWithNumberPrototype{%
15235   \markdownRendererOlItemWithNumber}\}%
15236 \def\markdownRendererFancyOlItemEndPrototype{}\}%
15237 \def\markdownRendererOlEndPrototype{}\}%
15238 \def\markdownRendererOlEndTightPrototype{}\}%
15239 \def\markdownRendererFancyOlEndPrototype{\markdownRendererOlEnd}\}%
15240 \def\markdownRendererFancyOlEndTightPrototype{%
15241   \markdownRendererOlEndTight}\}%
15242 \def\markdownRendererDlBeginPrototype{}\}%
15243 \def\markdownRendererDlBeginTightPrototype{}\}%
15244 \def\markdownRendererDlItemPrototype#1{#1}\}%

```

```

15245 \def\markdownRendererDlItemEndPrototype{}%
15246 \def\markdownRendererDlDefinitionBeginPrototype{}%
15247 \def\markdownRendererDlDefinitionEndPrototype{\par}%
15248 \def\markdownRendererDlEndPrototype{}%
15249 \def\markdownRendererDlEndTightPrototype{}%
15250 \def\markdownRendererEmphasisPrototype#1{\it#1}%
15251 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
15252 \def\markdownRendererBlockQuoteBeginPrototype{\begingroup\it}%
15253 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
15254 \def\markdownRendererLineBlockBeginPrototype{\begingroup\parindent=Opt}%
15255 \def\markdownRendererLineBlockEndPrototype{\endgroup}%
15256 \def\markdownRendererInputVerbatimPrototype#1{%
15257   \par{\tt\input#1\relax{}}\par}%
15258 \def\markdownRendererInputFencedCodePrototype#1#2#3{%
15259   \markdownRendererInputVerbatim{#1}}%
15260 \def\markdownRendererHeadingOnePrototype#1{#1}%
15261 \def\markdownRendererHeadingTwoPrototype#1{#1}%
15262 \def\markdownRendererHeadingThreePrototype#1{#1}%
15263 \def\markdownRendererHeadingFourPrototype#1{#1}%
15264 \def\markdownRendererHeadingFivePrototype#1{#1}%
15265 \def\markdownRendererHeadingSixPrototype#1{#1}%
15266 \def\markdownRendererThematicBreakPrototype{}%
15267 \def\markdownRendererNotePrototype#1{#1}%
15268 \def\markdownRendererCitePrototype#1{}%
15269 \def\markdownRendererTextCitePrototype#1{}%
15270 \def\markdownRendererTickedBoxPrototype{[X]}%
15271 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
15272 \def\markdownRendererUntickedBoxPrototype{[ ]}%
15273 \def\markdownRendererStrikeThroughPrototype#1{#1}%
15274 \def\markdownRendererSuperscriptPrototype#1{#1}%
15275 \def\markdownRendererSubscriptPrototype#1{#1}%
15276 \def\markdownRendererDisplayMathPrototype#1{$$#1$$}%
15277 \def\markdownRendererInlineMathPrototype#1{$#1$}%
15278 \ExplSyntaxOn
15279 \cs_gset:Npn
15280   \markdownRendererHeaderAttributeContextBeginPrototype
15281   {
15282     \group_begin:
15283     \color_group_begin:
15284   }
15285 \cs_gset:Npn
15286   \markdownRendererHeaderAttributeContextEndPrototype
15287   {
15288     \color_group_end:
15289     \group_end:
15290   }
15291 \cs_gset_eq:NN

```



```

15292 \markdownRendererBracketedSpanAttributeContextBeginPrototype
15293 \markdownRendererHeaderAttributeContextBeginPrototype
15294 \cs_gset_eq:NN
15295 \markdownRendererBracketedSpanAttributeContextEndPrototype
15296 \markdownRendererHeaderAttributeContextEndPrototype
15297 \cs_gset_eq:NN
15298 \markdownRendererFencedDivAttributeContextBeginPrototype
15299 \markdownRendererHeaderAttributeContextBeginPrototype
15300 \cs_gset_eq:NN
15301 \markdownRendererFencedDivAttributeContextEndPrototype
15302 \markdownRendererHeaderAttributeContextEndPrototype
15303 \cs_gset_eq:NN
15304 \markdownRendererFencedCodeAttributeContextBeginPrototype
15305 \markdownRendererHeaderAttributeContextBeginPrototype
15306 \cs_gset_eq:NN
15307 \markdownRendererFencedCodeAttributeContextEndPrototype
15308 \markdownRendererHeaderAttributeContextEndPrototype
15309 \cs_gset:Npn
15310 \markdownRendererReplacementCharacterPrototype
15311 { \codepoint_str_generate:n { fffd } }
15312 \ExplSyntaxOff
15313 \def\markdownRendererSectionBeginPrototype{}%
15314 \def\markdownRendererSectionEndPrototype{}%
15315 \ExplSyntaxOn
15316 \cs_gset:Npn
15317 \markdownRendererWarningPrototype
15318 #1#2#3#4
15319 {
15320   \tl_set:Nn
15321     \l_tmpa_tl
15322     { #2 }
15323   \tl_if_empty:nF
15324     { #4 }
15325     {
15326       \tl_put_right:Nn
15327         \l_tmpa_tl
15328         { \iow_newline: #4 }
15329     }
15330   \exp_args:NV
15331     \markdownWarning
15332     \l_tmpa_tl
15333 }
15334 \ExplSyntaxOff
15335 \def\markdownRendererErrorPrototype#1#2#3#4{%
15336   \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.

```

15337 \ExplSyntaxOn
15338 \cs_new:Nn
15339   \@@_plain_tex_default_input_raw_inline:nn
15340   {
15341     \str_case:nn
15342       { #2 }
15343       {
15344         { md } { \markdownInput{#1} }
15345         { tex } { \markdownEscape{#1} \unskip }
15346       }
15347   }
15348 \cs_new:Nn
15349   \@@_plain_tex_default_input_raw_block:nn
15350   {
15351     \str_case:nn
15352       { #2 }
15353       {
15354         { md } { \markdownInput{#1} }
15355         { tex } { \markdownEscape{#1} }
15356       }
15357   }
15358 \cs_gset:Npn
15359   \markdownRendererInputRawInlinePrototype#1#2
15360   {
15361     \@@_plain_tex_default_input_raw_inline:nn
15362       { #1 }
15363       { #2 }
15364   }
15365 \cs_gset:Npn
15366   \markdownRendererInputRawBlockPrototype#1#2
15367   {
15368     \@@_plain_tex_default_input_raw_block:nn
15369       { #1 }
15370       { #2 }
15371   }
15372 \ExplSyntaxOff

```

### 3.2.3.2 Simple yaml Metadata Renderer Prototypes

In this section, we implement the simple high-level interface for processing simple YAML metadata using the key-value `markdown/jekyllData`. See also Section 2.2.6.1.

To keep track of the current type of structure we inhabit when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_datatypes_seq` stack.

At every step of the traversal, the stack will contain one of the following constants at any position  $p$ :

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

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

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

```
15373 \ExplSyntaxOn
15374 \seq_new:N \g_@@_jekyll_data_datatypes_seq
15375 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
15376 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
15377 \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.

```
15378 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
15379 \cs_new:Nn \@@_jekyll_data_push_address_segment:n
15380 {
15381   \seq_if_empty:NF
15382     \g_@@_jekyll_data_datatypes_seq
15383     {
15384       \seq_get_right:NN
15385       \g_@@_jekyll_data_datatypes_seq
15386       \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.

```
15387   \str_if_eq:NNTF
15388     \l_tmpa_tl
15389     \c_@@_jekyll_data_sequence_tl
15390     {
15391       \seq_put_right:Nn
15392       \g_@@_jekyll_data_wildcard_absolute_address_seq
15393       { * }
15394     }
15395   {
15396     \seq_put_right:Nn
15397     \g_@@_jekyll_data_wildcard_absolute_address_seq
```

```

15398         { #1 }
15399     }
15400 }
15401 }

```

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.

```

15402 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
15403 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
15404 \cs_new:Nn \@@_jekyll_data_concatenate_address:NN
15405 {
15406     \seq_pop_left:NN #1 \l_tmpa_tl
15407     \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
15408     \seq_put_left:NV #1 \l_tmpa_tl
15409 }
15410 \cs_new:Nn \@@_jekyll_data_update_address_tls:
15411 {
15412     \@@_jekyll_data_concatenate_address:NN
15413     \g_@@_jekyll_data_wildcard_absolute_address_seq
15414     \g_@@_jekyll_data_wildcard_absolute_address_tl
15415     \seq_get_right:NN
15416     \g_@@_jekyll_data_wildcard_absolute_address_seq
15417     \g_@@_jekyll_data_wildcard_relative_address_tl
15418 }

```

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.

```

15419 \cs_new:Nn \@@_jekyll_data_push:nN
15420 {
15421   \@@_jekyll_data_push_address_segment:n
15422   { #1 }
15423   \seq_put_right:NV
15424   \g_@@_jekyll_data_datatypes_seq
15425   #2
15426   \@@_jekyll_data_update_address_tls:
15427 }
15428 \cs_new:Nn \@@_jekyll_data_pop:
15429 {
15430   \seq_pop_right:NN
15431   \g_@@_jekyll_data_wildcard_absolute_address_seq
15432   \l_tmpa_tl
15433   \seq_pop_right:NN
15434   \g_@@_jekyll_data_datatypes_seq
15435   \l_tmpa_tl
15436   \@@_jekyll_data_update_address_tls:
15437 }

```

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.

```

15438 \cs_new:Nn \@@_jekyll_data_set_keyval_known:nn
15439 {
15440   \keys_set_known:nn
15441   { markdown/jekyllData }
15442   { { #1 } = { #2 } }
15443 }
15444 \cs_generate_variant:Nn
15445   \@@_jekyll_data_set_keyval_known:nn
15446   { Vn }
15447 \cs_new:Nn \@@_jekyll_data_set_keyvals_known:nn
15448 {
15449   \@@_jekyll_data_push:nN
15450   { #1 }
15451   \c_@@_jekyll_data_scalar_tl
15452   \@@_jekyll_data_set_keyval_known:Vn
15453   \g_@@_jekyll_data_wildcard_absolute_address_tl
15454   { #2 }
15455   \@@_jekyll_data_set_keyval_known:Vn
15456   \g_@@_jekyll_data_wildcard_relative_address_tl
15457   { #2 }
15458   \@@_jekyll_data_pop:
15459 }

```

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

```

15460 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
15461   \@@_jekyll_data_push:nN
15462     { #1 }
15463     \c_@@_jekyll_data_sequence_tl
15464 }
15465 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
15466   \@@_jekyll_data_push:nN
15467     { #1 }
15468     \c_@@_jekyll_data_mapping_tl
15469 }
15470 \def\markdownRendererJekyllDataSequenceEndPrototype{
15471   \@@_jekyll_data_pop:
15472 }
15473 \def\markdownRendererJekyllDataMappingEndPrototype{
15474   \@@_jekyll_data_pop:
15475 }
15476 \def\markdownRendererJekyllDataBooleanPrototype#1#2{
15477   \@@_jekyll_data_set_keyvals_known:nn
15478     { #1 }
15479     { #2 }
15480 }
15481 \def\markdownRendererJekyllDataEmptyPrototype#1{}
15482 \def\markdownRendererJekyllDataNumberPrototype#1#2{
15483   \@@_jekyll_data_set_keyvals_known:nn
15484     { #1 }
15485     { #2 }
15486 }

```

We will process all string scalar values assuming that they may contain markdown markup and are intended for typesetting.

```

15487 \def\markdownRendererJekyllDataProgrammaticStringPrototype#1#2{}
15488 \def\markdownRendererJekyllDataTypographicStringPrototype#1#2{
15489   \@@_jekyll_data_set_keyvals_known:nn
15490     { #1 }
15491     { #2 }
15492 }
15493 \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.

```

15494 \ExplSyntaxOn
15495 \@@_with_various_cases:nn

```

```

15496 { jekyllDataKeyValue }
15497 {
15498   \keys_define:nn
15499     { markdown/options }
15500     {
15501       #1 .code:n = {
15502         \@@_route_jekyll_data_to_key_values:n
15503         { ##1 }
15504       },

```

When no  $\langle module \rangle$  has been provided, assume that the YAML metadata specify absolute paths to key-values.

```

15505       #1 .default:n = { },
15506     }
15507   }
15508 \seq_new:N
15509   \l_@@_jekyll_data_current_position_seq
15510 \tl_new:N
15511   \l_@@_jekyll_data_current_position_tl
15512 \cs_new:Nn
15513   \@@_route_jekyll_data_to_key_values:n
15514   {
15515     \markdownSetup
15516     {
15517       renderers = {
15518         jekyllData(Sequence|Mapping)Begin = {
15519           \bool_lazy_and:nnTF
15520             {
15521               \seq_if_empty_p:N
15522                 \l_@@_jekyll_data_current_position_seq
15523             }
15524             {
15525               \str_if_eq_p:nn
15526                 { ##1 }
15527                 { null }
15528             }
15529             {
15530               \tl_if_empty:nF
15531                 { #1 }
15532                 {
15533                   \seq_put_right:Nn
15534                     \l_@@_jekyll_data_current_position_seq
15535                     { #1 }
15536                 }
15537             }
15538           {
15539             \seq_put_right:Nn

```

```

15540         \l_@@_jekyll_data_current_position_seq
15541         { ##1 }
15542     }
15543 },
15544 jekyllData(Sequence|Mapping)End = {
15545     \seq_pop_right:NN
15546     \l_@@_jekyll_data_current_position_seq
15547     \l_tmpa_tl
15548 },

```

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`.

```

15549     jekyllDataBoolean = {
15550         \tl_set:Nx
15551         \l_@@_jekyll_data_current_position_tl
15552         {
15553             \seq_use:Nn
15554             \l_@@_jekyll_data_current_position_seq
15555             { / }
15556         }
15557         \keys_if_exist:VnTF
15558         \l_@@_jekyll_data_current_position_tl
15559         { ##1 / boolean }
15560         {
15561             \@@_keys_set:xn
15562             {
15563                 \tl_use:N
15564                 \l_@@_jekyll_data_current_position_tl
15565                 / ##1 / boolean
15566             }
15567             { ##2 }
15568         }
15569         {
15570             \@@_keys_set:xn
15571             {
15572                 \tl_use:N
15573                 \l_@@_jekyll_data_current_position_tl
15574                 / ##1
15575             }
15576             { ##2 }
15577         }
15578     },
15579     jekyllDataNumber = {
15580         \tl_set:Nx
15581         \l_@@_jekyll_data_current_position_tl

```



```

15582         {
15583             \seq_use:Nn
15584             \l_@@_jekyll_data_current_position_seq
15585             { / }
15586         }
15587     \keys_if_exist:VnTF
15588     \l_@@_jekyll_data_current_position_tl
15589     { ##1 / number }
15590     {
15591         \@@_keys_set:xn
15592         {
15593             \tl_use:N
15594             \l_@@_jekyll_data_current_position_tl
15595             / ##1 / number
15596         }
15597         { ##2 }
15598     }
15599     {
15600         \@@_keys_set:xn
15601         {
15602             \tl_use:N
15603             \l_@@_jekyll_data_current_position_tl
15604             / ##1
15605         }
15606         { ##2 }
15607     }
15608 },

```

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`.

```

15609 jekyllDataEmpty = {
15610     \tl_set:Nx
15611     \l_@@_jekyll_data_current_position_tl
15612     {
15613         \seq_use:Nn
15614         \l_@@_jekyll_data_current_position_seq
15615         { / }
15616     }
15617     \keys_if_exist:VnTF
15618     \l_@@_jekyll_data_current_position_tl
15619     { ##1 / empty }
15620     {
15621         \keys_set:xn
15622         {
15623             \tl_use:N
15624             \l_@@_jekyll_data_current_position_tl

```

```

15625         / ##1
15626     }
15627     { empty }
15628 }
15629 {
15630     \keys_set:Vn
15631     \l_@@_jekyll_data_current_position_tl
15632     { ##1 }
15633 }
15634 },

```

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.

