

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

Vít Starý Novotný  
witiko@mail.muni.cz

Version 3.4.2-0-ga45cf0ed  
2024-03-09

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>	<b>3</b>	<b>Implementation</b>	<b>134</b>
1.1	Requirements . . . . .	2	3.1	Lua Implementation . . .	134
1.2	Feedback . . . . .	6	3.2	Plain T <sub>E</sub> X Implementation	327
1.3	Acknowledgements . . . .	6	3.3	L <sup>A</sup> T <sub>E</sub> X Implementation . .	346
<b>2</b>	<b>Interfaces</b>	<b>6</b>	3.4	ConT <sub>E</sub> Xt Implementation	376
2.1	Lua Interface . . . . .	7			
2.2	Plain T <sub>E</sub> X Interface . . . .	49		<b>References</b>	<b>384</b>
2.3	L <sup>A</sup> T <sub>E</sub> X Interface . . . . .	124			
2.4	ConT <sub>E</sub> Xt Interface . . . .	131		<b>Index</b>	<b>385</b>

## List of Figures

1	A block diagram of the Markdown package . . . . .	7
2	A sequence diagram of typesetting a document using the T <sub>E</sub> X interface . .	45
3	A sequence diagram of typesetting a document using the Lua CLI . . . . .	46
4	Various formats of mathematical formulae . . . . .	129
5	The banner of the Markdown package . . . . .	130

## 1 Introduction

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

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

---

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

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

number of tutorials and code examples. You can find more of these in the user manual.<sup>3</sup>

```
1 local metadata = {
2   version   = "(((VERSION)))",
3   comment   = "A module for the conversion from markdown to plain TeX",
4   author    = "John MacFarlane, Hans Hagen, Vít Starý Novotný",
5   copyright = {"2009-2016 John MacFarlane, Hans Hagen",
6               "2016-2023 Vít Starý Novotný"},
7   license   = "LPPL 1.3c"
8 }
9
10 if not modules then modules = { } end
11 modules['markdown'] = metadata
```

## 1.1 Requirements

This section gives an overview of all resources required by the package.

### 1.1.1 Lua Requirements

The Lua part of the package requires that the following Lua modules are available from within the LuaTeX engine (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  $\geq 2013$ ).

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

**Selene Unicode** A library that provides support for the processing of wide strings. It is used by the Lunamark library to cast image, link, and note tags to the lower case. Selene Unicode is included in all releases of LuaTeX (TeXLive  $\geq 2008$ ).

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

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

```
14 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>.

```
15 (function()
```

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

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

All the abovelisted modules are statically linked into the current version of the LuaTeX engine [1, Section 4.3]. Beside these, we also include the following third-party Lua libraries:

**lua-uni-algos** A package that implements Unicode case-folding in TeX Live  $\geq$  2020.

```
23 local uni_algos = require("lua-uni-algos")
```

**api7/lua-tinyyaml** A library that provides a regex-based recursive descent YAML (subset) parser that is used to read YAML metadata when the `jeekyllData` option is enabled. We carry a copy of the library in file `markdown-tinyyaml.lua` distributed together with the Markdown package.

### 1.1.2 Plain TeX Requirements

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

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

```
24 </tex>
25 <*context>
26 \unprotect
27 </context>
28 <*context, tex>
29 \ifx\ExplSyntaxOn\undefined
30   \input expl3-generic
31 \fi
32 </context, tex>
33 <*tex>
```

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

The plain TeX part of the package also requires the following Lua module:

**Lua File System** A library that provides access to the filesystem via OS-specific syscalls. It is used by the plain TeX code to create the cache directory specified by the **cacheDir** option before interfacing with the Lunamark library. Lua File System is included in all releases of LuaTeX (TeXLive  $\geq$  2008).

The plain TeX code makes use of the **isdir** method that was added to the Lua File System library by the LuaTeX engine developers [1, Section 4.2.4].

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

Unless you convert markdown documents to TeX manually using the Lua command-line interface (see Section 2.1.7), the plain TeX part of the package will require that either the LuaTeX **\directlua** primitive or the shell access file stream 18 is available in your TeX engine. If only the shell access file stream is available in your TeX engine (as is the case with pdfTeX and XeTeX), then unless your TeX engine is globally configured to enable shell access, you will need to provide the **-shell-escape** parameter to your engine when typesetting a document.

### 1.1.3 L<sup>A</sup>TeX Requirements

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