```

15635     jekyllDataTypographicString = {
15636         \tl_set:Nx
15637         \l_@@_jekyll_data_current_position_tl
15638         {
15639             \seq_use:Nn
15640             \l_@@_jekyll_data_current_position_seq
15641             { / }
15642         }
15643     \keys_if_exist:VnTF
15644     \l_@@_jekyll_data_current_position_tl
15645     { ##1 / typographicString }
15646     {
15647         \@@_keys_set:xn
15648         {
15649             \tl_use:N
15650             \l_@@_jekyll_data_current_position_tl
15651             / ##1 / typographicString
15652         }
15653         { ##2 }
15654     }
15655     {
15656         \keys_if_exist:VnTF
15657         \l_@@_jekyll_data_current_position_tl
15658         { ##1 / programmaticString }
15659         {
15660             \@@_keys_set_known:xn

```

```

15661         {
15662             \tl_use:N
15663             \l_@@_jekyll_data_current_position_tl
15664             / ##1
15665         }
15666         { ##2 }
15667     }
15668     {
15669         \@@_keys_set:xn
15670         {
15671             \tl_use:N
15672             \l_@@_jekyll_data_current_position_tl
15673             / ##1
15674         }
15675         { ##2 }
15676     }
15677 }
15678 },
15679 jekyllDataProgrammaticString = {
15680     \tl_set:Nx
15681     \l_@@_jekyll_data_current_position_tl
15682     {
15683         \seq_use:Nn
15684         \l_@@_jekyll_data_current_position_seq
15685         { / }
15686     }
15687     \keys_if_exist:VnT
15688     \l_@@_jekyll_data_current_position_tl
15689     { ##1 / programmaticString }
15690     {
15691         \@@_keys_set:xn
15692         {
15693             \tl_use:N
15694             \l_@@_jekyll_data_current_position_tl
15695             / ##1 / programmaticString
15696         }
15697         { ##2 }
15698     }
15699 },
15700 },
15701 }
15702 }
15703 \cs_new:Nn
15704 \@@_keys_set:nn
15705 {
15706     \keys_set:nn
15707     { }

```

```

15708      { { #1 } = { #2 } }
15709    }
15710    \cs_new:Nn
15711      \@@_keys_set_known:nn
15712      {
15713        \keys_set_known:nn
15714          { }
15715          { { #1 } = { #2 } }
15716      }
15717    \cs_generate_variant:Nn
15718      \@@_keys_set:nn
15719      { xn }
15720    \cs_generate_variant:Nn
15721      \@@_keys_set_known:nn
15722      { xn }
15723    \cs_generate_variant:Nn
15724      \keys_set:nn
15725      { xn, Vn }
15726    \prg_generate_conditional_variant:Nnn
15727      \keys_if_exist:nn
15728      { Vn }
15729      { T, TF }
15730    \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).

```

15731    \ExplSyntaxOn
15732    \str_if_eq:VVT
15733      \c_@@_top_layer_tl
15734      \c_@@_option_layer_plain_tex_tl
15735      {
15736        \use:c
15737          { ExplSyntaxOff }
15738        \@@_if_option:nF
15739          { noDefaults }
15740        {
15741          \@@_if_option:nTF
15742            { experimental }
15743            {
15744              \@@_setup:n
15745                { theme = witiko/markdown/defaults@experimental }
15746            }
15747          {
15748            \@@_setup:n
15749              { theme = witiko/markdown/defaults }
15750          }
15751        }
15752      }

```

```

15751     }
15752     \use:c
15753     { ExplSyntaxOn }
15754   }
15755 \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).

```

15756 \ExplSyntaxOn
15757 \tl_new:N \g_@@_formatted_lua_options_tl
15758 \cs_new:Nn \@@_format_lua_options:
15759 {
15760   \tl_gclear:N
15761   \g_@@_formatted_lua_options_tl
15762   \seq_map_function:NN
15763   \g_@@_lua_options_seq
15764   \@@_format_lua_option:n
15765 }
15766 \cs_new:Nn \@@_format_lua_option:n
15767 {
15768   \@@_typecheck_option:n
15769   { #1 }
15770   \@@_get_option_type:nN
15771   { #1 }
15772   \l_tmpa_tl
15773   \bool_case_true:nF
15774   {
15775     {
15776       \str_if_eq_p:VV
15777       \l_tmpa_tl
15778       \c_@@_option_type_boolean_tl ||
15779       \str_if_eq_p:VV
15780       \l_tmpa_tl
15781       \c_@@_option_type_number_tl ||
15782       \str_if_eq_p:VV
15783       \l_tmpa_tl
15784       \c_@@_option_type_counter_tl
15785     }
15786     {
15787       \@@_get_option_value:nN
15788       { #1 }
15789       \l_tmpa_tl
15790       \tl_gput_right:Nx

```

```

15791         \g_@@_formatted_lua_options_tl
15792         { #1~::~ \l_tmpa_tl ,~ }
15793     }
15794 {
15795     \str_if_eq_p:VV
15796     \l_tmpa_tl
15797     \c_@@_option_type_clist_tl
15798 }
15799 {
15800     \@@_get_option_value:nN
15801     { #1 }
15802     \l_tmpa_tl
15803     \tl_gput_right:Nx
15804     \g_@@_formatted_lua_options_tl
15805     { #1~::~\c_left_brace_str }
15806     \clist_map_inline:Vn
15807     \l_tmpa_tl
15808     {
15809         \@@_lua_escape:xN
15810         { ##1 }
15811         \l_tmpb_tl
15812         \tl_gput_right:Nn
15813         \g_@@_formatted_lua_options_tl
15814         { " }
15815         \tl_gput_right:NV
15816         \g_@@_formatted_lua_options_tl
15817         \l_tmpb_tl
15818         \tl_gput_right:Nn
15819         \g_@@_formatted_lua_options_tl
15820         { " ,~ }
15821     }
15822     \tl_gput_right:Nx
15823     \g_@@_formatted_lua_options_tl
15824     { \c_right_brace_str ,~ }
15825 }
15826 }
15827 {
15828     \@@_get_option_value:nN
15829     { #1 }
15830     \l_tmpa_tl
15831     \@@_lua_escape:xN
15832     { \l_tmpa_tl }
15833     \l_tmpb_tl
15834     \tl_gput_right:Nn
15835     \g_@@_formatted_lua_options_tl
15836     { #1~::~ " }
15837     \tl_gput_right:NV

```

```

15838         \g_@@_formatted_lua_options_tl
15839         \l_tmpb_tl
15840         \tl_gput_right:Nn
15841         \g_@@_formatted_lua_options_tl
15842         { " ,~ }
15843     }
15844 }
15845 \cs_generate_variant:Nn
15846 \clist_map_inline:nn
15847 { Vn }
15848 \let
15849 \markdownPrepareLuaOptions
15850 \@@_format_lua_options:
15851 \def
15852 \markdownLuaOptions
15853 {
15854     {
15855         \g_@@_formatted_lua_options_tl
15856     }
15857 }
15858 \sys_if_engine luatex:TF
15859 {
15860     \cs_new:Nn
15861     \@@_lua_escape:nN
15862     {
15863         \tl_set:Nx
15864         #2
15865         {
15866             \lua_escape:n
15867             { #1 }
15868         }
15869     }
15870 }
15871 {
15872     \regex_const:Nn
15873     \c_@@_lua_escape_regex
15874     { [\\"] }
15875     \cs_new:Nn
15876     \@@_lua_escape:nN
15877     {
15878         \tl_set:Nn
15879         #2
15880         { #1 }
15881         \regex_replace_all:NnN
15882         \c_@@_lua_escape_regex
15883         { \u { c_backslash_str } \0 }
15884         #2

```

```

15885     }
15886   }
15887   \cs_generate_variant:Nn
15888     \@@_lua_escape:nN
15889     { 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.

```

15890 \tl_new:N
15891   \markdownInputFilename
15892 \cs_new:Npn
15893   \markdownPrepareInputFilename
15894   #1
15895   {
15896     \@@_lua_escape:xN
15897     { #1 }
15898     \markdownInputFilename
15899     \tl_gset:Nx
15900     \markdownInputFilename
15901     { " \markdownInputFilename " }
15902   }

```

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.

```

15903 \cs_new:Npn
15904   \markdownPrepare
15905   {

```

First, ensure that the `cacheDir` directory exists.

```

15906     local~lfs = require("lfs")
15907     local~options = \markdownLuaOptions
15908     if~not~lfs.isdir(options.cacheDir) then~
15909       assert(lfs.mkdir(options.cacheDir))
15910     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.

```

15911     local~md = require("markdown")
15912     local~convert = md.new(options)
15913   }

```

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.

```

15914 \cs_new:Npn
15915   \markdownConvert

```



```

15916 {
15917     local~filename = \markdownInputFilename
15918     local~file = assert(io.open(filename, "r"),
15919         [[Could~not~open~file~]] .. filename .. [[~for~reading]])
15920     local~input = assert(file:read("*a"))
15921     assert(file:close())
15922     print(convert(input))
15923 }
15924 \ExplSyntaxOff

```

The `\markdownCleanup` macro contains the Lua code that is executed after any conversion from markdown to plain T<sub>E</sub>X.

```

15925 \def\markdownCleanup{%

```

Remove the `options.cacheDir` directory if it is empty.

```

15926     if options.cacheDir then
15927         lfs.rmdir(options.cacheDir)
15928     end
15929 }%

```

### 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.

```

15930 \csname newread\endcsname\markdownInputFileStream
15931 \csname newwrite\endcsname\markdownOutputFileStream

```

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

```

15932 \begingroup
15933     \catcode`\^^I=12%
15934     \gdef\markdownReadAndConvertTab{^^I}%
15935 \endgroup

```

The `\markdownReadAndConvert` macro is largely a rewrite of the L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> `\filecontents` macro to plain T<sub>E</sub>X.

```

15936 \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.

```

15937     \catcode`\^^M=13%
15938     \catcode`\^^I=13%
15939     \catcode`|=0%
15940     \catcode`\=12%
15941     |catcode`@=14%

```

```

15942 |catcode`|=12@
15943 |gdef|markdownReadAndConvert#1#2{@
15944 |begingroup@

```

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

```

15945 |markdownIfOption{frozenCache}{-}{@
15946 |immediate|openout|markdownOutputFileStream@
15947 |markdownOptionInputTempFileName|relax@
15948 |markdownInfo{@
15949 Buffering block-level markdown input into the temporary @
15950 input file "|markdownOptionInputTempFileName" and scanning @
15951 for the closing token sequence "#1"}@
15952 }@

```

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.

```

15953 |def|do##1{|catcode`##1=12}|dospecials@
15954 |catcode`|=12@
15955 |markdownMakeOther@

```

The `\markdownReadAndConvertStripPercentSigns` macro will process the individual lines of output, stripping away leading percent signs (%) when `stripPercentSigns` is enabled. Notice the use of the comments (@) to ensure that the entire macro is at a single line and therefore no (active) newline symbols (`^^M`) are produced.

```

15956 |def|markdownReadAndConvertStripPercentSign##1{@
15957 |markdownIfOption{stripPercentSigns}{@
15958 |if##1%@
15959 |expandafter|expandafter|expandafter@
15960 |markdownReadAndConvertProcessLine@
15961 |else@
15962 |expandafter|expandafter|expandafter@
15963 |markdownReadAndConvertProcessLine@
15964 |expandafter|expandafter|expandafter##1@
15965 |fi@
15966 }{@
15967 |expandafter@
15968 |markdownReadAndConvertProcessLine@
15969 |expandafter##1@
15970 }@
15971 }@

```

The `\markdownReadAndConvertProcessLine` macro will process the individual lines of output. Notice the use of the comments (@) to ensure that the entire macro is at a single line and therefore no (active) newline symbols (`^^M`) are produced.

```

15972 |def|markdownReadAndConvertProcessLine##1##2##3|relax{@

```

If we are not reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, store the line in the `inputTempFileName` file. If we are reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, gobble the line.

```

15973      |ifx|relax##3|relax@
15974      |markdownIfOption{frozenCache}{-}{@
15975      |immediate|write|markdownOutputStream{##1}@
15976      }@
15977      |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.

```

15978      |def^^M{@
15979      |markdownInfo{The ending token sequence was found}@
15980      |markdownIfOption{frozenCache}{-}{@
15981      |immediate|closeout|markdownOutputStream@
15982      }@
15983      |endgroup@
15984      |markdownInput{@
15985      |markdownOptionOutputDir@
15986      /|markdownOptionInputTempFileName@
15987      }@
15988      #2}@
15989      |fi@

```

Repeat with the next line.

```

15990      ^^M}@

```

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

```

15991      |catcode`|^I=13@
15992      |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.

```

15993      |catcode`|^M=13@
15994      |def^^M##1^^M{@
15995      |def^^M###1^^M{@
15996      |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
15997      ^^M}@
15998      ^^M}@

```

Reset the character categories back to the former state.

```

15999 |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.

```

16000 \ExplSyntaxOn
16001 \cs_new:Npn
16002   \markdownLuaExecute
16003   #1
16004   {
16005     \int_compare:nNnT
16006       { \g_luabridge_method_int }
16007       =
16008       { \c_luabridge_method_shell_int }
16009       {
16010         \sys_if_shell_unrestricted:F
16011         {
16012           \sys_if_shell:TF
16013           {
16014             \msg_error:nn
16015               { markdown }
16016               { restricted-shell-access }
16017           }
16018           {
16019             \msg_error:nn
16020               { markdown }
16021               { disabled-shell-access }
16022           }
16023         }
16024       }
16025     \str_gset:NV
16026       \g_luabridge_output_dirname_str
16027       \markdownOptionOutputDir
16028     \luabridge_now:e
16029     { #1 }
16030   }
16031 \cs_generate_variant:Nn
16032   \msg_new:nnnn
16033   { nnnV }
16034 \tl_set:Nn
16035   \l_tmpa_tl
16036   {
16037     You~may~need~to~run~TeX~with~the~--shell-escape~or~the~
16038     --enable-write18~flag,~or~write~shell_escape=t~in~the~
16039     texmf.cnf~file.
16040   }
16041 \msg_new:nnnV
16042   { markdown }
16043   { restricted-shell-access }
16044   { Shell~escape~is~restricted }

```

```

16045 \l_tmpa_tl
16046 \msg_new:nnnV
16047 { markdown }
16048 { disabled-shell-access }
16049 { Shell-escape~is~disabled }
16050 \l_tmpa_tl
16051 \ExplSyntaxOff

```

### 3.2.6 Buffering Inline Markdown Input

This section describes the implementation of the macro `\markinline`.

```

16052 \ExplSyntaxOn
16053 \tl_new:N
16054 \g_@@_after_markinline_tl
16055 \tl_gset:Nn
16056 \g_@@_after_markinline_tl
16057 { \unskip }
16058 \cs_new:Npn
16059 \markinline
16060 {

```

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.

```

16061 \group_begin:
16062 \cctab_select:N
16063 \c_other_cctab

```

Unless we are reading markdown documents from the frozen cache, open the file `inputTempFileName` for writing.

```

16064 \@@_if_option:nF
16065 { frozenCache }
16066 {
16067 \immediate
16068 \openout
16069 \markdownOutputFileStream
16070 \markdownOptionInputTempFileName
16071 \relax
16072 \msg_info:nne
16073 { markdown }
16074 { buffering-markinline }
16075 { \markdownOptionInputTempFileName }
16076 }

```

Peek ahead and extract the inline markdown text.

```

16077 \peek_regex_replace_once:nnF
16078 { { (.*) } }
16079 {

```

Unless we are reading markdown documents from the frozen cache, store the text in the file `inputTempFileName` and close it.

```

16080         \c { @@_if_option:nF }
16081         \cB { frozenCache \cE }
16082         \cB {
16083             \c { immediate }
16084             \c { write }
16085             \c { markdownOutputFileStream }
16086             \cB { \1 \cE }
16087             \c { immediate }
16088             \c { closeout }
16089             \c { markdownOutputFileStream }
16090         \cE }

```

Reset the category codes and `\input` the result of the conversion.

```

16091         \c { group_end: }
16092         \c { group_begin: }
16093         \c { @@_setup:n }
16094         \cB { contentLevel = inline \cE }
16095         \c { markdownInput }
16096         \cB {
16097             \c { markdownOptionOutputDir } /
16098             \c { markdownOptionInputTempFileName }
16099         \cE }
16100         \c { group_end: }
16101         \c { tl_use:N }
16102         \c { g_@@_after_markinline_tl }
16103     }
16104     {
16105         \msg_error:nn
16106         { markdown }
16107         { markinline-peek-failure }
16108         \group_end:
16109         \tl_use:N
16110         \g_@@_after_markinline_tl
16111     }
16112 }
16113 \msg_new:nnn
16114 { markdown }
16115 { buffering-markinline }
16116 { Buffering~inline~markdown~input~into~
16117   the~temporary~input~file~"#1". }
16118 \msg_new:nnnn
16119 { markdown }
16120 { markinline-peek-failure }
16121 { Use-of~\iow_char:N \\ markinline~doesn't~match~its~definition }
16122 { The~macro~should~be~followed~by~inline~

```

```

16123     markdown~text~in~curly~braces }
16124 \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.

```

16125 \ExplSyntaxOn
16126 \cs_new:Npn
16127   \markdownInput
16128   #1
16129   {
16130     \@@_if_option:nTF
16131       { frozenCache }
16132       {
16133         \markdownInputRaw
16134         { #1 }
16135       }
16136     {

```

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`.

```

16137       \tl_set:Nx
16138         \l_tmpa_tl
16139         { #1 }
16140       \file_get_full_name:VNTF
16141         \l_tmpa_tl
16142         \l_tmpb_tl
16143       {
16144         \exp_args:NV
16145           \markdownInputRaw
16146           \l_tmpb_tl
16147       }
16148       {
16149         \msg_error:nnV
16150           { markdown }
16151           { markdown-file-does-not-exist }
16152         \l_tmpa_tl
16153       }
16154     }
16155   }
16156 \msg_new:nnn
16157   { markdown }
16158   { markdown-file-does-not-exist }
16159   {

```

```

16160     Markdown~file~#1~does~not~exist
16161   }
16162   \ExplSyntaxOff
16163   \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.

```

16164     \catcode`\|=0%
16165     \catcode`\|=12%
16166     \catcode`\&=6%
16167     \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.

```

16168     |begingroup
16169     |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.

```

16170     |catcode`|#=12

```

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

```

16171     |markdownIfOption{frozenCache}{%
16172       |ifnum|markdownOptionFrozenCacheCounter=0|relax
16173         |markdownInfo{Reading frozen cache from
16174           "||markdownOptionFrozenCacheFileName"}%
16175         |input|markdownOptionFrozenCacheFileName|relax
16176       |fi
16177       |markdownInfo{Including markdown document number
16178         "|the|markdownOptionFrozenCacheCounter" from frozen cache}%
16179       |csname markdownFrozenCache%
16180         |the|markdownOptionFrozenCacheCounter|endcsname
16181       |global|advance|markdownOptionFrozenCacheCounter by 1|relax
16182     }{%
16183     |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.

```

16184     |openin|markdownInputFileStream{&1}%
16185     |closein|markdownInputFileStream
16186     |markdownPrepareLuaOptions
16187     |markdownPrepareInputFilename{&1}%
16188     |markdownLuaExecute{%

```



```

16189         |markdownPrepare
16190         |markdownConvert
16191         |markdownCleanup}%

```

If we are finalizing the frozen cache, increment `frozenCacheCounter`.

```

16192         |markdownIfOption{finalizeCache}{%
16193         |global|advance|markdownOptionFrozenCacheCounter by 1|relax}{}%
16194     }%
16195     |endgroup
16196 }%
16197 |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.

```

16198 \gdef\markdownEscape#1{%
16199     \catcode`\%=14\relax
16200     \catcode`\#=6\relax
16201     \input #1\relax
16202     \catcode`\%=12\relax
16203     \catcode`\#=12\relax
16204 }%

```

### 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.

```

16205 \def\markdownVersionSpace{ }%
16206 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
16207     \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).

```

16208 \ExplSyntaxOn
16209 \cs_gset_eq:NN
16210     \markinlinePlainTeX
16211     \markinline
16212 \cs_gset:Npn
16213     \markinline
16214     {

```

```

16215 \peek_regex_replace_once:nn
16216 { ( \[ (.*?) \] ) ? }
16217 {

```

Apply the options locally.

```

16218 \c { group_begin: }
16219 \c { @@_setup:n }
16220 \cB { \2 \cE }
16221 \c { tl_put_right:Nn }
16222 \c { g_@@_after_markinline_tl }
16223 \cB { \c { group_end: } \cE }
16224 \c { markinlinePlainTeX }
16225 }
16226 }
16227 \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 L<sup>A</sup>T<sub>E</sub>X interface (see Section 2.3.3).

```

16228 \let\markdownInputPlainTeX\markdownInput
16229 \renewcommand\markdownInput[2][]{%
16230 \begingroup
16231 \markdownSetup{#1}%
16232 \markdownInputPlainTeX{#2}%
16233 \endgroup}%
16234 \renewcommand\yamlInput[2][]{%
16235 \begingroup
16236 \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData, #1}%
16237 \markdownInputPlainTeX{#2}%
16238 \endgroup}%

```

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

```

16239 \ExplSyntaxOn
16240 \renewenvironment
16241 { markdown }
16242 {

```

In our implementation of the `markdown` L<sup>A</sup>T<sub>E</sub>X environment, we want to distinguish between the following two cases:

<code>\begin{markdown} [smartEllipses]</code>	<code>\begin{markdown}</code>
<code>% This is an optional argument ^</code>	<code>[smartEllipses]</code>
<code>% ...</code>	<code>% ^ This is link</code>
<code>\end{markdown}</code>	<code>\end{markdown}</code>

Therefore, we cannot use the built-in  $\text{\LaTeX}$  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`  $\text{\LaTeX}$  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  $\text{\TeX}$  via the `\endlinechar` plain  $\text{\TeX}$  macro and it needs to have the correct category code, so that `\markdownReadAndConvert` processes it correctly.

```
16243 \group_begin:
16244 \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 11 (letter).

```
16245 \char_set_catcode_letter:n { 35 }
```

After we have matched the opening `[` that begins the optional argument, we accept carriage returns as well.

```
16246 \peek_regex_replace_once:nnF
16247 { \ *[\r*([~]*)\][^~\r]* }
16248 {
```

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.

```
16249 \c { group_end: }
16250 \c { tl_set_rescan:Nnn } \c { l_tmpa_tl } { } { \1 }
```

Then, we pass the retokenized content to the `\markdownSetup` macro.

```
16251 \c { @@_setup:V } \c { l_tmpa_tl }
```

Finally, regardless of whether or not we have matched the optional argument, we let the `\markdownReadAndConvert` macro process the rest of the  $\text{\LaTeX}$  environment.

We also make provision for using the `\markdown` command as a part of a different  $\text{\LaTeX}$  environment as follows:

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

```

16252         \c { exp_args:NV }
16253         \c { markdownReadAndConvert@ }
16254         \c { @currenvir }
16255     }
16256     {
16257         \group_end:
16258         \exp_args:NV
16259         \markdownReadAndConvert@
16260         \@currenvir
16261     }
16262 }
16263 { \markdownEnd }
16264 \renewenvironment
16265 { markdown* }
16266 [ 1 ]
16267 {
16268     \@@_if_option:nTF
16269     { experimental }
16270     {
16271         \msg_error:nn
16272         { markdown }
16273         { latex-markdown-star-deprecated }
16274     }
16275     {
16276         \msg_warning:nn
16277         { markdown }
16278         { latex-markdown-star-deprecated }
16279     }
16280     \@@_setup:n
16281     { #1 }
16282     \markdownReadAndConvert@
16283     { markdown* }
16284 }
16285 { \markdownEnd }
16286 \renewenvironment
16287 { yaml }
16288 {
16289     \group_begin:
16290     \yamlSetup
16291     { jekyllData, expectJekyllData, ensureJekyllData }
16292     \markdown
16293 }
16294 { \yamlEnd }
16295 \msg_new:nnn
16296 { markdown }
16297 { latex-markdown-star-deprecated }
16298 {

```

```

16299     The~markdown*~LaTeX~environment~has~been~deprecated~and~will~
16300     be~removed~in~the~next~major~version~of~the~Markdown~package.
16301   }
16302   \cs_generate_variant:Nn % noqa: w402
16303   \@@_setup:n
16304   { V }
16305   \ExplSyntaxOff
16306   \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*.

```

16307   \catcode`\|=0\catcode`\<=1\catcode`\>=2%
16308   \catcode`\|=12\catcode`\{=12\catcode`\}=12%
16309   \gdef|markdownReadAndConvert@#1<%
16310     |markdownReadAndConvert<\end{#1}>%
16311     <|end<#1>>>%
16312 |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.

```

16313 \ExplSyntaxOn
16314 \prop_new:N \g_@@_latex_loaded_themes_linenos_prop
16315 \prop_new:N \g_@@_latex_loaded_themes_versions_prop
16316 \cs_gset:Nn
16317   \@@_load_theme:nnn
16318   {

```

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.

```

16319   \ifmarkdownLaTeXLoaded
16320     \ifx\@onlypreamble\@notprerr

```

If both conditions are true, end with an error, since we cannot load  $\LaTeX$  themes after the preamble.

```

16321     \bool_if:nTF
16322     {
16323       \bool_lazy_or_p:nn
16324       {
16325         \prop_if_in_p:Nn
16326         \g_@@_latex_built_in_themes_prop

```

```

16327         { #1 }
16328     }
16329     {
16330         \file_if_exist_p:n
16331         { markdown theme #3.sty }
16332     }
16333 }
16334 {
16335     \msg_error:nn
16336     { markdown }
16337     { latex-theme-after-preamble }
16338 }

```

Otherwise, try loading a plain T<sub>E</sub>X theme instead.

```

16339     {
16340         \@@_plain_tex_load_theme:nnn
16341         { #1 }
16342         { #2 }
16343         { #3 }
16344     }
16345 \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.

```

16346     \bool_if:nTF
16347     {
16348         \bool_lazy_or_p:nn
16349         {
16350             \prop_if_in_p:Nn
16351             \g_@@_latex_built_in_themes_prop
16352             { #1 }
16353         }
16354         {
16355             \file_if_exist_p:n
16356             { markdown theme #3.sty }
16357         }
16358     }
16359     {
16360         \prop_get:NnNTF
16361         \g_@@_latex_loaded_themes_linenos_prop
16362         { #1 }
16363         \l_tmpa_tl
16364         {
16365             \prop_get:NnN
16366             \g_@@_latex_loaded_themes_versions_prop
16367             { #1 }
16368             \l_tmpb_tl
16369             \str_if_eq:nVTF

```

```

16370         { #2 }
16371     \l_tmpb_tl
16372     {
16373         \msg_warning:nnnVn
16374         { markdown }
16375         { repeatedly-loaded-latex-theme }
16376         { #1 }
16377         \l_tmpa_tl
16378         { #2 }
16379     }
16380     {
16381         \msg_error:nnnnVV
16382         { markdown }
16383         { different-versions-of-latex-theme }
16384         { #1 }
16385         { #2 }
16386         \l_tmpb_tl
16387         \l_tmpa_tl
16388     }
16389 }
16390 {
16391     \prop_gput:Nnx
16392     \g_@@_latex_loaded_themes_linenos_prop
16393     { #1 }
16394     { \tex_the:D \tex_inputlineno:D } % noqa: W200
16395     \prop_gput:Nnn
16396     \g_@@_latex_loaded_themes_versions_prop
16397     { #1 }
16398     { #2 }

```

Load built-in plain TeX themes from the prop `\g_@@_latex_built_in_themes_prop` and from the filesystem otherwise.

```

16399     \prop_if_in:NnTF
16400     \g_@@_latex_built_in_themes_prop
16401     { #1 }
16402     {
16403         \msg_info:nnnn
16404         { markdown }
16405         { loading-built-in-latex-theme }
16406         { #1 }
16407         { #2 }
16408         \prop_item:Nn
16409         \g_@@_latex_built_in_themes_prop
16410         { #1 }
16411     }
16412     {
16413         \msg_info:nnnn

```

```

16414             { markdown }
16415             { loading-latex-theme }
16416             { #1 }
16417             { #2 }
16418             \RequirePackage
16419             { markdown theme #3 }
16420         }
16421     }
16422 }
16423 {
16424     \@@_plain_tex_load_theme:nnn
16425     { #1 }
16426     { #2 }
16427     { #3 }
16428 }
16429 \fi
16430 \else

```

If the Markdown package has not yet been loaded, postpone the loading until the Markdown package has finished loading.

```

16431     \msg_info:nnnn
16432     { markdown }
16433     { theme-loading-postponed }
16434     { #1 }
16435     { #2 }
16436     \AtEndOfPackage
16437     {
16438         \@@_set_theme:n
16439         { #1 @ #2 }
16440     }
16441 \fi
16442 }
16443 \msg_new:nnn
16444 { markdown }
16445 { theme-loading-postponed }
16446 {
16447     Postponing~loading~version~#2~of~Markdown~theme~#1~until~
16448     Markdown~package~has~finished~loading
16449 }
16450 \msg_new:nnn
16451 { markdown }
16452 { loading-built-in-latex-theme }
16453 { Loading~version~#2~of~built-in~LaTeX~Markdown~theme~#1 }
16454 \msg_new:nnn
16455 { markdown }
16456 { loading-latex-theme }
16457 { Loading~version~#2~of~LaTeX~Markdown~theme~#1 }

```



```

16458 \msg_new:nnn
16459   { markdown }
16460   { repeatedly-loaded-latex-theme }
16461   {
16462     Version~#3~of~LaTeX~Markdown~theme~#1~was~previously~
16463     loaded~on~line~#2,~not~loading~it~again
16464   }
16465 \msg_new:nnn
16466   { markdown }
16467   { different-versions-of-latex-theme }
16468   {
16469     Tried~to~load~version~#2~of~LaTeX~Markdown~theme~#1~
16470     but~version~#3~has~already~been~loaded~on~line~#4
16471   }
16472 \cs_generate_variant:Nn
16473   \msg_new:nnnn
16474   { nnVV }
16475 \tl_set:Nn
16476   \l_tmpa_tl
16477   { Cannot~load~LaTeX~Markdown~theme~#1~after~ }
16478 \tl_put_right:NV
16479   \l_tmpa_tl
16480   \c_backslash_str
16481 \tl_put_right:Nn
16482   \l_tmpa_tl
16483   { begin { document } } }
16484 \tl_set:Nn
16485   \l_tmpb_tl
16486   { Load~Markdown~theme~#1~before~ }
16487 \tl_put_right:NV
16488   \l_tmpb_tl
16489   \c_backslash_str
16490 \tl_put_right:Nn
16491   \l_tmpb_tl
16492   { begin { document } } }
16493 \msg_new:nnVV
16494   { markdown }
16495   { latex-theme-after-preamble }
16496   \l_tmpa_tl
16497   \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.

```

16498 \tl_set:Nn
16499   \l_tmpa_tl
16500   {
16501     \RequirePackage

```

```

16502     { graphicx }
16503     \markdownLoadPlainTeXTheme
16504   }
16505   \prop_gput:NnV
16506     \g_@@_latex_built_in_themes_prop
16507     { witiko / dot }
16508     \l_tmpa_tl
16509   \prop_gput:NnV
16510     \g_@@_latex_built_in_themes_prop
16511     { witiko / graphicx / http }
16512     \l_tmpa_tl

```

The [witiko/glossaries](#) theme first checks that the version is either [v1](#) or [latest](#), issuing a warning for the latter.

```

16513   \prop_gput:Nnn
16514     \g_@@_latex_built_in_themes_prop
16515     { witiko / glossaries }
16516   {
16517     \str_case:enF
16518       { \markdownThemeVersion }
16519       {
16520         { latest }
16521         {
16522           \msg_warning:nnnn
16523             { markdown }
16524             { pin-theme-version }
16525             { witiko/glossaries }
16526             { v1 }
16527         }
16528         { v1 }
16529         { }
16530       }
16531     {
16532       \msg_error:nnnen
16533         { markdown }
16534         { unknown-theme-version }
16535         { witiko/glossaries }
16536         { \markdownThemeVersion }
16537         { v1 }
16538     }
16539   }
16540   \ExplSyntaxOff

```

In Section 3.3.3, we'll continue the implementation of the theme by defining the snippet [witiko/glossaries/import-acronyms](#). 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.

```

16541   \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 3.3.5 for the actual definitions.

### 3.3.3 Snippets

This section defines the built-in L<sup>A</sup>T<sub>E</sub>X snippet `witiko/glossaries/import-acronyms`.

```

16542 \ExplSyntaxOn
16543 \prop_get:NnN
16544   \g_@@_latex_built_in_themes_prop
16545   { witiko / glossaries }
16546   \l_tmpa_tl
16547 \tl_put_right:Nn
16548   \l_tmpa_tl
16549   {
16550     \cs_new:Nn
16551       \@@_register_acronym:n
16552       {
16553         \@@_setup:n
16554         {

```

When we register an acronym with the `acronyms` option, enclose it in an extra set of braces in case the acronym contained a comma.

```

16555           acronyms += {{ #1 }},
16556         }
16557       }
16558     \cs_generate_variant:Nn
16559       \@@_register_acronym:n
16560       { V }
16561     \prop_new:N
16562       \g_@@_acronyms_to_labels_prop
16563     \tl_new:N
16564       \l_@@_acronym_short_tl

```

When we import an acronym from the glossaries package, we receive a  $\langle label \rangle$ , which doesn't correspond to an acronym. Therefore, we register the short form of the corresponding acronym instead and keep a mapping from these short forms back to the labels when we need to display an acronym in the markdown text.

```

16565     \cs_new:Nn
16566       \@@_register_acronym_label:n
16567       {
16568         \tl_set:Nx
16569           \l_@@_acronym_short_tl
16570           { \glsentryshort { #1 } }

```

Be idempotent and only import each acronym once.

```

16571     \prop_if_in:NVF
16572       \g_@@_acronyms_to_labels_prop

```

```

16573         \l_@@_acronym_short_tl
16574     {
16575         \@@_register_acronym:V
16576         \l_@@_acronym_short_tl
16577         \prop_gput:NVn
16578         \g_@@_acronyms_to_labels_prop
16579         \l_@@_acronym_short_tl
16580         { #1 }
16581     }
16582 }
16583 \cs_generate_variant:Nn
16584 \@@_register_acronym_label:n
16585 { V }

```

When we need to display an acronym in the markdown text, first consult our mapping to see if it has been imported from the glossaries package.

```

16586 \tl_new:N
16587 \l_@@_acronym_label_tl
16588 \cs_new:Nn
16589 \@@_print_acronym:n
16590 {
16591     \prop_get:NnNTF
16592     \g_@@_acronyms_to_labels_prop
16593     { #1 }
16594     \l_@@_acronym_label_tl
16595     {

```

If it has, retrieve the  $\langle label \rangle$  and display the acronym using the glossaries package, linking it to the corresponding glossary entry.