```
34 \NeedsTeXFormat{LaTeX2e}%
```

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

The following packages are soft prerequisites. They are only used to provide default token renderer prototypes (see sections 2.2.6 and 3.3.5) or L<sup>A</sup>TeX themes (see Section 2.3.3) and will not be loaded if the option **plain** has been enabled (see Section 2.2.2.3):

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

**graphicx** A package that provides the **\includegraphics** macro for the typesetting of images.

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

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

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

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

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

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

**catchfile** A package that catches the contents of a file and puts it in a macro. It is used in the [witiko/graphicx/http](#) L<sup>A</sup>T<sub>E</sub>X theme, see Section 2.3.3.

**graphicx** A package that builds upon the graphics package, which is part of the L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> kernel. It provides a key-value interface that is used in the default renderer prototypes for image attribute contexts.

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

**etoolbox** A package that is used to polyfill the general hook management system in the default renderer prototypes for YAML metadata, see Section 3.3.5.8, and also in the default renderer prototype for identifier attributes.

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

**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.

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

35 `\RequirePackage{expl3}`

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

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

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

### 1.2 Feedback

Please use the Markdown project page on GitHub<sup>4</sup> to report bugs and submit feature requests. If you do not want to report a bug or request a feature but are simply in need of assistance, you might want to consider posting your question to the T<sub>E</sub>X- $\LaTeX$  Stack Exchange.<sup>5</sup> community question answering web site under the `markdown` tag.

### 1.3 Acknowledgements

The Lunamark Lua module provides speedy markdown parsing for the package. I would like to thank John Macfarlane, the creator of Lunamark, for releasing Lunamark under a permissive license, which enabled its use in the Markdown package.

Extensive user documentation for the Markdown package was kindly written by Lian Tze Lim and published by Overleaf.

Funding by the Faculty of Informatics at the Masaryk University in Brno [2] is gratefully acknowledged.

Support for content slicing (Lua options `shiftHeadings` and `slice`) and pipe tables (Lua options `pipeTables` and `tableCaptions`) was graciously sponsored by David Vins and Omedym.

The T<sub>E</sub>X implementation of the package draws inspiration from several sources including the source code of  $\LaTeX_{2\epsilon}$ , 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

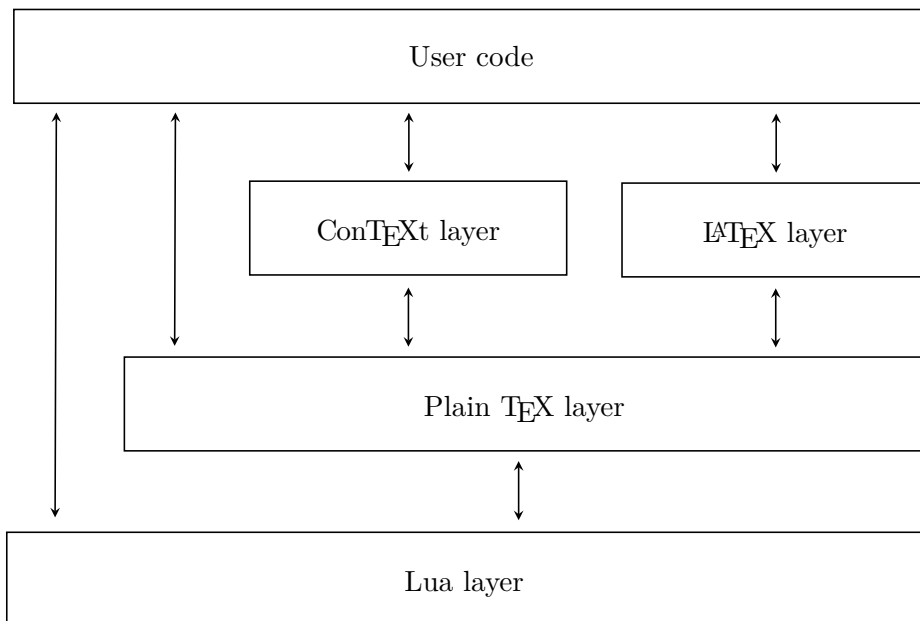
---

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

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

structuring this documentation and as a promise to the user that if they only access the package through the interface, the future minor versions of the package should remain backwards compatible.

Figure 1 shows the high-level structure of the Markdown package: The translation from markdown to  $\text{\TeX}$  *token renderers* is exposed by the Lua layer. The plain  $\text{\TeX}$  layer exposes the conversion capabilities of Lua as  $\text{\TeX}$  macros. The  $\text{\LaTeX}$  and  $\text{\ConTeXt}$  layers provide syntactic sugar on top of plain  $\text{\TeX}$  macros. The user can interface with any and all layers.



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

## 2.1 Lua Interface

The Lua interface provides the conversion from UTF-8 encoded markdown to plain  $\text{\TeX}$ . This interface is used by the plain  $\text{\TeX}$  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.

```
36 local M = {metadata = metadata}
```

### 2.1.1 Conversion from Markdown to Plain $\text{\TeX}$

The Lua interface exposes the `new(options)` function. This function returns a conversion function from markdown to plain  $\text{\TeX}$  according to the table `options` that contains options recognized by the Lua interface (see Section 2.1.3). The

`options` parameter is optional; when unspecified, the behaviour will be the same as if `options` were an empty table.

The following example Lua code converts the markdown string `Hello *world*!` to a  $\text{\TeX}$  output using the default options and prints the  $\text{\TeX}$  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.

```
37 local walkable_syntax = {
38   Block = {
39     "Blockquote",
40     "Verbatim",
41     "ThematicBreak",
42     "BulletList",
43     "OrderedList",
44     "DisplayHtml",
45     "Heading",
46   },
47   BlockOrParagraph = {
48     "Block",
49     "Paragraph",
50     "Plain",
51   },
52   Inline = {
53     "Str",
54     "Space",
55     "Endline",
56     "EndlineBreak",
57     "LinkAndEmph",
58     "Code",
59     "AutoLinkUrl",
60     "AutoLinkEmail",
61     "AutoLinkRelativeReference",
```



```

62     "InlineHtml",
63     "HtmlEntity",
64     "EscapedChar",
65     "Smart",
66     "Symbol",
67 },
68 }

```

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

For example. if we'd like to insert `pattern` into the grammar between the `Inline -> LinkAndEmph` and `Inline -> Code` rules, we would call `reader->insert_pattern` with "`Inline after LinkAndEmph`" (or "`Inline before Code`") and `pattern` as the arguments.

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

### 2.1.3 Options

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

```

69 local defaultOptions = {}

```

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

```

70 \ExplSyntaxOn
71 \seq_new:N \g_@@_lua_options_seq

```

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

```

72 \prop_new:N \g_@@_lua_option_types_prop
73 \prop_new:N \g_@@_default_lua_options_prop
74 \seq_new:N \g_@@_option_layers_seq
75 \tl_const:Nn \c_@@_option_layer_lua_tl { lua }
76 \seq_gput_right:NV \g_@@_option_layers_seq \c_@@_option_layer_lua_tl
77 \cs_new:Nn
78   \@@_add_lua_option:nnn
79   {
80     \@@_add_option:Vnnn

```