```

16596         \group_begin:
16597         \cs_set_eq:NN
16598         \acronymfont
16599         \markdownRendererAcronymPrototype
16600         \exp_args:NV
16601         \acrshort
16602         \l_@@_acronym_label_tl
16603     \group_end:
16604 }
16605 {

```

Otherwise, just format the acronym using the current renderer prototype.

```

16606         \markdownRendererAcronymPrototype
16607         { #1 }
16608     }
16609 }
16610 \cs_generate_variant:Nn
16611 \@@_print_acronym:n
16612 { x }

```

In the snippet [witiko/glossaries/import-acronyms](#), first import all acronyms from the glossaries package and register them with the [acronyms](#) option.

```

16613 \markdownSetupSnippet
16614 { import-acronyms }
16615 {
16616   code = {
16617     \forlslentries
16618     [ acronym ]
16619     \l_@@_acronym_label_tl
16620     {
16621       \@@_register_acronym_label:V
16622       \l_@@_acronym_label_tl
16623     }

```

Then, redefine the corresponding token renderers to format acronyms that appear in the markdown text.

```

16624 \@@_setup:n
16625 {
16626   renderers = {
16627     acronym = {

```

Fold nested acronyms.

```

16628 \group_begin:
16629 \markdownSetup {
16630   unprotectedRenderers = {
16631     acronym = { ##1 },
16632   }
16633 }
16634 \@@_print_acronym:x
16635 { #1 }
16636 \group_end:
16637 }
16638 }
16639 }
16640 }
16641 }
16642 }
16643 \prop_gput:NnV
16644 \g_@@_latex_built_in_themes_prop
16645 { witiko / glossaries }
16646 \l_tmpa_tl
16647 \ExplSyntaxOff

```

### 3.3.4 Options

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

```

16648 \DeclareOption*{%

```

```

16649 \expandafter\markdownSetup\expandafter{\CurrentOption}}%
16650 \ProcessOptions\relax

```

### 3.3.5 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.

```

16651 \markdownIfOption{plain}{\iffalse}{\iftrue}

```

#### 3.3.5.1 Acronyms

If the `acronyms` option is non-empty, render acronyms, initialisms, and other all-caps sequences using small caps.

```

16652 \ExplSyntaxOn
16653 \int_new:N
16654 \g_@@_heading_nesting_depth_int
16655 \markdownSetup {
16656   rendererPrototypes = {
16657     acronym = {
16658       \int_if_zero:nTF
16659         { \g_@@_heading_nesting_depth_int }
16660         {

```

Prevent acronyms in headings.

```

16658       \int_if_zero:nTF
16659         { \g_@@_heading_nesting_depth_int }
16660         {

```

Prevent nested acronyms.

```

16661       \group_begin:
16662         \markdownSetup {
16663           unprotectedRenderers = {
16664             acronym = { ##1 },
16665           }
16666         }
16667       \tl_set:Nn
16668         \l_tmpa_tl
16669         { #1 }

```

Render all characters that are neither digits nor ASCII lower case characters in small caps. Ideally, we would also be able to exclude Unicode lower-case characters but that doesn't seem possible with `expl3` at the moment [20, Chapter 34 (The `l3unicode` module)].

```

16670       \regex_replace_all:nnN
16671         { [^\d a-z]+ }
16672         {
16673           \c{ textsc } \cB\{
16674             \c{ MakeLowercase } \cB\{ \0 \cE\}
16675           \cE\}
16676         }
16677       \l_tmpa_tl

```

Furthermore, if the current typeface supports it, render all digits using old-style numerals.

```

16678         \cs_if_exist:NT
16679             \oldstylenums
16680             {
16681                 \regex_replace_all:nnN
16682                 { \d+ }
16683                 { \c{ oldstylenums } \cB\{ \0 \cE\} }
16684                 \l_tmpa_tl
16685             }

```

If the package `microtype` is loaded, letter-space the small caps.

```

16686         \@ifpackageloaded
16687         { microtype }
16688         {
16689             \textls
16690             {
16691                 \tl_use:N
16692                 \l_tmpa_tl
16693             }
16694         }
16695         {
16696             \tl_use:N
16697             \l_tmpa_tl
16698         }
16699     \group_end:
16700 }

```

Prevent acronyms in headings.

```

16701         { #1 }
16702     },
16703 },
16704 }

```

If the package `glossaries` is loaded with the option `acronym`, use the built-in snippet `LATEX witiko/glossaries/import-acronyms`.

```

16705 \@ifpackageloaded
16706 { glossaries }
16707 {
16708     \cs_if_exist:NT
16709         \@gls@do@acronymsdef
16710         {
16711             \markdownSetup {
16712                 import = /witiko/glossaries@v1,
16713                 snippet = /witiko/glossaries/import-acronyms,
16714             }
16715         }
16716 }

```

```
16717 { }
```

### 3.3.5.2 Lists

If either the `tightLists` or the `fancyLists` 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`.

```
16718 \bool_new:N
16719   \g_@@_tight_or_fancy_lists_bool
16720 \bool_gset_false:N
16721   \g_@@_tight_or_fancy_lists_bool
16722 \@@_if_option:nTF
16723   { tightLists }
16724   {
16725     \bool_gset_true:N
16726       \g_@@_tight_or_fancy_lists_bool
16727   }
16728   {
16729     \@@_if_option:nT
16730       { fancyLists }
16731       {
16732         \bool_gset_true:N
16733           \g_@@_tight_or_fancy_lists_bool
16734       }
16735   }
16736 \bool_new:N
16737   \g_@@_beamer_paralist_or_enumitem_bool
16738 \bool_gset_true:N
16739   \g_@@_beamer_paralist_or_enumitem_bool
16740 \@ifclassloaded
16741   { beamer }
16742   { }
16743   {
16744     \@ifpackageloaded
16745       { paralist }
16746       { }
16747       {
16748         \@ifpackageloaded
16749           { enumitem }
16750           { }
16751           {
16752             \bool_gset_false:N
16753               \g_@@_beamer_paralist_or_enumitem_bool
16754           }
16755       }
16756   }
```



```

16755     }
16756   }
16757   \bool_if:nT
16758   {
16759     \g_@@_tight_or_fancy_lists_bool &&
16760     ! \g_@@_beamer_paralist_or_enumitem_bool
16761   }
16762   {
16763     \str_if_eq:enTF
16764     { \markdownThemeVersion }
16765     { experimental }
16766     {
16767       \RequirePackage
16768       { enumitem }
16769     }
16770     {
16771       \IfDocumentMetadataTF
16772       {
16773         \RequirePackage
16774         { enumitem }
16775       }
16776       {
16777         \RequirePackage
16778         { paralist }
16779       }
16780     }
16781   }
16782   \ExplSyntaxOff

```

If we loaded the enumitem package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```

16783   \ExplSyntaxOn
16784   \cs_new:Nn
16785     \@@_latex_fancy_list_item_label_number:nn
16786   {
16787     \str_case:nn
16788     { #1 }
16789     {
16790       { Decimal } { #2 }
16791       { LowerRoman } { \int_to_roman:n { #2 } }
16792       { UpperRoman } { \int_to_Roman:n { #2 } }
16793       { LowerAlpha } { \int_to_alph:n { #2 } }
16794       { UpperAlpha } { \int_to_Alph:n { #2 } }
16795     }
16796   }
16797   \cs_new:Nn
16798     \@@_latex_fancy_list_item_label_delimiter:n

```

```

16799 {
16800   \str_case:nn
16801     { #1 }
16802     {
16803       { Default } { . }
16804       { OneParen } { ) }
16805       { Period } { . }
16806     }
16807   }
16808 \cs_new:Nn
16809   \@@_latex_fancy_list_item_label:nnn
16810   {
16811     \@@_latex_fancy_list_item_label_number:nn
16812       { #1 }
16813       { #3 }
16814     \@@_latex_fancy_list_item_label_delimiter:n
16815       { #2 }
16816   }
16817 \cs_generate_variant:Nn
16818   \@@_latex_fancy_list_item_label:nnn
16819   { VVn }
16820 \tl_new:N
16821   \l_@@_latex_fancy_list_item_label_number_style_tl
16822 \tl_new:N
16823   \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16824 \@ifpackageloaded { enumitem } {
16825   \markdownSetup { rendererPrototypes = {
First, let's define the tight list item renderer prototypes.
16826     ulBeginTight = {
16827       \begin
16828         { itemize }
16829         [ noitemsep ]
16830     },
16831     ulEndTight = {
16832       \end
16833         { itemize }
16834     },
16835     olBeginTight = {
16836       \begin
16837         { enumerate }
16838         [ noitemsep ]
16839     },
16840     olEndTight = {
16841       \end
16842         { enumerate }
16843     },
16844     dlBeginTight = {

```

```

16845     \begin
16846     { description }
16847     [ noitemsep ]
16848   },
16849   dlEndTight = {
16850     \end
16851     { description }
16852   },

```

Second, let's define the fancy list item renderer prototypes.

```

16853   fancyOlBegin = {
16854     \group_begin:
16855     \tl_set:Nn
16856       \l_@@_latex_fancy_list_item_label_number_style_tl
16857       { #1 }
16858     \tl_set:Nn
16859       \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16860       { #2 }
16861     \begin
16862       { enumerate }
16863   },
16864   fancyOlBeginTight = {
16865     \group_begin:
16866     \tl_set:Nn
16867       \l_@@_latex_fancy_list_item_label_number_style_tl
16868       { #1 }
16869     \tl_set:Nn
16870       \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16871       { #2 }
16872     \begin
16873       { enumerate }
16874       [ noitemsep ]
16875   },
16876   fancyOlEnd(|Tight) = {
16877     \end { enumerate }
16878     \group_end:
16879   },
16880   fancyOlItemWithNumber = {
16881     \item
16882     [
16883       \@@_latex_fancy_list_item_label:VVn
16884       \l_@@_latex_fancy_list_item_label_number_style_tl
16885       \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16886       { #1 }
16887     ]
16888   },
16889 } }

```

Otherwise, if we loaded the paralist package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```
16890 }
16891 { \@ifpackageloaded { paralist } {
16892   \markdownSetup { rendererPrototypes = {
```

Make tight bullet lists a little less compact by adding extra vertical space above and below them.

```
16893     ulBeginTight = {
16894       \group_begin:
16895       \pltopsep=\topsep
16896       \plpartopsep=\partopsep
16897       \begin { compactitem }
16898     },
16899     ulEndTight = {
16900       \end { compactitem }
16901       \group_end:
16902     },
16903     fancyOlBegin = {
16904       \group_begin:
16905       \tl_set:Nn
16906         \l_@@_latex_fancy_list_item_label_number_style_tl
16907         { #1 }
16908       \tl_set:Nn
16909         \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16910         { #2 }
16911       \begin { enumerate }
16912     },
16913     fancyOlEnd = {
16914       \end { enumerate }
16915       \group_end:
16916     },
```

Make tight ordered lists a little less compact by adding extra vertical space above and below them.

```
16917     olBeginTight = {
16918       \group_begin:
16919       \plpartopsep=\partopsep
16920       \pltopsep=\topsep
16921       \begin { compactenum }
16922     },
16923     olEndTight = {
16924       \end { compactenum }
16925       \group_end:
16926     },
16927     fancyOlBeginTight = {
16928       \group_begin:
```

```

16929     \tl_set:Nn
16930     \l_@@_latex_fancy_list_item_label_number_style_tl
16931     { #1 }
16932     \tl_set:Nn
16933     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16934     { #2 }
16935     \plpartopsep=\partopsep
16936     \pltopsep=\topsep
16937     \begin { compactenum }
16938 },
16939 fancyOlEndTight = {
16940     \end { compactenum }
16941     \group_end:
16942 },
16943 fancyOlItemWithNumber = {
16944     \item
16945     [
16946         \@@_latex_fancy_list_item_label:VVn
16947         \l_@@_latex_fancy_list_item_label_number_style_tl
16948         \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16949         { #1 }
16950     ]
16951 },

```

Make tight definition lists a little less compact by adding extra vertical space above and below them.

```

16952     dlBeginTight = {
16953         \group_begin:
16954         \plpartopsep=\partopsep
16955         \pltopsep=\topsep
16956         \begin { compactdesc }
16957     },
16958     dlEndTight = {
16959         \end { compactdesc }
16960         \group_end:
16961     }
16962 } }
16963 }
16964 {

```

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.

```

16965     \markdownSetup
16966     {
16967         rendererPrototypes = {
16968             ulBeginTight = \markdownRendererUlBegin,
16969             ulEndTight = \markdownRendererUlEnd,

```

```

16970 fancyO1Begin = \markdownRendererO1Begin,
16971 fancyO1End = \markdownRendererO1End,
16972 olBeginTight = \markdownRendererO1Begin,
16973 olEndTight = \markdownRendererO1End,
16974 fancyO1BeginTight = \markdownRendererO1Begin,
16975 fancyO1EndTight = \markdownRendererO1End,
16976 dlBeginTight = \markdownRendererDlBegin,
16977 dlEndTight = \markdownRendererDlEnd,
16978 },
16979 }
16980 } }
16981 \ExplSyntaxOff
16982 \RequirePackage{amsmath}

```

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

```

16983 \@ifpackageloaded{unicode-math}{
16984   \markdownSetup{rendererPrototypes={
16985     untickedBox = {\mdlgwhtsquare$},
16986   }}
16987 }{
16988   \RequirePackage{amssymb}
16989   \markdownSetup{rendererPrototypes={
16990     untickedBox = {\square$},
16991   }}
16992 }
16993 \RequirePackage{csvsimple}
16994 \RequirePackage{fancyvrb}
16995 \RequirePackage{graphicx}
16996 \markdownSetup{rendererPrototypes={
16997   hardLineBreak = {\},
16998   leftBrace = {\textbraceleft},
16999   rightBrace = {\textbraceright},
17000   dollarSign = {\textdollar},
17001   underscore = {\textunderscore},
17002   circumflex = {\textasciicircum},
17003   backslash = {\textbackslash},
17004   tilde = {\textasciitilde},
17005   pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by  $\text{T}_{\text{E}}\text{X}$  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>37</sup> we can reliably detect math mode inside the renderer.

---

<sup>37</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.

Here, we will redefine the code span renderer prototype to typeset upright text in math formulae and typewriter text outside math formulae.

```

17006   codeSpan = {%
17007       \ifmmode
17008         \text{#1}%
17009       \else
17010         \texttt{#1}%
17011       \fi
17012   }}}
```

### 3.3.5.3 Content Blocks

In content block renderer prototypes, display the content as a table using the package `csvsimple` when the raw attribute is `csv`, display the content using the default templates of the package `luaxml` when the raw attribute is `html`, execute the content with TeX when the raw attribute is `tex`, and display the content as markdown otherwise.

```

17013 \ExplSyntaxOn
17014 \markdownSetup{
17015   rendererPrototypes = {
17016     contentBlock = {
17017       \str_case:nnF
17018         { #1 }
17019         {
17020           { csv }
17021           {
17022             \begin { table }
17023               \begin { center }
17024                 \csvautotabular { #3 }
17025               \end { center }
17026             \tl_if_empty:nF
17027               { #4 }
17028               { \caption { #4 } }
17029             \end { table }
17030           }
17031         { html }
17032         {
```

If we are using `TeX4ht`<sup>38</sup>, we will pass HTML elements to the output HTML document unchanged.

```

17033       \cs_if_exist:NTF
17034       \HCode
17035       {
17036         \if_mode_vertical:
17037         \IgnorePar
```

---

<sup>38</sup>See <https://tug.org/tex4ht/>.

```

17038         \fi:
17039         \EndP
17040         \special
17041           { t4ht* < #3 }
17042         \par
17043         \ShowPar
17044       }
17045       {
17046         \@@_luaxml_print_html:n
17047           { #3 }
17048       }
17049     }
17050   { tex }
17051   {
17052     \markdownEscape
17053       { #3 }
17054   }
17055 }
17056 {
17057   \markdownInput
17058     { #3 }
17059 }
17060 },
17061 },
17062 }
17063 \ExplSyntaxOff
17064 \markdownSetup{rendererPrototypes={
17065   ulBegin = {\begin{itemize}},
17066   ulEnd = {\end{itemize}},
17067   olBegin = {\begin{enumerate}},
17068   olItem = {\item{}},
17069   olItemWithNumber = {\item[#1.]},
17070   olEnd = {\end{enumerate}},
17071   dlBegin = {\begin{description}},
17072   dlItem = {\item[#1]},
17073   dlEnd = {\end{description}},
17074   emphasis = {\emph{#1}},
17075   tickedTextBox = {\$\boxtimes$},
17076   halfTickedTextBox = {\$\boxdot$}}

```

If HTML identifiers appear after a heading, we make them produce `\label` macros.

```

17077 \ExplSyntaxOn
17078 \seq_new:N
17079   \g_@@_header_identifiers_seq
17080 \markdownSetup
17081 {
17082   rendererPrototypes = {
17083     headerAttributeContextBegin = {