```

81     \c_@@_option_layer_lua_tl
82     { #1 }
83     { #2 }
84     { #3 }
85 }
86 \cs_new:Nn
87   \@@_add_option:nnnn
88   {
89     \seq_gput_right:cn
90     { g_@@_ #1 _options_seq }
91     { #2 }
92     \prop_gput:cnn
93     { g_@@_ #1 _option_types_prop }
94     { #2 }
95     { #3 }
96     \prop_gput:cnn
97     { g_@@_default_ #1 _options_prop }
98     { #2 }
99     { #4 }
100   \@@_typecheck_option:n
101     { #2 }
102   }
103 \cs_generate_variant:Nn
104   \@@_add_option:nnnn
105   { Vnnn }
106 \tl_const:Nn \c_@@_option_value_true_tl { true }
107 \tl_const:Nn \c_@@_option_value_false_tl { false }
108 \cs_new:Nn \@@_typecheck_option:n
109   {
110     \@@_get_option_type:nN
111     { #1 }
112     \l_tmpa_tl
113     \str_case_e:Vn
114     \l_tmpa_tl
115     {
116       { \c_@@_option_type_boolean_tl }
117       {
118         \@@_get_option_value:nN
119         { #1 }
120         \l_tmpa_tl
121         \bool_if:nF
122         {
123           \str_if_eq_p:VV
124           \l_tmpa_tl
125           \c_@@_option_value_true_tl ||
126           \str_if_eq_p:VV
127           \l_tmpa_tl

```

```

128         \c_@@_option_value_false_tl
129     }
130     {
131         \msg_error:nnnV
132         { markdown }
133         { failed-typecheck-for-boolean-option }
134         { #1 }
135         \l_tmpa_tl
136     }
137 }
138 }
139 }
140 \msg_new:nnn
141 { markdown }
142 { failed-typecheck-for-boolean-option }
143 {
144     Option~#1~has~value~#2,~
145     but~a~boolean~(true~or~false)~was~expected.
146 }
147 \cs_generate_variant:Nn
148   \str_case_e:nn
149   { Vn }
150 \cs_generate_variant:Nn
151   \msg_error:nnnn
152   { nnnV }
153 \seq_new:N \g_@@_option_types_seq
154 \tl_const:Nn \c_@@_option_type_clist_tl { clist }
155 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_clist_tl
156 \tl_const:Nn \c_@@_option_type_counter_tl { counter }
157 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_counter_tl
158 \tl_const:Nn \c_@@_option_type_boolean_tl { boolean }
159 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_boolean_tl
160 \tl_const:Nn \c_@@_option_type_number_tl { number }
161 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_number_tl
162 \tl_const:Nn \c_@@_option_type_path_tl { path }
163 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_path_tl
164 \tl_const:Nn \c_@@_option_type_slice_tl { slice }
165 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_slice_tl
166 \tl_const:Nn \c_@@_option_type_string_tl { string }
167 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_string_tl
168 \cs_new:Nn
169   \@@_get_option_type:nN
170   {
171     \bool_set_false:N
172       \l_tmpa_bool
173     \seq_map_inline:Nn
174       \g_@@_option_layers_seq

```

```

175     {
176         \prop_get:cnNT
177         { g_@@_ ##1 _option_types_prop }
178         { #1 }
179         \l_tmpa_tl
180         {
181             \bool_set_true:N
182             \l_tmpa_bool
183             \seq_map_break:
184         }
185     }
186     \bool_if:nF
187     \l_tmpa_bool
188     {
189         \msg_error:nnn
190         { markdown }
191         { undefined-option }
192         { #1 }
193     }
194     \seq_if_in:NVF
195     \g_@@_option_types_seq
196     \l_tmpa_tl
197     {
198         \msg_error:nnnV
199         { markdown }
200         { unknown-option-type }
201         { #1 }
202         \l_tmpa_tl
203     }
204     \tl_set_eq:NN
205     #2
206     \l_tmpa_tl
207 }
208 \msg_new:nnn
209 { markdown }
210 { unknown-option-type }
211 {
212     Option~#1~has~unknown~type~#2.
213 }
214 \msg_new:nnn
215 { markdown }
216 { undefined-option }
217 {
218     Option~#1~is~undefined.
219 }
220 \cs_new:Nn
221 \@@_get_default_option_value:nN

```

```

222 {
223   \bool_set_false:N
224   \l_tmpa_bool
225   \seq_map_inline:Nn
226   \g_@@_option_layers_seq
227   {
228     \prop_get:cnNT
229     { g_@@_default_ ##1 _options_prop }
230     { #1 }
231     #2
232     {
233       \bool_set_true:N
234       \l_tmpa_bool
235       \seq_map_break:
236     }
237   }
238   \bool_if:nF
239   \l_tmpa_bool
240   {
241     \msg_error:nnn
242     { markdown }
243     { undefined-option }
244     { #1 }
245   }
246 }
247 \cs_new:Nn
248 \@@_get_option_value:nN
249 {
250   \@@_option_tl_to_csname:nN
251   { #1 }
252   \l_tmpa_tl
253   \cs_if_free:cTF
254   { \l_tmpa_tl }
255   {
256     \@@_get_default_option_value:nN
257     { #1 }
258     #2
259   }
260   {
261     \@@_get_option_type:nN
262     { #1 }
263     \l_tmpa_tl
264     \str_if_eq:NNTF
265     \c_@@_option_type_counter_tl
266     \l_tmpa_tl
267     {
268       \@@_option_tl_to_csname:nN

```

```

269         { #1 }
270         \l_tmpa_tl
271         \tl_set:Nx
272         #2
273         { \the \cs:w \l_tmpa_tl \cs_end: }
274     }
275     {
276         \@@_option_tl_to_csname:nN
277         { #1 }
278         \l_tmpa_tl
279         \tl_set:Nv
280         #2
281         { \l_tmpa_tl }
282     }
283 }
284 }
285 \cs_new:Nn \@@_option_tl_to_csname:nN
286 {
287     \tl_set:Nn
288     \l_tmpa_tl
289     { \str_uppercase:n { #1 } }
290     \tl_set:Nx
291     #2
292     {
293         markdownOption
294         \tl_head:f { \l_tmpa_tl }
295         \tl_tail:n { #1 }
296     }
297 }

```

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.

```

298 \cs_new:Nn \@@_with_various_cases:nn
299 {
300     \seq_clear:N
301     \l_tmpa_seq
302     \seq_map_inline:Nn
303     \g_@@_cases_seq
304     {
305         \tl_set:Nn
306         \l_tmpa_tl
307         { #1 }
308         \use:c { ##1 }
309         \l_tmpa_tl
310         \seq_put_right:NV
311         \l_tmpa_seq

```

```

312         \l_tmpa_tl
313     }
314     \seq_map_inline:Nn
315         \l_tmpa_seq
316         { #2 }
317 }

```

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

```

318 \cs_new:Nn \@@_with_various_cases_break:
319 {
320     \seq_map_break:
321 }

```

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

```

322 \seq_new:N \g_@@_cases_seq
323 \cs_new:Nn \@@_camel_case:N
324 {
325     \regex_replace_all:nnN
326         { _ ([a-z]) }
327         { \c { str_uppercase:n } \cB\{ \1 \cE\} }
328         #1
329     \tl_set:Nx
330         #1
331         { #1 }
332 }
333 \seq_gput_right:Nn \g_@@_cases_seq { @@_camel_case:N }
334 \cs_new:Nn \@@_snake_case:N
335 {
336     \regex_replace_all:nnN
337         { ([a-z])([A-Z]) }
338         { \1 _ \c { str_lowercase:n } \cB\{ \2 \cE\} }
339         #1
340     \tl_set:Nx
341         #1
342         { #1 }
343 }
344 \seq_gput_right:Nn \g_@@_cases_seq { @@_snake_case:N }

```

#### 2.1.4 General Behavior

`eagerCache=true, false` default: `false`

**true**      Converted markdown documents will be cached in `cacheDir`. This can be useful for post-processing the converted documents and for recovering historical versions of the documents from the cache. However, it also

produces a large number of auxiliary files on the disk and obscures the output of the Lua command-line interface when it is used for plumbing.

This behavior will always be used if the `finalizeCache` option is enabled.

**false**      Converted markdown documents will not be cached. This decreases the number of auxiliary files that we produce and makes it easier to use the Lua command-line interface for plumbing.

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

```
345 \@@_add_lua_option:nnn
346   { eagerCache }
347   { boolean }
348   { false }

349 defaultOptions.eagerCache = 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 issue #226<sup>6</sup>.

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

```
350 \@@_add_lua_option:nnn
351   { singletonCache }
352   { boolean }
353   { true }

354 defaultOptions.singletonCache = true

355 local singletonCache = {
356   convert = nil,
357   options = nil,
358 }
```

---

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



### 2.1.5 File and Directory Names

`cacheDir`=*<path>* default: .

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

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

```
359 \@@_add_lua_option:nnn
360   { cacheDir }
361   { path }
362   { \markdownOptionOutputDir / _markdown\_jobname }
363 defaultOptions.cacheDir = "."
```

`contentBlocksLanguageMap`=*<filename>*  
default: `markdown-languages.json`

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

```
364 \@@_add_lua_option:nnn
365   { contentBlocksLanguageMap }
366   { path }
367   { markdown-languages.json }
368 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 3.1.6) and user-defined syntax extensions (see Section 2.1.2) have been applied.

```
369 \@@_add_lua_option:nnn
370   { debugExtensionsFileName }
371   { path }
372   { \markdownOptionOutputDir / \jobname .debug-extensions.json }
373 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.

```
374 \@@_add_lua_option:nnn
375   { frozenCacheFileName }
376   { path }
377   { \markdownOptionCacheDir / frozenCache.tex }
378 defaultOptions.frozenCacheFileName = "frozenCache.tex"
```

### 2.1.6 Parser Options

`autoIdentifiers`=true, false default: false

**true** Enable the Pandoc auto identifiers syntax extension<sup>7</sup>:

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

**# 123 Sesame Street**

**false** Disable the Pandoc auto identifiers syntax extension.

See also the option `gfmAutoIdentifiers`.

```
379 \@@_add_lua_option:nnn
380   { autoIdentifiers }
381   { boolean }
382   { false }
383 defaultOptions.autoIdentifiers = false
```

`blankBeforeBlockquote`=true, false default: false

**true** Require a blank line between a paragraph and the following blockquote.

**false** Do not require a blank line between a paragraph and the following blockquote.

---

<sup>7</sup>See [https://pandoc.org/MANUAL.html#extension-auto\\_identifiers](https://pandoc.org/MANUAL.html#extension-auto_identifiers).

```

384 \@@_add_lua_option:nnn
385   { blankBeforeBlockquote }
386   { boolean }
387   { false }

388 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.

```

389 \@@_add_lua_option:nnn
390   { blankBeforeCodeFence }
391   { boolean }
392   { false }

393 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.

```

394 \@@_add_lua_option:nnn
395   { blankBeforeDivFence }
396   { boolean }
397   { false }

398 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.

```

399 \@@_add_lua_option:nnn
400   { blankBeforeHeading }
401   { boolean }
402   { false }

403 defaultOptions.blankBeforeHeading = false

```

`blankBeforeList=true, false` default: false

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

```
404 \@@_add_lua_option:nnn
405   { blankBeforeList }
406   { boolean }
407   { false }

408 defaultOptions.blankBeforeList = false
```

`bracketedSpans=true, false` default: false

- `true`      Enable the Pandoc bracketed span syntax extension<sup>8</sup>:

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

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

```
409 \@@_add_lua_option:nnn
410   { bracketedSpans }
411   { boolean }
412   { false }

413 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.

```
414 \@@_add_lua_option:nnn
415   { breakableBlockquotes }
416   { boolean }
417   { true }

418 defaultOptions.breakableBlockquotes = true
```

---

<sup>8</sup>See [https://pandoc.org/MANUAL.html#extension-bracketed\\_spans](https://pandoc.org/MANUAL.html#extension-bracketed_spans).