```



```

17084     \markdownSetup
17085     {
17086         rendererPrototypes = {
17087             attributeIdentifier = {
17088                 \seq_gput_right:Nn
17089                 \g_@@_header_identifiers_seq
17090                 { ##1 }
17091             },
17092         },
17093     },
17094     headerAttributeContextEnd = {
17095         \seq_map_inline:Nn
17096         \g_@@_header_identifiers_seq
17097         { \label { ##1 } }
17098         \seq_gclear:N
17099         \g_@@_header_identifiers_seq
17100     },
17101 },
17102 },
17103 }

```

If the `unnumbered` HTML class (or the `{-}` shorthand) appears after a heading the heading and all its subheadings will be unnumbered.

```

17104 \bool_new:N
17105 \l_@@_header_unnumbered_bool
17106 \markdownSetup
17107 {
17108     rendererPrototypes = {
17109         headerAttributeContextBegin += {
17110             \markdownSetup
17111             {
17112                 rendererPrototypes = {
17113                     attributeClassName = {
17114                         \bool_if:nT
17115                         {
17116                             \str_if_eq_p:nn
17117                             { ##1 }
17118                             { unnumbered } &&
17119                             ! \l_@@_header_unnumbered_bool
17120                         }
17121                     {
17122                         \group_begin:
17123                         \bool_set_true:N
17124                         \l_@@_header_unnumbered_bool
17125                         \c@secnumdepth = -2
17126                         \markdownSetup
17127                         {

```

```

17128             rendererPrototypes = {
17129                 sectionBegin = {
17130                     \group_begin:
17131                 },
17132                 sectionEnd = {
17133                     \group_end:
17134                 },
17135             },
17136         },
17137     },
17138 },
17139 },
17140 },
17141 },
17142 },
17143 }
17144 \ExplSyntaxOff
17145 \markdownSetup{rendererPrototypes={
17146     superscript = {\textsuperscript{#1}},
17147     subscript = {\textsubscript{#1}},
17148     blockQuoteBegin = {\begin{quotation}},
17149     blockQuoteEnd = {\end{quotation}},
17150     inputVerbatim = {\VerbatimInput{#1}},
17151     thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
17152     note = {\footnote{#1}}}}

```

### 3.3.5.4 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

17153 \RequirePackage{ltxcmds}
17154 \ExplSyntaxOn
17155 \cs_gset_protected:Npn
17156   \markdownRendererInputFencedCodePrototype#1#2#3
17157   {
17158     \tl_if_empty:nTF
17159       { #2 }
17160       { \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.

```

17161     {
17162       \regex_extract_once:nnN
17163         { \w* }
17164         { #2 }
17165       \l_tmpa_seq
17166       \seq_pop_left:NN
17167       \l_tmpa_seq
17168       \l_tmpa_tl

```

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

```

17169      \ltx@ifpackageloaded
17170      { minted }
17171      {
17172        \catcode`\%=14\relax
17173        \catcode`\#=6\relax
17174        \exp_args:NV
17175        \inputminted
17176        \l_tmpa_tl
17177        { #1 }
17178        \catcode`\%=12\relax
17179        \catcode`\#=12\relax
17180      }
17181      {

```

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

```

17182      \ltx@ifpackageloaded
17183      { listings }
17184      { \lstinputlisting [ language = \l_tmpa_tl ] { #1 } }

```

When neither the listings package nor the minted package is loaded, act as though no infostring were given.

```

17185      { \markdownRendererInputFencedCode { #1 } { } { } }
17186    }
17187  }
17188 }
17189 \ExplSyntaxOff

```

Support the nesting of strong emphasis.

```

17190 \ExplSyntaxOn
17191 \def\markdownLATEXStrongEmphasis#1{
17192   \str_if_in:NnTF
17193     \f@series
17194     { b }
17195     { \textnormal{#1} }
17196     { \textbf{#1} }
17197 }
17198 \ExplSyntaxOff
17199 \markdownSetup{rendererPrototypes={strongEmphasis={%
17200   \protect\markdownLATEXStrongEmphasis{#1}}}}

```

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

```

17201 \@ifundefined{chapter}{%
17202   \markdownSetup{rendererPrototypes = {
17203     headingOne = {\section{#1}},
17204     headingTwo = {\subsection{#1}},
17205     headingThree = {\subsubsection{#1}},
17206     headingFour = {\paragraph{#1}},

```

```

17207     headingFive = {\subparagraph{#1}}}}
17208 }{%
17209   \markdownSetup{rendererPrototypes = {
17210     headingOne = {\chapter{#1}},
17211     headingTwo = {\section{#1}},
17212     headingThree = {\subsection{#1}},
17213     headingFour = {\subsubsection{#1}},
17214     headingFive = {\paragraph{#1}},
17215     headingSix = {\subparagraph{#1}}}}
17216 }%

```

Prevent acronyms in headings.

```

17217 \ExplSyntaxOn
17218 \markdownSetup {
17219   rendererPrototypes = {
17220     heading* ^= {
17221       \int_gincr:N
17222       \g_@@_heading_nesting_depth_int
17223     },
17224     heading* $= {
17225       \int_gdecr:N
17226       \g_@@_heading_nesting_depth_int
17227     }
17228   }
17229 }
17230 \ExplSyntaxOff

```

### 3.3.5.5 Tickboxes

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

```

17231 \markdownSetup{
17232   rendererPrototypes = {
17233     ulItem = {%
17234       \futurelet\markdownLaTeXCheckbox\markdownLaTeXUListItem
17235     },
17236   },
17237 }
17238 \def\markdownLaTeXUListItem{%
17239   \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
17240     \item[\markdownLaTeXCheckbox]%
17241     \expandafter\@gobble
17242   \else
17243     \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
17244       \item[\markdownLaTeXCheckbox]%
17245       \expandafter\expandafter\expandafter\@gobble
17246     \else
17247       \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox

```

```

17248         \item[\markdownLaTeXCheckbox]%
17249         \expandafter\expandafter\expandafter\expandafter
17250         \expandafter\expandafter\expandafter\@gobble
17251     \else
17252         \item{}%
17253     \fi
17254 \fi
17255 \fi
17256 }

```

### 3.3.5.6 html elements

If the `html` option is enabled and we are using  $\text{\TeX}4\text{ht}$ <sup>39</sup>, we will pass HTML elements to the output HTML document unchanged.

```

17257 \@ifundefined{HCode}{}{
17258     \markdownSetup{
17259         rendererPrototypes = {
17260             inlineHtmlTag = {%
17261                 \ifvmode
17262                     \IgnorePar
17263                 \EndP
17264                 \fi
17265                 \HCode{#1}%
17266             },
17267             inputBlockHtmlElement = {%
17268                 \ifvmode
17269                     \IgnorePar
17270                 \fi
17271                 \EndP
17272                 \special{t4ht*<#1}%
17273                 \par
17274                 \ShowPar
17275             },
17276         },
17277     }
17278 }

```

### 3.3.5.7 Citations

Here is a basic implementation for citations that uses the  $\text{\LaTeX}$  `\cite` macro. There are also implementations that use the `natbib` `\citep`, and `\citete` macros, and the `Bib $\text{\LaTeX}$`  `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```

17279 \newcount\markdownLaTeXCitationsCounter
17280

```

---

<sup>39</sup>See <https://tug.org/tex4ht/>.

```

17281 % Basic implementation
17282 \long\def\@gobblethree#1#2#3{}%
17283 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
17284   \advance\markdownLaTeXCitationsCounter by 1\relax
17285   \ifx\relax#4\relax
17286     \ifx\relax#5\relax
17287       \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
17288         \relax
17289         \cite{#1#2#6}% No prenotes/postnotes, just accumulate cites
17290         \expandafter\expandafter\expandafter
17291         \expandafter\expandafter\expandafter\expandafter
17292         \@gobblethree
17293       \fi
17294     \else% Before a postnote (#5), dump the accumulator
17295       \ifx\relax#1\relax\else
17296         \cite{#1}%
17297       \fi
17298       \cite[#5]{#6}%
17299       \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
17300         \relax
17301       \else
17302         \expandafter\expandafter\expandafter
17303         \expandafter\expandafter\expandafter\expandafter
17304         \expandafter\expandafter\expandafter
17305         \expandafter\expandafter\expandafter\expandafter
17306         \markdownLaTeXBasicCitations
17307       \fi
17308       \expandafter\expandafter\expandafter
17309       \expandafter\expandafter\expandafter\expandafter{%
17310       \expandafter\expandafter\expandafter
17311       \expandafter\expandafter\expandafter\expandafter}%
17312       \expandafter\expandafter\expandafter
17313       \expandafter\expandafter\expandafter\expandafter{%
17314       \expandafter\expandafter\expandafter
17315       \expandafter\expandafter\expandafter\expandafter}%
17316       \expandafter\expandafter\expandafter
17317       \@gobblethree
17318     \fi
17319   \else% Before a prenote (#4), dump the accumulator
17320     \ifx\relax#1\relax\else
17321       \cite{#1}%
17322     \fi
17323     \ifnum\markdownLaTeXCitationsCounter>1\relax
17324       \space % Insert a space before the prenote in later citations
17325     \fi
17326     #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
17327     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal

```

```

17328 \relax
17329 \else
17330 \expandafter\expandafter\expandafter
17331 \expandafter\expandafter\expandafter\expandafter
17332 \markdownLaTeXBasicCitations
17333 \fi
17334 \expandafter\expandafter\expandafter{%
17335 \expandafter\expandafter\expandafter}%
17336 \expandafter\expandafter\expandafter{%
17337 \expandafter\expandafter\expandafter}%
17338 \expandafter
17339 \@gobblethree
17340 \fi\markdownLaTeXBasicCitations{#1#2#6},}
17341 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
17342
17343 % Natbib implementation
17344 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
17345 \advance\markdownLaTeXCitationsCounter by 1\relax
17346 \ifx\relax#3\relax
17347 \ifx\relax#4\relax
17348 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
17349 \relax
17350 \citep{#1,#5}% No prenotes/postnotes, just accumulate cites
17351 \expandafter\expandafter\expandafter
17352 \expandafter\expandafter\expandafter\expandafter
17353 \@gobbletwo
17354 \fi
17355 \else% Before a postnote (#4), dump the accumulator
17356 \ifx\relax#1\relax\else
17357 \citep{#1}%
17358 \fi
17359 \citep[] [#4]{#5}%
17360 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
17361 \relax
17362 \else
17363 \expandafter\expandafter\expandafter
17364 \expandafter\expandafter\expandafter\expandafter
17365 \expandafter\expandafter\expandafter
17366 \expandafter\expandafter\expandafter\expandafter
17367 \markdownLaTeXNatbibCitations
17368 \fi
17369 \expandafter\expandafter\expandafter
17370 \expandafter\expandafter\expandafter\expandafter{%
17371 \expandafter\expandafter\expandafter
17372 \expandafter\expandafter\expandafter\expandafter}%
17373 \expandafter\expandafter\expandafter
17374 \@gobbletwo

```

```

17375 \fi
17376 \else% Before a prenote (#3), dump the accumulator
17377 \ifx\relax#1\relax\relax\else
17378 \citep{#1}%
17379 \fi
17380 \citep[#3][#4]{#5}%
17381 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
17382 \relax
17383 \else
17384 \expandafter\expandafter\expandafter
17385 \expandafter\expandafter\expandafter\expandafter
17386 \markdownLaTeXNatbibCitations
17387 \fi
17388 \expandafter\expandafter\expandafter{%
17389 \expandafter\expandafter\expandafter}%
17390 \expandafter
17391 \@gobbletwo
17392 \fi\markdownLaTeXNatbibCitations{#1,#5}}
17393 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
17394 \advance\markdownLaTeXCitationsCounter by 1\relax
17395 \ifx\relax#3\relax
17396 \ifx\relax#4\relax
17397 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
17398 \relax
17399 \citet{#1,#5}% No prenotes/postnotes, just accumulate cites
17400 \expandafter\expandafter\expandafter
17401 \expandafter\expandafter\expandafter\expandafter
17402 \@gobbletwo
17403 \fi
17404 \else% After a prenote or a postnote, dump the accumulator
17405 \ifx\relax#1\relax\else
17406 \citet{#1}%
17407 \fi
17408 , \citet[#3][#4]{#5}%
17409 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
17410 \relax
17411 ,
17412 \else
17413 \ifnum
17414 \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
17415 \relax
17416 ,
17417 \fi
17418 \fi
17419 \expandafter\expandafter\expandafter
17420 \expandafter\expandafter\expandafter\expandafter
17421 \markdownLaTeXNatbibTextCitations

```



```

17422     \expandafter\expandafter\expandafter
17423     \expandafter\expandafter\expandafter\expandafter{%
17424     \expandafter\expandafter\expandafter
17425     \expandafter\expandafter\expandafter\expandafter}%
17426     \expandafter\expandafter\expandafter
17427     \@gobbletwo
17428     \fi
17429 \else% After a prenote or a postnote, dump the accumulator
17430     \ifx\relax#1\relax\relax\else
17431         \citet{#1}%
17432     \fi
17433     , \citet[#3][#4]{#5}%
17434     \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
17435     \relax
17436     ,
17437     \else
17438         \ifnum
17439             \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
17440         \relax
17441         ,
17442         \fi
17443     \fi
17444     \expandafter\expandafter\expandafter
17445     \markdownLaTeXNatbibTextCitations
17446     \expandafter\expandafter\expandafter{%
17447     \expandafter\expandafter\expandafter}%
17448     \expandafter
17449     \@gobbletwo
17450 \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
17451
17452 % BibLaTeX implementation
17453 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
17454     \advance\markdownLaTeXCitationsCounter by 1\relax
17455     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
17456     \relax
17457         \autocites#1[#3][#4]{#5}%
17458     \expandafter\@gobbletwo
17459     \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
17460 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
17461     \advance\markdownLaTeXCitationsCounter by 1\relax
17462     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
17463     \relax
17464         \textcites#1[#3][#4]{#5}%
17465     \expandafter\@gobbletwo
17466     \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}}
17467
17468 \markdownSetup{rendererPrototypes = {

```

```

17469 cite = {%
17470     \markdownLaTeXCitationsCounter=1%
17471     \def\markdownLaTeXCitationsTotal{#1}%
17472     \@ifundefined{autocites}{%
17473         \@ifundefined{citep}{%
17474             \expandafter\expandafter\expandafter
17475             \markdownLaTeXBasicCitations
17476             \expandafter\expandafter\expandafter{%
17477                 \expandafter\expandafter\expandafter}%
17478             \expandafter\expandafter\expandafter{%
17479                 \expandafter\expandafter\expandafter}%
17480         }{%
17481             \expandafter\expandafter\expandafter
17482             \markdownLaTeXNatbibCitations
17483             \expandafter\expandafter\expandafter{%
17484                 \expandafter\expandafter\expandafter}%
17485         }%
17486     }{%
17487         \expandafter\expandafter\expandafter
17488         \markdownLaTeXBibLaTeXCitations
17489         \expandafter{\expandafter}%
17490     }},
17491 textCite = {%
17492     \markdownLaTeXCitationsCounter=1%
17493     \def\markdownLaTeXCitationsTotal{#1}%
17494     \@ifundefined{autocites}{%
17495         \@ifundefined{citep}{%
17496             \expandafter\expandafter\expandafter
17497             \markdownLaTeXBasicTextCitations
17498             \expandafter\expandafter\expandafter{%
17499                 \expandafter\expandafter\expandafter}%
17500             \expandafter\expandafter\expandafter{%
17501                 \expandafter\expandafter\expandafter}%
17502         }{%
17503             \expandafter\expandafter\expandafter
17504             \markdownLaTeXNatbibTextCitations
17505             \expandafter\expandafter\expandafter{%
17506                 \expandafter\expandafter\expandafter}%
17507         }%
17508     }{%
17509         \expandafter\expandafter\expandafter
17510         \markdownLaTeXBibLaTeXTextCitations
17511         \expandafter{\expandafter}%
17512     }}}}

```

### 3.3.5.8 Links

Here is an implementation for hypertext links and relative references.

```

17513 \RequirePackage{url}
17514 \RequirePackage{expl3}
17515 \ExplSyntaxOn
17516 \cs_gset_protected:Npn
17517   \markdownRendererLinkPrototype
17518   #1#2#3#4
17519   {
17520     \tl_set:Nn \l_tmpa_tl { #1 }
17521     \tl_set:Nn \l_tmpb_tl { #2 }
17522     \bool_set:Nn
17523       \l_tmpa_bool
17524       {
17525         \tl_if_eq_p:NN
17526           \l_tmpa_tl
17527           \l_tmpb_tl
17528       }
17529     \tl_set:Nn \l_tmpa_tl { #4 }
17530     \bool_set:Nn
17531       \l_tmpb_bool
17532       {
17533         \tl_if_empty_p:N
17534           \l_tmpa_tl
17535       }

```

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.