`citationNbsps=true, false`

default: false

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

```
419 \@@_add_lua_option:nnn
420 { citationNbsps }
421 { boolean }
422 { true }

423 defaultOptions.citationNbsps = true
```

`citations=true, false`

default: false

- true** Enable the Pandoc citation syntax extension<sup>9</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.

```
424 \@@_add_lua_option:nnn
425 { citations }
426 { boolean }
427 { false }

428 defaultOptions.citations = false
```

---

<sup>9</sup>See <https://pandoc.org/MANUAL.html#extension-citations>.

`codeSpans=true, false`

default: true

**true** Enable the code span syntax:

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

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

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

```
429 \@@_add_lua_option:nnn  
430 { codeSpans }  
431 { boolean }  
432 { true }  
  
433 defaultOptions.codeSpans = true
```

`contentBlocks=true, false`

default: false

**true**

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

```
``` md  
http://example.com/minard.jpg (Napoleon's  
disastrous Russian campaign of 1812)  
/Flowchart.png "Engineering Flowchart"  
/Savings Account.csv 'Recent Transactions'  
/Example.swift  
/Lorem Ipsum.txt  
~~~~~
```

**false** Disable the iA Writer content blocks syntax extension.

```
434 \@@_add_lua_option:nnn  
435 { contentBlocks }  
436 { boolean }  
437 { false }  
  
438 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
```

```
439 \@@_add_lua_option:nnn
440   { contentLevel }
441   { string }
442   { block }
443 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 3.1.6) 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.

```
444 \@@_add_lua_option:nnn
445   { debugExtensions }
446   { boolean }
447   { false }
448 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.

```

449 \@@_add_lua_option:nnn
450   { definitionLists }
451   { boolean }
452   { false }

453 defaultOptions.definitionLists = false

```

**expectJekyllData=true, false**

default: false

**false**      When the **jekyllData** option is enabled, then a markdown document may begin with YAML metadata if and only if the metadata begin with the end-of-directives marker (**---**) and they end with either the end-of-directives or the end-of-document marker (**...**):

```

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

```



`true` When the `jeekyllData` option is enabled, then a markdown document may begin directly with YAML metadata and may contain nothing but YAML metadata.

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

```
454 \@@_add_lua_option:nnn
455   { expectJekyllData }
456   { boolean }
457   { false }
458 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  $\text{T}_{\text{E}}\text{X}$  directory structure.

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

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

```

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

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

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

return strike_through

```

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

```

459 metadata.user_extension_api_version = 2
460 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.

```

461 \cs_generate_variant:Nn
462 \@@_add_lua_option:nnn
463 { nnV }
464 \@@_add_lua_option:nnV
465 { extensions }
466 { clist }
467 \c_empty_clist
468 defaultOptions.extensions = {}

```

`fancyLists=true, false`

default: false

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

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

`false` Disable the Pandoc fancy list syntax extension.

```
469 \@@_add_lua_option:nnn
470 { fancyLists }
471 { boolean }
472 { false }
473 defaultOptions.fancyLists = false
```

`fencedCode=true, false`

default: true

`true` Enable the commonmark fenced code block extension:

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

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

`false` Disable the commonmark fenced code block extension.

```
474 \@@_add_lua_option:nnn
475 { fencedCode }
476 { boolean }
477 { true }
478 defaultOptions.fencedCode = true
```

---

<sup>10</sup>See <https://pandoc.org/MANUAL.html#org-fancy-lists>.

`fencedCodeAttributes=true, false`

default: false

**true** Enable the Pandoc fenced code attribute syntax extension<sup>11</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.

```
479 \@@_add_lua_option:nnn
480 { fencedCodeAttributes }
481 { boolean }
482 { false }

483 defaultOptions.fencedCodeAttributes = false
```

`fencedDivs=true, false`

default: false

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

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

And another.
:::::
```

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

```
484 \@@_add_lua_option:nnn
485 { fencedDivs }
486 { boolean }
487 { false }

488 defaultOptions.fencedDivs = false
```

<sup>11</sup>See [https://pandoc.org/MANUAL.html#extension-fenced\\_code\\_attributes](https://pandoc.org/MANUAL.html#extension-fenced_code_attributes).

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

`finalizeCache=true, false`

default: false

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

The frozen cache makes it possible to later typeset a plain  $\text{\TeX}$  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.

```
489 \@@_add_lua_option:nnn
490   { finalizeCache }
491   { boolean }
492   { false }

493 defaultOptions.finalizeCache = false
```

`frozenCacheCounter=<number>`

default: 0

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

Each frozen cache entry will define a  $\text{\TeX}$  macro `\markdownFrozenCache<number>` that will typeset markdown document number `<number>`.

```
494 \@@_add_lua_option:nnn
495   { frozenCacheCounter }
496   { counter }
497   { 0 }

498 defaultOptions.frozenCacheCounter = 0
```

`gfmAutoIdentifiers=true, false`

default: false

`true` Enable the Pandoc GitHub-flavored auto identifiers syntax extension<sup>13</sup>:

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

`# 123 Sesame Street`

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

---

<sup>13</sup>See [https://pandoc.org/MANUAL.html#extension-gfm\\_auto\\_identifiers](https://pandoc.org/MANUAL.html#extension-gfm_auto_identifiers).

See also the option `autoIdentifiers`.

```
499 \@@_add_lua_option:nnn
500   { gfmAutoIdentifiers }
501   { boolean }
502   { false }

503 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.

```
504 \@@_add_lua_option:nnn
505   { hashEnumerators }
506   { boolean }
507   { false }

508 defaultOptions.hashEnumerators = false
```

`headerAttributes=true, false`

default: `false`

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

```
# My first heading {#foo}

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

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

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

```
509 \@@_add_lua_option:nnn
510   { headerAttributes }
511   { boolean }
512   { false }

513 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.

```
514 \@@_add_lua_option:nnn
515   { html }
516   { boolean }
517   { true }

518 defaultOptions.html = true
```

`hybrid=true, false` default: `false`

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

```
519 \@@_add_lua_option:nnn
520   { hybrid }
521   { boolean }
522   { false }

523 defaultOptions.hybrid = false
```

`inlineCodeAttributes=true, false` default: `false`

- true** Enable the Pandoc inline code span attribute extension<sup>14</sup>:

``<$>`{.haskell}`

---

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

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

```
524 \@@_add_lua_option:nnn
525   { inlineCodeAttributes }
526   { boolean }
527   { false }

528 defaultOptions.inlineCodeAttributes = false
```

`inlineNotes=true, false`      default: `false`

`true`      Enable the Pandoc inline note syntax extension<sup>15</sup>:

Here is an inline note.<sup>15</sup>[Inlines notes are easier to write, since you don't have to pick an identifier and move down to type the note.]

`false`      Disable the Pandoc inline note syntax extension.

```
529 \@@_add_lua_option:nnn
530   { inlineNotes }
531   { boolean }
532   { false }

533 defaultOptions.inlineNotes = false
```

`jeekyllData=true, false`      default: `false`

`true`      Enable the Pandoc YAML metadata block syntax extension<sup>16</sup> for entering metadata in YAML:

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

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

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

<sup>16</sup>See [https://pandoc.org/MANUAL.html#extension-yaml\\_metadata\\_block](https://pandoc.org/MANUAL.html#extension-yaml_metadata_block).



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

```
534 \@@_add_lua_option:nnn
535   { jekyllData }
536   { boolean }
537   { false }

538 defaultOptions.jekyllData = false
```

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

**true**      Enable the Pandoc link and image attribute syntax extension<sup>17</sup>:

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

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

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

```
539 \@@_add_lua_option:nnn
540   { linkAttributes }
541   { boolean }
542   { false }

543 defaultOptions.linkAttributes = false
```

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

**true**      Enable the Pandoc line block syntax extension<sup>18</sup>:

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

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

```
544 \@@_add_lua_option:nnn
545   { lineBlocks }
546   { boolean }
547   { false }

548 defaultOptions.lineBlocks = false
```

---

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

<sup>18</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>19</sup>:

This ==is highlighted text.==

`false` Disable the Pandoc mark syntax extension.

```
549 \@@_add_lua_option:nnn
550   { mark }
551   { boolean }
552   { false }
553 defaultOptions.mark = false
```

`notes=true, false`

default: false

`true` Enable the Pandoc note syntax extension<sup>20</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.

```
554 \@@_add_lua_option:nnn
555   { notes }
556   { boolean }
557   { false }
558 defaultOptions.notes = false
```

---

<sup>19</sup>See <https://pandoc.org/MANUAL.html#extension-mark>.

<sup>20</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.

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

563 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.

```
564 \@@_add_lua_option:nnn
565 { preserveTabs }
566 { boolean }
567 { true }

568 defaultOptions.preserveTabs = true
```

`rawAttribute=true, false`

default: false

**true** Enable the Pandoc raw attribute syntax extension<sup>21</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>21</sup>See [https://pandoc.org/MANUAL.html#extension-raw\\_attribute](https://pandoc.org/MANUAL.html#extension-raw_attribute).

```

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

```

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

`false`      Disable the Pandoc raw attribute syntax extension.

```

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

573 defaultOptions.rawAttribute = false

```

`relativeReferences=true, false` default: false

`true`      Enable relative references<sup>22</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.

```

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

578 defaultOptions.relativeReferences = false

```

---

<sup>22</sup>See <https://datatracker.ietf.org/doc/html/rfc3986#section-4.2>.

`shiftHeadings`=*<shift amount>*

default: 0

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

```
579 \@@_add_lua_option:nnn
580   { shiftHeadings }
581   { number }
582   { 0 }

583 defaultOptions.shiftHeadings = 0
```

`slice`=*<the beginning and the end of a slice>*

default: `^ $`

Two space-separated selectors that specify the slice of a document that will be processed, whereas the remainder of the document will be ignored. The following selectors are recognized:

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

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

```
584 \@@_add_lua_option:nnn
585   { slice }
586   { slice }
587   { ^~$ }

588 defaultOptions.slice = "^ $"
```

`smartEllipses=true, false` default: false

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

`false` Preserve all ellipses in the input.

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

593 defaultOptions.smartEllipses = false
```

`startNumber=true, false` default: true

`true` Make the number in the first item of an ordered lists significant. The item numbers will be passed to the `\markdownRendererOliItemWithNumber` TeX macro.

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

```
594 \@@_add_lua_option:nnn
595 { startNumber }
596 { boolean }
597 { true }

598 defaultOptions.startNumber = true
```

`strikeThrough=true, false` default: false

`true` Enable the Pandoc strike-through syntax extension<sup>23</sup>:

This ~~is deleted text.~~

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

```
599 \@@_add_lua_option:nnn
600 { strikeThrough }
601 { boolean }
602 { false }

603 defaultOptions.strikeThrough = false
```

---

<sup>23</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.

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

`subscripts=true, false`

default: false

**true** Enable the Pandoc subscript syntax extension<sup>24</sup>:

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

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

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

---

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

`superscripts=true, false`

default: false

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

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

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

```
614 \@@_add_lua_option:nnn
615   { superscripts }
616   { boolean }
617   { false }

618 defaultOptions.superscripts = false
```

`tableAttributes=true, false`

default: false

**true**

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

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

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

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

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

623 defaultOptions.tableAttributes = false
```

---

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



`tableCaptions=true, false`

default: `false`

`true`

: Enable the Pandoc table caption syntax extension<sup>26</sup> for pipe tables (see the `pipeTables` option).