```

17536   \bool_if:nTF
17537     {
17538       \l_tmpa_bool && \l_tmpb_bool
17539     }
17540     {
17541       \markdownLaTeXRendererAutolink { #2 } { #3 }
17542     }
17543     {
17544       \markdownLaTeXRendererDirectOrIndirectLink
17545         { #1 } { #2 } { #3 } { #4 }
17546     }
17547   }
17548 \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.

```

17549   \tl_set:Nn
17550     \l_tmpa_tl
17551     { #2 }

```

```

17552 \tl_trim_spaces:N
17553 \l_tmpa_tl
17554 \tl_set:Nx
17555 \l_tmpb_tl
17556 {
17557 \tl_range:Nnn
17558 \l_tmpa_tl
17559 { 1 }
17560 { 1 }
17561 }
17562 \str_if_eq:NNTF
17563 \l_tmpb_tl
17564 \c_hash_str
17565 {
17566 \tl_set:Nx
17567 \l_tmpb_tl
17568 {
17569 \tl_range:Nnn
17570 \l_tmpa_tl
17571 { 2 }
17572 { -1 }
17573 }
17574 \exp_args:NV
17575 \ref
17576 \l_tmpb_tl
17577 }
17578 {
17579 \url { #2 }
17580 }
17581 }
17582 \ExplSyntaxOff
17583 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
17584 #1\footnote{\ifx\empty#4\empty\else#4: \fi\url{#3}}}%

```

### 3.3.5.9 Tables

Here is a basic implementation of tables. If the booktabs package is loaded, then it is used to produce horizontal lines.

```

17585 \newcount\markdownLaTeXRowCount
17586 \newcount\markdownLaTeXRowTotal
17587 \newcount\markdownLaTeXColumnCounter
17588 \newcount\markdownLaTeXColumnTotal
17589 \newtoks\markdownLaTeXTable
17590 \newtoks\markdownLaTeXTableAlignment
17591 \newtoks\markdownLaTeXTableEnd
17592 \AtBeginDocument{%
17593 \ifpackageloaded{booktabs}{%

```

```

17594 \def\markdownLaTeXTopRule{\toprule}%
17595 \def\markdownLaTeXMidRule{\midrule}%
17596 \def\markdownLaTeXBottomRule{\bottomrule}%
17597 }{%
17598 \def\markdownLaTeXTopRule{\hline}%
17599 \def\markdownLaTeXMidRule{\hline}%
17600 \def\markdownLaTeXBottomRule{\hline}%
17601 }%
17602 }
17603 \markdownSetup{rendererPrototypes={
17604   table = {%
17605     \markdownLaTeXTable={}%
17606     \markdownLaTeXTableAlignment={}%
17607     \markdownLaTeXTableEnd={%
17608       \markdownLaTeXBottomRule
17609     \end{tabular}}%
17610     \ifx\empty#1\empty\else
17611       \addto@hook\markdownLaTeXTable{%
17612         \begin{table}
17613         \centering}%
17614       \addto@hook\markdownLaTeXTableEnd{%
17615         \caption{#1}}%
17616     \fi
17617   }
17618 }}

```

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.

```

17619 \ExplSyntaxOn
17620 \seq_new:N
17621 \l_@@_table_identifiers_seq
17622 \markdownSetup {
17623   rendererPrototypes = {
17624     table += {
17625       \seq_map_inline:Nn
17626       \l_@@_table_identifiers_seq
17627       {
17628         \addto@hook
17629         \markdownLaTeXTableEnd
17630         { \label { ##1 } }
17631       }
17632     },
17633   }
17634 }
17635 \markdownSetup {
17636   rendererPrototypes = {
17637     tableAttributeContextBegin = {

```

```

17638     \group_begin:
17639     \markdownSetup {
17640         rendererPrototypes = {
17641             attributeIdentifier = {
17642                 \seq_put_right:Nn
17643                 \l_@@_table_identifiers_seq
17644                 { ##1 }
17645             },
17646         },
17647     }
17648 },
17649 tableAttributeContextEnd = {
17650     \group_end:
17651 },
17652 },
17653 }
17654 \ExplSyntaxOff
17655 \markdownSetup{rendererPrototypes={
17656     table += {%
17657         \ifx\empty#1\empty\else
17658             \addto@hook\markdownLaTeXTableEnd{%
17659                 \end{table}}%
17660         \fi
17661         \addto@hook\markdownLaTeXTable{\begin{tabular}}%
17662         \markdownLaTeXRowCounter=0%
17663         \markdownLaTeXRowTotal=#2%
17664         \markdownLaTeXColumnTotal=#3%
17665         \markdownLaTeXRenderTableRow
17666     }
17667 }}
17668 \def\markdownLaTeXRenderTableRow#1{%
17669     \markdownLaTeXColumnCounter=0%
17670     \ifnum\markdownLaTeXRowCounter=0\relax
17671         \markdownLaTeXReadAlignments#1%
17672         \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
17673             \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
17674                 \the\markdownLaTeXTableAlignment}}%
17675         \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
17676     \else
17677         \markdownLaTeXRenderTableCell#1%
17678     \fi
17679     \ifnum\markdownLaTeXRowCounter=1\relax
17680         \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
17681     \fi
17682     \advance\markdownLaTeXRowCounter by 1\relax
17683     \ifnum\markdownLaTeXRowCounter>\markdownLaTeXRowTotal\relax
17684         \the\markdownLaTeXTable

```

```

17685 \the\markdownLaTeXTableEnd
17686 \expandafter\@gobble
17687 \fi\markdownLaTeXRenderTableRow}
17688 \def\markdownLaTeXReadAlignments#1{%
17689 \advance\markdownLaTeXColumnCounter by 1\relax
17690 \if#1d%
17691 \addto@hook\markdownLaTeXTableAlignment{1}%
17692 \else
17693 \addto@hook\markdownLaTeXTableAlignment{#1}%
17694 \fi
17695 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
17696 \expandafter\@gobble
17697 \fi\markdownLaTeXReadAlignments}
17698 \def\markdownLaTeXRenderTableCell#1{%
17699 \advance\markdownLaTeXColumnCounter by 1\relax
17700 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
17701 \addto@hook\markdownLaTeXTable{#1&}%
17702 \else
17703 \addto@hook\markdownLaTeXTable{#1\\}%
17704 \expandafter\@gobble
17705 \fi\markdownLaTeXRenderTableCell}

```

### 3.3.5.10 Line Blocks

Here is a basic implementation of line blocks. If the verse package is loaded, then it is used to produce the verses.

```

17706
17707 \markdownIfOption{lineBlocks}{%
17708 \RequirePackage{verse}
17709 \markdownSetup{rendererPrototypes={
17710 lineBlockBegin = {%
17711 \begingroup
17712 \def\markdownRendererHardLineBreak{\\}%
17713 \begin{verse}%
17714 },
17715 lineBlockEnd = {%
17716 \end{verse}%
17717 \endgroup
17718 },
17719 }}
17720 }{}
17721

```

### 3.3.5.11 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.

```

17722 \ExplSyntaxOn
17723 \keys_define:nn
17724 { markdown / jekyllData }
17725 {
17726   author .code:n = {
17727     \author
17728     { #1 }
17729   },
17730   date .code:n = {
17731     \date
17732     { #1 }
17733   },
17734   title .code:n = {
17735     \title
17736     { #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, temporarily resetting the category codes of the percent sign and the hash sign back to comment and parameter, respectively.

```

17737   \char_set_catcode_comment:N \%
17738   \char_set_catcode_parameter:N \#
17739   \AddToHook
17740   { begindocument / end }
17741   { \maketitle }
17742   \char_set_catcode_other:N \%
17743   \char_set_catcode_other:N \#
17744 },
17745 }

```

### 3.3.5.12 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.

```

17746 \@@_if_option:nT
17747 { mark }
17748 {
17749   \sys_if_engine luatex:TF
17750   {
17751     \RequirePackage
17752     { luacolor }

```



```

17753     \RequirePackage
17754     { lua-ul }
17755     \markdownSetup
17756     {
17757         rendererPrototypes = {
17758             mark = {
17759                 \highLight
17760                 { #1 }
17761             },
17762         }
17763     }
17764 }
17765 {
17766     \RequirePackage
17767     { xcolor }
17768     \RequirePackage
17769     { soul }
17770     \markdownSetup
17771     {
17772         rendererPrototypes = {
17773             mark = {
17774                 \hl
17775                 { #1 }
17776             },
17777         }
17778     }
17779 }
17780 }

```

### 3.3.5.13 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.

```

17781 \@@_if_option:nT
17782 { strikeThrough }
17783 {
17784     \sys_if_engine luatex:TF
17785     {
17786         \RequirePackage
17787         { lua-ul }
17788         \markdownSetup
17789         {
17790             rendererPrototypes = {
17791                 strikeThrough = {
17792                     \strikeThrough
17793                     { #1 }
17794                 },

```

```

17795         }
17796     }
17797 }
17798 {
17799     \RequirePackage
17800     { soul }
17801     \markdownSetup
17802     {
17803         rendererPrototypes = {
17804             strikeThrough = {
17805                 \st
17806                 { #1 }
17807             },
17808         }
17809     }
17810 }
17811 }

```

### 3.3.5.14 Images and their attributes

We define images to be rendered as floating figures using the command `\includegraphics`, where the image label is the alt text and the image title is the caption of the figure.

If the `linkAttributes` option is enabled, we will make attributes in the form  $\langle key \rangle = \langle value \rangle$  set the corresponding keys of the `graphicx` package to the corresponding values and we will register any identifiers, so that they can be used as  $\text{\LaTeX}$  labels for referencing figures.

```

17812 \seq_new:N
17813 \l_@@_image_identifiers_seq
17814 \markdownSetup {
17815     rendererPrototypes = {
17816         image = {
17817             \tl_if_empty:nTF
17818             { #4 }
17819             {
17820                 \begin { center }
17821                     \includegraphics
17822                     [ alt = { #1 } ]
17823                     { #3 }
17824                 \end { center }
17825             }
17826         {
17827             \begin { figure }
17828                 \begin { center }
17829                     \includegraphics
17830                     [ alt = { #1 } ]
17831                     { #3 }

```

```

17832             \caption { #4 }
17833             \seq_map_inline:Nn
17834             \l_@@_image_identifiers_seq
17835             { \label { ##1 } }
17836         \end { center }
17837     \end { figure }
17838 }
17839 },
17840 }
17841 }
17842 \@@_if_option:nT
17843 { linkAttributes }
17844 {
17845     \RequirePackage { graphicx }
17846 }
17847 \markdownSetup {
17848     rendererPrototypes = {
17849         imageAttributeContextBegin = {
17850             \group_begin:
17851             \markdownSetup {
17852                 rendererPrototypes = {
17853                     attributeIdentifier = {
17854                         \seq_put_right:Nn
17855                         \l_@@_image_identifiers_seq
17856                         { ##1 }
17857                     },
17858                     attributeKeyValue = {
17859                         \setkeys
17860                         { Gin }
17861                         { { ##1 } = { ##2 } }
17862                     },
17863                 },
17864             }
17865         },
17866         imageAttributeContextEnd = {
17867             \group_end:
17868         },
17869     },
17870 }
17871 \ExplSyntaxOff

```

### 3.3.5.15 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`.

```

17872 \ExplSyntaxOn
17873 \cs_new:Nn
17874   \@@_luaxml_print_html:n
17875   {
17876     \luabridge_now:n
17877     {
17878       local~input_file = assert(io.open(" #1 ", "r"))
17879       local~input = assert(input_file:read("*a"))
17880       assert(input_file:close())
17881       input = "<body>" .. input .. "</body>"
17882       local~dom = require("luaxml-domobject").html_parse(input)
17883       local~output = require("luaxml-htmltemplates"):process_dom(dom)
17884       print(output)
17885     }
17886   }
17887 \cs_gset_protected:Npn
17888   \markdownRendererInputRawInlinePrototype#1#2
17889   {
17890     \str_case:nnF
17891       { #2 }
17892       {
17893         { latex }
17894         {
17895           \@@_plain_tex_default_input_raw_inline:nn
17896           { #1 }
17897           { tex }
17898         }
17899         { html }
17900         {

```

If we are using  $\text{\TeX}4\text{ht}$ <sup>40</sup>, we will pass HTML elements to the output HTML document unchanged.

```

17901       \cs_if_exist:NTF
17902       \HCode
17903       {
17904         \if_mode_vertical:
17905         \IgnorePar
17906         \EndP
17907         \fi:
17908         \special
17909         { t4ht* < #1 }
17910       }
17911       {
17912         \@@_luaxml_print_html:n
17913         { #1 }
17914       }

```

---

<sup>40</sup>See <https://tug.org/tex4ht/>.

```

17915         }
17916     }
17917     {
17918         \@@_plain_tex_default_input_raw_inline:nn
17919         { #1 }
17920         { #2 }
17921     }
17922 }
17923 \cs_gset_protected:Npn
17924   \markdownRendererInputRawBlockPrototype#1#2
17925   {
17926     \str_case:nnF
17927       { #2 }
17928       {
17929         { latex }
17930         {
17931           \@@_plain_tex_default_input_raw_block:nn
17932           { #1 }
17933           { tex }
17934         }
17935         { html }
17936         {

```

If we are using  $\text{\TeX}4\text{ht}$ <sup>41</sup>, we will pass HTML elements to the output HTML document unchanged.

```

17937         \cs_if_exist:NTF
17938         \HCode
17939         {
17940           \if_mode_vertical:
17941           \IgnorePar
17942           \fi:
17943           \EndP
17944           \special
17945           { t4ht* < #1 }
17946           \par
17947           \ShowPar
17948         }
17949         {
17950           \@@_luaxml_print_html:n
17951           { #1 }
17952         }
17953     }
17954 }
17955 {
17956   \@@_plain_tex_default_input_raw_block:nn
17957   { #1 }

```

---

<sup>41</sup>See <https://tug.org/tex4ht/>.

```

17958         { #2 }
17959     }
17960 }

```

### 3.3.5.16 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.

```

17961 \seq_new:N
17962   \l_@@_bracketed_span_identifiers_seq
17963 \markdownSetup {
17964   rendererPrototypes = {
17965     bracketedSpanAttributeContextBegin $= {
17966       \markdownSetup {
17967         rendererPrototypes = {
17968           attributeIdentifier = {
17969             \seq_put_right:Nn
17970               \l_@@_bracketed_span_identifiers_seq
17971               { ##1 }
17972           },
17973         },
17974       }
17975     },
17976     bracketedSpanAttributeContextEnd ^= {
17977       \seq_map_inline:Nn
17978         \l_@@_bracketed_span_identifiers_seq
17979         { \label { ##1 } }
17980     },
17981   },
17982 }

```

We will also allow acronyms to be manually marked up using the HTML class `acronym`.

```

17983 \markdownSetup {
17984   rendererPrototypes = {
17985     bracketedSpanAttributeContextBegin += {
17986       \markdownSetup {
17987         rendererPrototypes = {
17988           attributeClassName = {
17989             \str_if_eq:nnT
17990               { ##1 }
17991               { acronym }
17992             {
17993               \markdownSetup {
17994                 rendererPrototypes = {
17995                   bracketedSpan = {
17996                     \markdownRendererAcronym
17997                       { #####1 }

```

```

17998         }
17999     }
18000 }
18001 }
18002 },
18003 },
18004 }
18005 },
18006 },
18007 }
18008 \ExplSyntaxOff
18009 \fi % Closes ~\markdownIfOption{plain}{\iffalse}{\iftrue}~

```

### 3.3.6 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.

```

18010 \newcommand\markdownMakeOther{%
18011     \count0=128\relax
18012     \loop
18013         \catcode\count0=11\relax
18014         \advance\count0 by 1\relax
18015     \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.

```

18016 \def\markdownMakeOther{%
18017     \count0=128\relax
18018     \loop
18019         \catcode\count0=11\relax
18020         \advance\count0 by 1\relax
18021     \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.

```
18022 \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).

```
18023 \long\def\inputmarkdown{%
18024   \dosingleempty
18025   \doinputmarkdown}%
18026 \long\def\doinputmarkdown[#1]#2{%
18027   \begingroup
18028     \iffirstargument
18029     \setupmarkdown[#1]%
18030     \fi
18031     \markdownInput{#2}%
18032   \endgroup}%
18033 \long\def\inputyaml{%
18034   \dosingleempty
18035   \doinputyaml}%
18036 \long\def\doinputyaml[#1]#2{%
18037   \doinputmarkdown
18038   [jekyllData, expectJekyllData, ensureJekyllData, #1]{#2}}%
```

The `\startmarkdown`, `\stopmarkdown`, `\startyaml`, and `\stopyaml` macros are implemented using the `\markdownReadAndConvert` macro.

In Knuth’s T<sub>E</sub>X, trailing spaces are removed very early on when a line is being put to the input buffer. [21, sec. 31]. According to Eijkhout [22, 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)T<sub>E</sub>X, but ConT<sub>E</sub>Xt MkIV funnels all input through its own input handler. This makes it possible to suppress the removal of trailing spaces in ConT<sub>E</sub>Xt MkIV and therefore to insert hard line breaks into markdown text.

```
18039 \startluacode
18040   document.markdown_buffering = false
18041   local function preserve_trailing_spaces(line)
18042     if document.markdown_buffering then
18043       line = line:gsub("[ \t][ \t]$", "\t\t")
18044     end
18045     return line
18046   end
18047   resolvers.installinputlinehandler(preserve_trailing_spaces)
18048 \stopluacode
18049 \begingroup
18050   \catcode`\|=0%
18051   \catcode`\|=12%
```



```

18052 |gdef|startmarkdown{%
18053   |ctxlua{document.markdown_buffering = true}%
18054   |markdownReadAndConvert{\stopmarkdown}%
18055                               {|stopmarkdown}}%
18056 |gdef|stopmarkdown{%
18057   |ctxlua{document.markdown_buffering = false}%
18058   |markdownEnd}%
18059 |gdef|startyaml{%
18060   |begingroup
18061   |ctxlua{document.markdown_buffering = true}%
18062   |setupyaml[jekyllData, expectJekyllData, ensureJekyllData]%
18063   |markdownReadAndConvert{\stopyaml}%
18064                               {|stopyaml}}%
18065 |gdef|stopyaml{%
18066   |ctxlua{document.markdown_buffering = false}%
18067   |yamlEnd}%
18068 |endgroup

```

### 3.4.2 Themes

This section overrides the plain  $\text{\TeX}$  implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in  $\text{\ConTeXt}$  themes provided with the Markdown package.

```

18069 \ExplSyntaxOn
18070 \prop_new:N \g_@@_context_loaded_themes_linenos_prop
18071 \prop_new:N \g_@@_context_loaded_themes_versions_prop
18072 \cs_gset:Nn
18073   \@@_load_theme:nnn
18074   {

```

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  $\text{\TeX}$  theme instead.

```

18075   \bool_if:nTF
18076     {
18077       \bool_lazy_or_p:nn
18078         {
18079           \prop_if_in_p:Nn
18080             \g_@@_context_built_in_themes_prop
18081             { #1 }
18082         }
18083       {
18084         \file_if_exist_p:n
18085           { t - markdown theme #3.tex }
18086       }

```

```

18087     }
18088   {
18089     \prop_get:NnNTF
18090     \g_@@_context_loaded_themes_linenos_prop
18091     { #1 }
18092     \l_tmpa_tl
18093     {
18094       \prop_get:NnN
18095       \g_@@_context_loaded_themes_versions_prop
18096       { #1 }
18097       \l_tmpb_tl
18098       \str_if_eq:nVTF
18099       { #2 }
18100       \l_tmpb_tl
18101       {
18102         \msg_warning:nnnVn
18103         { markdown }
18104         { repeatedly-loaded-context-theme }
18105         { #1 }
18106         \l_tmpa_tl
18107         { #2 }
18108       }
18109     }
18110     \msg_error:nnnnVV
18111     { markdown }
18112     { different-versions-of-context-theme }
18113     { #1 }
18114     { #2 }
18115     \l_tmpb_tl
18116     \l_tmpa_tl
18117   }
18118 }
18119 {
18120   \prop_gput:Nnx
18121   \g_@@_context_loaded_themes_linenos_prop
18122   { #1 }
18123   { \tex_the:D \tex_inputlineno:D } % noqa: W200
18124   \prop_gput:Nnn
18125   \g_@@_context_loaded_themes_versions_prop
18126   { #1 }
18127   { #2 }

```

Load built-in plain T<sub>E</sub>X themes from the prop `\g_@@_context_built_in_themes_prop` and from the filesystem otherwise.

```

18128   \prop_if_in:NnTF
18129   \g_@@_context_built_in_themes_prop
18130   { #1 }

```

```

18131         {
18132             \msg_info:nnnn
18133             { markdown }
18134             { loading-built-in-context-theme }
18135             { #1 }
18136             { #2 }
18137             \prop_item:Nn
18138             \g_@@_context_built_in_themes_prop
18139             { #1 }
18140         }
18141         {
18142             \msg_info:nnnn
18143             { markdown }
18144             { loading-context-theme }
18145             { #1 }
18146             { #2 }
18147             \usemodule
18148             [ t ]
18149             [ markdown theme #3 ]
18150         }
18151     }
18152 }
18153 {
18154     \@@_plain_tex_load_theme:nnn
18155     { #1 }
18156     { #2 }
18157     { #3 }
18158 }
18159 }
18160 \msg_new:nnn
18161 { markdown }
18162 { loading-built-in-context-theme }
18163 { Loading~version~#2~of~built-in~ConTeXt~Markdown~theme~#1 }
18164 \msg_new:nnn
18165 { markdown }
18166 { loading-context-theme }
18167 { Loading~version~#2~of~ConTeXt~Markdown~theme~#1 }
18168 \msg_new:nnn
18169 { markdown }
18170 { repeatedly-loaded-context-theme }
18171 {
18172     Version~#3~of~ConTeXt~Markdown~theme~#1~was~previously~
18173     loaded~on~line~#2,~not~loading~it~again
18174 }
18175 \msg_new:nnn
18176 { markdown }
18177 { different-versions-of-context-theme }

```

```

18178 {
18179   Tried-to-load-version-#2-of-ConTeXt-Markdown-theme-#1~
18180   but-version-#3-has-already-been-loaded-on-line-#4
18181 }
18182 \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:

```
18183 \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.

```

18184 \markdownIfOption{plain}{\iffalse}{\iftrue}
18185 \def\markdownRendererHardLineBreakPrototype{\blank}%
18186 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
18187 \def\markdownRendererRightBracePrototype{\textbraceright}%
18188 \def\markdownRendererDollarSignPrototype{\textdollar}%
18189 \def\markdownRendererPercentSignPrototype{\percent}%
18190 \def\markdownRendererUnderscorePrototype{\textunderscore}%
18191 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
18192 \def\markdownRendererBackslashPrototype{\textbackslash}%
18193 \def\markdownRendererTildePrototype{\textasciitilde}%
18194 \def\markdownRendererPipePrototype{\char`|}%
18195 \def\markdownRendererLinkPrototype#1#2#3#4{%
18196   \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
18197   \fi\texttt<\hyphenatedurl{#3}>}}%
18198 \usemodule[database]
18199 \defineseparatedlist
18200   [MarkdownConTeXtCSV]
18201   [separator={,},
18202    before=\bTABLE,after=\eTABLE,
18203    first=\bTR,last=\eTR,
18204    left=\bTD,right=\eTD]
18205 \def\markdownConTeXtCSV{csv}
18206 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
18207   \def\markdownConTeXtCSV@arg{#1}%
18208   \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
18209     \placetable[] [tab:#1]{#4}{%
18210       \processseparatedfile[MarkdownConTeXtCSV][#3]}%
18211   \else
18212     \markdownInput{#3}%

```

```

18213 \fi}%
18214 \def\markdownRendererImagePrototype#1#2#3#4{%
18215 \placefigure[] []{#4}{\externalfigure[#3]}}%
18216 \def\markdownRendererUlBeginPrototype{\startitemize}%
18217 \def\markdownRendererUlBeginTightPrototype{\startitemize[packed]}%
18218 \def\markdownRendererUlItemPrototype{\item}%
18219 \def\markdownRendererUlEndPrototype{\stopitemize}%
18220 \def\markdownRendererUlEndTightPrototype{\stopitemize}%
18221 \def\markdownRendererOlBeginPrototype{\startitemize[n]}%
18222 \def\markdownRendererOlBeginTightPrototype{\startitemize[packed,n]}%
18223 \def\markdownRendererOlItemPrototype{\item}%
18224 \def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
18225 \def\markdownRendererOlEndPrototype{\stopitemize}%
18226 \def\markdownRendererOlEndTightPrototype{\stopitemize}%
18227 \definedescription
18228 [MarkdownConTeXtDlItemPrototype]
18229 [location=hanging,
18230 margin=standard,
18231 headstyle=bold]%
18232 \definestartstop
18233 [MarkdownConTeXtDlPrototype]
18234 [before=\blank,
18235 after=\blank]%
18236 \definestartstop
18237 [MarkdownConTeXtDlTightPrototype]
18238 [before=\blank\startpacked,
18239 after=\stoppacked\blank]%
18240 \def\markdownRendererDlBeginPrototype{%
18241 \startMarkdownConTeXtDlPrototype}%
18242 \def\markdownRendererDlBeginTightPrototype{%
18243 \startMarkdownConTeXtDlTightPrototype}%
18244 \def\markdownRendererDlItemPrototype#1{%
18245 \startMarkdownConTeXtDlItemPrototype{#1}}%
18246 \def\markdownRendererDlItemEndPrototype{%
18247 \stopMarkdownConTeXtDlItemPrototype}%
18248 \def\markdownRendererDlEndPrototype{%
18249 \stopMarkdownConTeXtDlPrototype}%
18250 \def\markdownRendererDlEndTightPrototype{%
18251 \stopMarkdownConTeXtDlTightPrototype}%
18252 \def\markdownRendererEmphasisPrototype#1{{\em#1}}%
18253 \def\markdownRendererStrongEmphasisPrototype#1{{\bf#1}}%
18254 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
18255 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
18256 \def\markdownRendererLineBlockBeginPrototype{%
18257 \begingroup
18258 \def\markdownRendererHardLineBreak{
18259 }%

```

```

18260     \startlines
18261 }%
18262 \def\markdownRendererLineBlockEndPrototype{%
18263     \stoplines
18264     \endgroup
18265 }%
18266 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%

```

### 3.4.3.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

18267 \ExplSyntaxOn
18268 \cs_gset:Npn
18269   \markdownRendererInputFencedCodePrototype#1#2#3
18270 {
18271   \tl_if_empty:nTF
18272     { #2 }
18273     { \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 `\definetytyping` macro, which allows the user to set up code highlighting mapping as follows:

```

\definetytyping [latex]
\setuptyping [latex] [option=TEX]

\starttext
  \startmarkdown
~~~ latex
\documentclass{article}
\begin{document}
  Hello world!
\end{document}
~~~
  \stopmarkdown
\stoptext

```

```

18274   {
18275     \regex_extract_once:nnN
18276       { \w* }
18277       { #2 }
18278       \l_tmpa_seq
18279     \seq_pop_left:NN
18280       \l_tmpa_seq
18281       \l_tmpa_tl

```

```

18282         \typefile[ \l_tmpa_tl ][] {#1}
18283     }
18284 }
18285 \ExplSyntaxOff
18286 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
18287 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
18288 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
18289 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
18290 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
18291 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
18292 \def\markdownRendererThematicBreakPrototype{%
18293     \blackrule[height=1pt, width=\hsize]}%
18294 \def\markdownRendererNotePrototype#1{\footnote{#1}}%
18295 \def\markdownRendererTickedBoxPrototype{$\boxtimes$}
18296 \def\markdownRendererHalfTickedBoxPrototype{$\boxdot$}
18297 \def\markdownRendererUntickedBoxPrototype{$\square$}
18298 \def\markdownRendererStrikeThroughPrototype#1{\overstrides{#1}}
18299 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
18300 \def\markdownRendererSubscriptPrototype#1{\low{#1}}
18301 \def\markdownRendererDisplayMathPrototype#1{%
18302     \startformula#1\stopformula}%

```

### 3.4.3.2 Tables

There is a basic implementation of tables.

```

18303 \newcount\markdownConTeXtRowCounter
18304 \newcount\markdownConTeXtRowTotal
18305 \newcount\markdownConTeXtColumnCounter
18306 \newcount\markdownConTeXtColumnTotal
18307 \newtoks\markdownConTeXtTable
18308 \newtoks\markdownConTeXtTableFloat
18309 \def\markdownRendererTablePrototype#1#2#3{%
18310     \markdownConTeXtTable={}%
18311     \ifx\empty#1\empty
18312         \markdownConTeXtTableFloat={%
18313             \the\markdownConTeXtTable}%
18314     \else
18315         \markdownConTeXtTableFloat={%
18316             \placetable{#1}{\the\markdownConTeXtTable}}%
18317     \fi
18318     \begingroup
18319     \setupTABLE[r][each][topframe=off, bottomframe=off,
18320         leftframe=off, rightframe=off]
18321     \setupTABLE[c][each][topframe=off, bottomframe=off,
18322         leftframe=off, rightframe=off]
18323     \setupTABLE[r][1][topframe=on, bottomframe=on]
18324     \setupTABLE[r][#1][bottomframe=on]

```

```

18325 \markdownConTeXtRowCounter=0%
18326 \markdownConTeXtRowTotal=#2%
18327 \markdownConTeXtColumnTotal=#3%
18328 \markdownConTeXtRenderTableRow}
18329 \def\markdownConTeXtRenderTableRow#1{%
18330 \markdownConTeXtColumnCounter=0%
18331 \ifnum\markdownConTeXtRowCounter=0\relax
18332 \markdownConTeXtReadAlignments#1%
18333 \markdownConTeXtTable={\bTABLE}%
18334 \else
18335 \markdownConTeXtTable=\expandafter{%
18336 \the\markdownConTeXtTable\bTR}%
18337 \markdownConTeXtRenderTableCell#1%
18338 \markdownConTeXtTable=\expandafter{%
18339 \the\markdownConTeXtTable\eTR}%
18340 \fi
18341 \advance\markdownConTeXtRowCounter by 1\relax
18342 \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
18343 \markdownConTeXtTable=\expandafter{%
18344 \the\markdownConTeXtTable\eTABLE}%
18345 \the\markdownConTeXtTableFloat
18346 \endgroup
18347 \expandafter\gobbleoneargument
18348 \fi\markdownConTeXtRenderTableRow}
18349 \def\markdownConTeXtReadAlignments#1{%
18350 \advance\markdownConTeXtColumnCounter by 1\relax
18351 \if#1d%
18352 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
18353 \fi\if#1l%
18354 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
18355 \fi\if#1c%
18356 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
18357 \fi\if#1r%
18358 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
18359 \fi
18360 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
18361 \else
18362 \expandafter\gobbleoneargument
18363 \fi\markdownConTeXtReadAlignments}
18364 \def\markdownConTeXtRenderTableCell#1{%
18365 \advance\markdownConTeXtColumnCounter by 1\relax
18366 \markdownConTeXtTable=\expandafter{%
18367 \the\markdownConTeXtTable\bTD#1\eTD}%
18368 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
18369 \else
18370 \expandafter\gobbleoneargument
18371 \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`.

```
18372 \ExplSyntaxOn
18373 \cs_gset:Npn
18374   \markdownRendererInputRawInlinePrototype#1#2
18375   {
18376     \str_case:nnF
18377       { #2 }
18378       {
18379         { latex }
18380         {
18381           \@@_plain_tex_default_input_raw_inline:nn
18382             { #1 }
18383             { context }
18384         }
18385       }
18386     {
18387       \@@_plain_tex_default_input_raw_inline:nn
18388         { #1 }
18389         { #2 }
18390     }
18391   }
18392 \cs_gset:Npn
18393   \markdownRendererInputRawBlockPrototype#1#2
18394   {
18395     \str_case:nnF
18396       { #2 }
18397       {
18398         { context }
18399         {
18400           \@@_plain_tex_default_input_raw_block:nn
18401             { #1 }
18402             { tex }
18403         }
18404       }
18405     {
18406       \@@_plain_tex_default_input_raw_block:nn
18407         { #1 }
18408         { #2 }
18409     }
18410   }
18411 \cs_gset_eq:NN
18412   \markdownRendererInputRawBlockPrototype
18413   \markdownRendererInputRawInlinePrototype
18414 \fi % Closes \markdownIfOption{plain}{\iffalse}{\iftrue}
18415 \ExplSyntaxOff
```