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

: Demonstration of pipe table syntax.
````
```

`false` Disable the Pandoc table caption syntax extension.

```
624 \@@_add_lua_option:nnn
625   { tableCaptions }
626   { boolean }
627   { false }

628 defaultOptions.tableCaptions = false
```

`taskLists=true, false`

default: `false`

`true` Enable the Pandoc task list syntax extension<sup>27</sup>:

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

`false` Disable the Pandoc task list syntax extension.

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

633 defaultOptions.taskLists = false
```

<sup>26</sup>See [https://pandoc.org/MANUAL.html#extension-table\\_captions](https://pandoc.org/MANUAL.html#extension-table_captions).

<sup>27</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.

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

`texMathDollars=true, false`

default: false

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

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

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

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

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

---

<sup>28</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>29</sup>:

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

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

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

648 defaultOptions.texMathDoubleBackslash = false
```

`texMathSingleBackslash=true, false` default: false

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

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

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

```
649 \@@_add_lua_option:nnn
650   { texMathSingleBackslash }
651   { boolean }
652   { false }

653 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>29</sup>See [https://pandoc.org/MANUAL.html#extension-tex\\_math\\_double\\_backslash](https://pandoc.org/MANUAL.html#extension-tex_math_double_backslash).

<sup>30</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.

```

654 \@@_add_lua_option:nnn
655   { tightLists }
656   { boolean }
657   { true }
658 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.

```

659 \@@_add_lua_option:nnn
660   { underscores }
661   { boolean }
662   { true }
663 \ExplSyntaxOff
664 defaultOptions.underscores = true

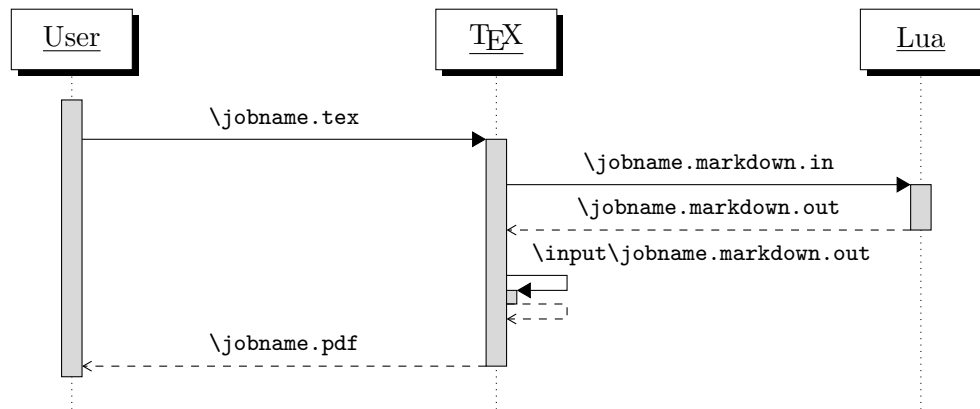
```

### 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.

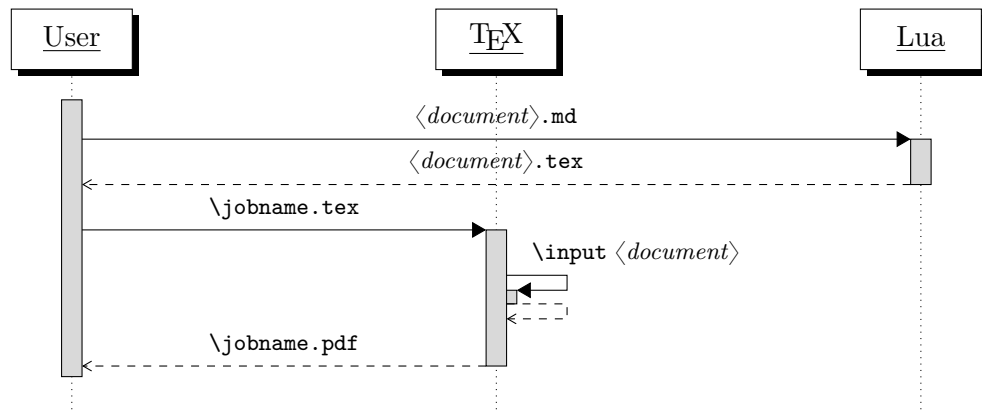
This procedure has the advantage of being fully automated. However, it also has several important disadvantages: The converted T<sub>E</sub>X documents are cached on the file system, taking up increasing amount of space. Unless the T<sub>E</sub>X engine includes a Lua interpreter, the package also requires 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**

```
665
666 local HELP_STRING = [[
667 Usage: texlua ]] .. arg[0] .. [[ [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
668 where OPTIONS are documented in the Lua interface section of the
669 technical Markdown package documentation.
670
671 When OUTPUT_FILE is unspecified, the result of the conversion will be
672 written to the standard output. When INPUT_FILE is also unspecified, the
673 result of the conversion will be read from the standard input.
674
675 Report bugs to: witiko@mail.muni.cz
676 Markdown package home page: <https://github.com/witiko/markdown>]]
677
```



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

```

678 local VERSION_STRING = [[
679 markdown-cli.lua (Markdown) ]] .. metadata.version .. [[
680
681 Copyright (C) ]] .. table.concat(metadata.copyright,
682                                   "\nCopyright (C) ") .. [[
683
684 License: ]] .. metadata.license
685
686 local function warn(s)
687   io.stderr:write("Warning: " .. s .. "\n") end
688
689 local function error(s)
690   io.stderr:write("Error: " .. s .. "\n")
691   os.exit(1)
692 end
  
```

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

```

693 local function camel_case(option_name)
694   local cased_option_name = option_name:gsub("_(%l)", function(match)
695     return match:sub(2, 2):upper()
696   end)
697   return cased_option_name
698 end
699
700 local function snake_case(option_name)
701   local cased_option_name = option_name:gsub("%l%u", function(match)
702     return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()
  
```

```

703     end)
704     return cased_option_name
705 end
706
707 local cases = {camel_case, snake_case}
708 local various_case_options = {}
709 for option_name, _ in pairs(defaultOptions) do
710     for _, case in ipairs(cases) do
711         various_case_options[case(option_name)] = option_name
712     end
713 end
714
715 local process_options = true
716 local options = {}
717 local input_filename
718 local output_filename
719 for i = 1, #arg do
720     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.