```
18416 \stopmodule
```

```
18417 \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).

```
18418 \ExplSyntaxOn
```

```
18419 \str_if_eq:VVT
```

```
18420 \c_@@_top_layer_tl
```

```
18421 \c_@@_option_layer_context_tl
```

```
18422 {
```

```
18423 \use:c
```

```
18424 { ExplSyntaxOff }
```

```
18425 \@@_if_option:nF
```

```
18426 { noDefaults }
```

```
18427 {
```

```
18428 \@@_if_option:nTF
```

```
18429 { experimental }
```

```
18430 {
```

```
18431 \@@_setup:n
```

```
18432 { theme = witiko/markdown/defaults@experimental }
```

```
18433 }
```

```
18434 {
```

```
18435 \@@_setup:n
```

```
18436 { theme = witiko/markdown/defaults }
```

```
18437 }
```

```
18438 }
```

```
18439 \use:c
```

```
18440 { ExplSyntaxOn }
```

```
18441 }
```

```
18442 \ExplSyntaxOff
```

```
18443 \stopmodule
```

```
18444 \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 2<sub>ε</sub> Sources*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/source2e.pdf> (visited on 01/08/2018).
- [20] L<sup>A</sup>T<sub>E</sub>X Project. *The L<sup>A</sup>T<sub>E</sub>X 3 Interfaces*. Jan. 19, 2026. URL: <https://mirrors.ctan.org/macros/latex/required/l3kernel/interface3.pdf> (visited on 02/18/2026).
- [21] 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.
- [22] 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

acronyms	23, 93, 171, 475, 477, 478
autoIdentifiers	23, 36, 94, 110
blankBeforeBlockquote	24
blankBeforeCodeFence	24
blankBeforeDivFence	24
blankBeforeHeading	25
blankBeforeHtmlBlock	25
blankBeforeList	25
bracketedSpans	26, 96, 510
breakableBlockquotes	26
cacheDir	4, 18, 21, 63, 169, 201, 414, 436, 456
citationNbsps	26
citations	27, 99, 100
codeSpans	27
contentBlocks	22, 28, 38
contentBlocksLanguageMap	22
contentLevel	28
debugExtensions	9, 22, 29, 355
debugExtensionsFileName	22, 29
defaultOptions	10, 55, 411, 413

definitionLists	29, 104
depth_first_search	178
eagerCache	18, 411
ensureJekyllData	30, 30, 31, 40
entities.char_entity	255
entities.dec_entity	255
entities.hex_entity	255
entities.hex_entity_with_x_char	255
escape_minimal	259
escape_programmatic_text	259
escape_typographic_text	259
expandtabs	315
expectJekyllData	30, 30, 31, 40
experimental	5, 19, 37, 480
experimentalOptions	10, 411, 413
extensions	32, 177, 360
extensions.acronyms	179, 360
extensions.citations	363
extensions.content_blocks	367
extensions.definition_lists	371
extensions.fancy_lists	373
extensions.fenced_code	379
extensions.fenced_divs	384
extensions.header_attributes	388
extensions.inline_code_attributes	390
extensions.jekyll_data	407
extensions.line_blocks	390
extensions.link_attributes	392
extensions.mark	391
extensions.notes	393
extensions.pipe_table	396
extensions.raw_inline	400
extensions.strike_through	401
extensions.subscripts	402
extensions.superscripts	402
extensions.tex_math	403
fancyLists	33, 121–126, 480
fencedCode	33, 43, 101, 108, 127, 425, 428
fencedCodeAttributes	34, 94, 108, 428
fencedDivs	34, 45, 109

finalizeCache	18, 19, 23, 35, 35, 63, 169, 411, 413
frozenCache	23, 35, 63, 85, 86, 169, 426, 436
frozenCacheCounter	35, 413, 464, 465
frozenCacheFileName	23, 35, 63, 413
\g_markdown_diagrams_infostrings_prop	430
gfmAutoIdentifiers	24, 36, 94, 110
hashEnumerators	36
headerAttributes	36, 45, 94, 110
html	37, 37, 113, 493
htmlOverLinks	19, 37
hybrid	38, 38, 44, 50, 52, 66, 86, 128, 169, 260, 316, 464
inlineCodeAttributes	39, 94, 102
inlineNotes	39
\inputmarkdown	172, 174, 175, 512
inputTempFileName	64, 67, 458, 459, 461, 462
\inputyaml	30, 31, 40, 172, 174, 512
iterlines	315
jeekyllData	3, 30, 31, 40, 40, 137–140, 143
languages_json	368, 368
lineBlocks	41, 116
linkAttributes	41, 94, 114, 118, 336, 506
mark	42, 119, 504
\markdown	164, 164, 165, 166, 467
markdown	163, 164, 165, 466, 467
markdown*	163, 164, 168, 466
\markdownBegin	57, 58–60, 162, 164, 165, 173
\markdownCleanup	457
\markdownConvert	456
\markdownEnd	57, 58–60, 162, 164–166, 173
\markdownError	161, 161, 162
\markdownEscape	57, 61, 465
\markdownIfOption	62
\markdownIfSnippetExists	87
\markdownInfo	161, 162
\markdownInput	57, 60, 163, 166, 168, 174, 463, 466
\markdownInputFilename	456
\markdownInputFileStream	457

\markdownInputPlainTeX	466
\markdownLoadPlainTeXTheme	170, 176, 424
\markdownLuaExecute	460, 463
\markdownLuaOptions	453, 456
\markdownMakeOther	162, 511
\markdownOptionExperimental	67
\markdownOptionFinalizeCache	63
\markdownOptionFrozenCache	63
\markdownOptionInputTempFileName	64
\markdownOptionNoDefaults	65
\markdownOptionOutputDir	64, 64, 68, 69
\markdownOptionPlain	65
\markdownOptionStripPercentSigns	66
\markdownOutputFileStream	457
\markdownPrepare	456
\markdownPrepareInputFilename	456
\markdownPrepareLuaOptions	453
\markdownReadAndConvert	162, 457, 466, 467, 512
\markdownReadAndConvertProcessLine	458, 459
\markdownReadAndConvertStripPercentSigns	458
\markdownReadAndConvertTab	457
\markdownRendererAcronym	93
\markdownRendererAttributeClassName	94
\markdownRendererAttributeIdentifier	94
\markdownRendererAttributeKeyValue	94
\markdownRendererBlockQuoteBegin	95
\markdownRendererBlockQuoteEnd	95
\markdownRendererBracketedSpan	96
\markdownRendererBracketedSpanAttributeContextBegin	96
\markdownRendererBracketedSpanAttributeContextEnd	96
\markdownRendererCite	99, 100
\markdownRendererCodeSpan	101
\markdownRendererCodeSpanAttributeContextBegin	102
\markdownRendererCodeSpanAttributeContextEnd	102
\markdownRendererContentBlock	102, 103
\markdownRendererContentBlockCode	103
\markdownRendererContentBlockOnlineImage	103
\markdownRendererDisplayMath	134
\markdownRendererDlBegin	104
\markdownRendererDlBeginTight	104
\markdownRendererDlDefinitionBegin	105
\markdownRendererDlDefinitionEnd	106

<code>\markdownRendererDlEnd</code>	106
<code>\markdownRendererDlEndTight</code>	107
<code>\markdownRendererDlItem</code>	105
<code>\markdownRendererDlItemEnd</code>	105
<code>\markdownRendererDocumentBegin</code>	119
<code>\markdownRendererDocumentEnd</code>	119
<code>\markdownRendererEllipsis</code>	45, 107
<code>\markdownRendererEmphasis</code>	107, 147
<code>\markdownRendererError</code>	136
<code>\markdownRendererFancyOlBegin</code>	121, 122
<code>\markdownRendererFancyOlBeginTight</code>	122
<code>\markdownRendererFancyOlEnd</code>	126
<code>\markdownRendererFancyOlEndTight</code>	126
<code>\markdownRendererFancyOlItem</code>	124
<code>\markdownRendererFancyOlItemEnd</code>	124
<code>\markdownRendererFancyOlItemWithNumber</code>	124
<code>\markdownRendererFencedCodeAttributeContextBegin</code>	108
<code>\markdownRendererFencedCodeAttributeContextEnd</code>	108
<code>\markdownRendererFencedDivAttributeContextBegin</code>	109
<code>\markdownRendererFencedDivAttributeContextEnd</code>	109
<code>\markdownRendererHalfTickBox</code>	135
<code>\markdownRendererHardLineBreak</code>	117
<code>\markdownRendererHeaderAttributeContextBegin</code>	110
<code>\markdownRendererHeaderAttributeContextEnd</code>	110
<code>\markdownRendererHeadingFive</code>	112
<code>\markdownRendererHeadingFour</code>	112
<code>\markdownRendererHeadingOne</code>	110
<code>\markdownRendererHeadingSix</code>	112
<code>\markdownRendererHeadingThree</code>	111
<code>\markdownRendererHeadingTwo</code>	111
<code>\markdownRendererImage</code>	114
<code>\markdownRendererImageAttributeContextBegin</code>	114
<code>\markdownRendererImageAttributeContextEnd</code>	114
<code>\markdownRendererInlineHtmlComment</code>	113
<code>\markdownRendererInlineHtmlTag</code>	113
<code>\markdownRendererInlineMath</code>	134
<code>\markdownRendererInputBlockHtmlElement</code>	113
<code>\markdownRendererInputFencedCode</code>	101
<code>\markdownRendererInputRawBlock</code>	127
<code>\markdownRendererInputRawInline</code>	126
<code>\markdownRendererInputVerbatim</code>	100
<code>\markdownRendererInterblockSeparator</code>	115



<code>\markdownRendererJekyllDataBegin</code>	137
<code>\markdownRendererJekyllDataBoolean</code>	139
<code>\markdownRendererJekyllDataEmpty</code>	143
<code>\markdownRendererJekyllDataEnd</code>	137
<code>\markdownRendererJekyllDataMappingBegin</code>	138
<code>\markdownRendererJekyllDataMappingEnd</code>	138
<code>\markdownRendererJekyllDataNumber</code>	140
<code>\markdownRendererJekyllDataProgrammaticString</code>	140, 140, 141
<code>\markdownRendererJekyllDataSequenceBegin</code>	139
<code>\markdownRendererJekyllDataSequenceEnd</code>	139
<code>\markdownRendererJekyllDataString</code>	141, 141, 145
<code>\markdownRendererJekyllDataStringPrototype</code>	157
<code>\markdownRendererJekyllDataTypographicString</code>	140, 140, 141, 408
<code>\markdownRendererLineBlockBegin</code>	116
<code>\markdownRendererLineBlockEnd</code>	116
<code>\markdownRendererLink</code>	117, 147
<code>\markdownRendererLinkAttributeContextBegin</code>	118
<code>\markdownRendererLinkAttributeContextEnd</code>	118
<code>\markdownRendererMark</code>	119
<code>\markdownRendererNbsp</code>	120
<code>\markdownRendererNote</code>	120
<code>\markdownRendererOlBegin</code>	121
<code>\markdownRendererOlBeginTight</code>	121
<code>\markdownRendererOlEnd</code>	125
<code>\markdownRendererOlEndTight</code>	125
<code>\markdownRendererOlItem</code>	46, 122
<code>\markdownRendererOlItemEnd</code>	123
<code>\markdownRendererOlItemWithNumber</code>	46, 123
<code>\markdownRendererParagraphSeparator</code>	116
<code>\markdownRendererReplacementCharacter</code>	128
<code>\markdownRendererSectionBegin</code>	127
<code>\markdownRendererSectionEnd</code>	127
<code>\markdownRendererSoftLineBreak</code>	117
<code>\markdownRendererStrikeThrough</code>	131
<code>\markdownRendererStrongEmphasis</code>	108
<code>\markdownRendererSubscript</code>	132
<code>\markdownRendererSuperscript</code>	132
<code>\markdownRendererTable</code>	133
<code>\markdownRendererTableAttributeContextBegin</code>	133
<code>\markdownRendererTableAttributeContextEnd</code>	133
<code>\markdownRendererTextCite</code>	100
<code>\markdownRendererThematicBreak</code>	135

<code>\markdownRendererTickedBox</code>	135
<code>\markdownRendererULBegin</code>	97
<code>\markdownRendererULBeginTight</code>	97
<code>\markdownRendererULEnd</code>	99
<code>\markdownRendererULEndTight</code>	99
<code>\markdownRendererULItem</code>	98
<code>\markdownRendererULItemEnd</code>	98
<code>\markdownRendererUntickedBox</code>	135
<code>\markdownRendererWarning</code>	136
<code>\markdownSetup</code>	62, 62, 66, 168, 175, 467, 477
<code>\markdownSetupSnippet</code>	86, 86
<code>\markdownThemeVersion</code>	76, 76
<code>\markdownWarning</code>	161, 162
<code>\markinline</code>	57, 59, 163, 166, 461, 465
<code>\markinlinePlainTeX</code>	465
<code>\mmdcCommand</code>	432
<code>new</code>	7, 19, 20, 411, 413
<code>notes</code>	42, 120
<code>parsers</code>	277, 315
<code>parsers.commented_line</code>	295
<code>parsers.unicode</code>	279
<code>pipeTables</code>	7, 43, 49, 133
<code>preserveTabs</code>	43, 47, 315
<code>rawAttribute</code>	38, 43, 44, 126, 127
<code>read_decompositions</code>	180
<code>reader</code>	8, 33, 177, 277, 314, 360
<code>reader-&gt;add_special_character</code>	8, 9, 33, 354
<code>reader-&gt;auto_link_email</code>	343
<code>reader-&gt;auto_link_url</code>	343
<code>reader-&gt;create_parser</code>	316
<code>reader-&gt;finalize_grammar</code>	350, 418
<code>reader-&gt;initialize_named_group</code>	354
<code>reader-&gt;insert_pattern</code>	8, 9, 33, 350, 356
<code>reader-&gt;lookup_note_reference</code>	328
<code>reader-&gt;lookup_reference</code>	328
<code>reader-&gt;normalize_tag</code>	315
<code>reader-&gt;options</code>	314
<code>reader-&gt;parser_functions</code>	316
<code>reader-&gt;parser_functions.name</code>	316
<code>reader-&gt;parsers</code>	315, 315

reader->register_link	328
reader->update_rule	350, 353, 356
reader->writer	314
reader.new	314, 314, 418
relativeReferences	37, 44
serialize_byte_parser	179
serialize_byte_range_parser	179
serialize_replacement	179
\setupmarkdown	175, 175
\setupyaml	175
shiftHeadings	7, 45
singletonCache	19
slice	7, 45, 256, 269, 270
smartEllipses	45, 107, 169
\startmarkdown	172, 173, 512
startNumber	46, 122–124
\startyaml	30, 31, 40, 172, 173, 512
\stopmarkdown	172, 173, 512
\stopyaml	30, 31, 40, 172, 173, 512
strikeThrough	46, 131, 505
stripIndent	47, 316
stripPercentSigns	457, 458
subscripts	47, 132
superscripts	48, 132
syntax	351, 356
tableAttributes	48, 133, 501
tableCaptions	7, 48, 49, 133
taskLists	49, 135, 492
texComments	50, 316
texMathDollars	39, 50, 134
texMathDoubleBackslash	39, 51, 134
texMathSingleBackslash	39, 51, 134
tightLists	51, 97, 99, 104, 107, 121, 122, 125, 126, 480
underscores	52
unicode_data.casefold_mapping	190, 201
unicode_data.categories	192, 279
unicode_data.ccc	194, 202
unicode_data.composition_mapping	186, 206
unicode_data.decomposition_mapping	181, 182, 205
unicodeNormalization	20, 20, 21

unicodeNormalizationForm	20, 20
util.cache	196, 197
util.cache_verbatim	197
util.canonically_order	202
util.casefold	201
util.compose	206
util.decompose	205
util.encode_json_string	197
util.err	196
util.escaper	200
util.expand_tabs_in_line	197
util.find_file	209
util.find_files	209
util.flatten	198
util.intersperse	199
util.map	199
util.normalize	209
util.pathname	200
util.rope_last	199
util.rope_to_string	198
util.salt	201
util.table_copy	197
util.walk	198, 198, 199
util.warning	201
walkable_syntax	8, 22, 29, 350, 353–356
writer	177, 177, 256, 360
writer->acronym	360
writer->active_attributes	268, 268, 269, 270
writer->attribute_type_levels	268
writer->attributes	266
writer->block_html_element	264
writer->blockquote	265
writer->bulletitem	263
writer->bulletlist	262
writer->citations	363
writer->code	261
writer->contentblock	368
writer->defer_call	276, 276
writer->definitionlist	371
writer->display_math	403
writer->div_begin	384

writer->div_end	384
writer->document	265
writer->ellipsis	258
writer->emphasis	264
writer->error	261
writer->escape	260
writer->escaped_chars	259, 259
writer->escaped_minimal_strings	259, 259
writer->escaped_strings	259
writer->escaped_uri_chars	259, 259
writer->fancyitem	374
writer->fancylist	373
writer->fencedCode	379
writer->flatten_inlines	256, 256
writer->get_state	276
writer->hard_line_break	258
writer->heading	274
writer->identifier	260
writer->image	262
writer->infostring	260
writer->inline_html_comment	264
writer->inline_html_tag	264
writer->inline_math	404
writer->interblocksep	258
writer->is_writing	256, 256
writer->jekyllData	407
writer->lineblock	390
writer->link	261
writer->mark	391
writer->math	260
writer->nbsp	257
writer->note	394
writer->options	256
writer->ordereditem	263
writer->orderedlist	263
writer->paragraph	257
writer->paragraphsep	258
writer->plain	257
writer->pop_attributes	269, 269, 270
writer->push_attributes	269, 269, 270
writer->rawBlock	380
writer->rawInline	401

writer->set_state	276
writer->slice_begin	256
writer->slice_end	256
writer->soft_line_break	258
writer->space	257
writer->span	362
writer->strike_through	401
writer->string	260
writer->strong	264
writer->subscript	402
writer->superscript	403
writer->table	397
writer->thematic_break	258
writer->checkbox	265
writer->undosep	258, 359
writer->uri	260
writer->verbatim	265
writer->warning	201, 260
writer.new	256, 256, 418
\yaml	166
yaml	30, 31, 40, 163, 165, 166, 466
\yamlBegin	30, 31, 40, 57, 58, 59, 162, 165, 173
\yamlEnd	30, 31, 40, 57, 58, 59, 162, 165, 166, 173
\yamlInput	30, 31, 40, 57, 60, 163, 166, 174, 466
\yamlSetup	62