```

721         if arg[i] == "--" then
722             process_options = false
723             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.

```

724         elseif arg[i]:match("=") then
725             local key, value = arg[i]:match("(.)=(.*)")
726             if defaultOptions[key] == nil and
727                 various_case_options[key] ~= nil then
728                 key = various_case_options[key]
729             end

```

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

```

730             local default_type = type(defaultOptions[key])
731             if default_type == "boolean" then
732                 options[key] = (value == "true")
733             elseif default_type == "number" then
734                 options[key] = tonumber(value)
735             elseif default_type == "table" then
736                 options[key] = {}
737                 for item in value:gmatch("[^,]+") do
738                     table.insert(options[key], item)

```

```

739         end
740     else
741         if default_type ~= "string" then
742             if default_type == "nil" then
743                 warn('Option "' .. key .. '" not recognized.')
744             else
745                 warn('Option "' .. key .. '" type not recognized, please file ' ..
746                     'a report to the package maintainer.')
747             end
748             warn('Parsing the ' .. 'value "' .. value ..'" of option "' ..
749                 key .. '" as a string.')
750         end
751         options[key] = value
752     end
753     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.

```

754     elseif arg[i] == "--help" or arg[i] == "-h" then
755         print(HELP_STRING)
756         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.

```

757     elseif arg[i] == "--version" or arg[i] == "-v" then
758         print(VERSION_STRING)
759         os.exit()
760     end
761 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.

```

762     if input_filename == nil then
763         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.

```

764     elseif output_filename == nil then
765         output_filename = arg[i]
766     else
767         error('Unexpected argument: "' .. arg[i] .. '".')
768     end
769     ::continue::
770 end

```



The command-line Lua interface is implemented by the `markdown-cli.lua` file that can be invoked from the command line as follows:

```
texlua /path/to/markdown-cli.lua cacheDir=. -- hello.md hello.tex
```

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

```
\input hello
```

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

The plain T<sub>E</sub>X interface provides macros for the typesetting of markdown input from within plain T<sub>E</sub>X, for setting the Lua interface options (see Section 2.1.3) used during the conversion from markdown to plain T<sub>E</sub>X and for changing the way markdown the tokens are rendered.

```
771 \def\markdownLastModified{((LASTMODIFIED))}%  
772 \def\markdownVersion{((VERSION))}%
```

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

```
\input markdown
```

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

### 2.2.1 Typesetting Markdown

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

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

```
773 \let\markdownBegin\relax  
774 \let\markdownEnd\relax
```

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

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

Another limitation concerns spaces at the right end of an input line. In markdown, these are used to produce a forced line break. However, any such spaces are removed before the lines enter the input buffer of T<sub>E</sub>X [6, p. 46]. As a corollary, the `\markdownBegin` macro also ignores them.

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

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

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

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

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

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

```
775 \let\markdownInput\relax
```

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

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

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

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

```
776 \let\markdownEscape\relax
```

## 2.2.2 Options

The plain  $\text{\TeX}$  options are represented by  $\text{\TeX}$  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  $\text{\TeX}$  interface.

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

```
777 \ExplSyntaxOn
778 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
779 \cs_generate_variant:Nn
780   \tl_const:Nn
781   { NV }
782 \tl_if_exist:NF
783   \c_@@_top_layer_tl
784   {
785     \tl_const:NV
786       \c_@@_top_layer_tl
787       \c_@@_option_layer_plain_tex_tl
788   }
```

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

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

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

```
790 \prop_new:N \g_@@_plain_tex_option_types_prop
791 \prop_new:N \g_@@_default_plain_tex_options_prop
792 \seq_gput_right:NV \g_@@_option_layers_seq \c_@@_option_layer_plain_tex_tl
793 \cs_new:Nn
794   \@@_add_plain_tex_option:nnn
795   {
796     \@@_add_option:Vnnn
797       \c_@@_option_layer_plain_tex_tl
798       { #1 }
799       { #2 }
800       { #3 }
801   }
```

The plain T<sub>E</sub>X options may be also be specified via the `\markdownSetup` macro. Here, the plain T<sub>E</sub>X options are represented by a comma-delimited list of  $\langle key \rangle = \langle value \rangle$  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.

```

802 \cs_new:Nn
803   \@@_setup:n
804   {
805     \keys_set:nn
806       { markdown/options }
807       { #1 }
808   }
809 \cs_gset_eq:NN
810   \markdownSetup
811   \@@_setup:n

```

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.

```

812 \prg_new_conditional:Nnn
813   \@@_if_option:n
814   { TF, T, F }
815   {
816     \@@_get_option_type:nN
817       { #1 }
818     \l_tmpa_tl
819     \str_if_eq:NNF
820       \l_tmpa_tl
821       \c_@@_option_type_boolean_tl
822     {
823       \msg_error:nnxx
824         { markdown }
825         { expected-boolean-option }
826         { #1 }
827         { \l_tmpa_tl }
828     }
829     \@@_get_option_value:nN
830       { #1 }
831     \l_tmpa_tl
832     \str_if_eq:NNTF
833       \l_tmpa_tl
834       \c_@@_option_value_true_tl
835     { \prg_return_true: }
836     { \prg_return_false: }
837   }
838 \msg_new:nnn
839   { markdown }

```

```

840 { expected-boolean-option }
841 {
842   Option~#1~has~type~#2,~
843   but~a~boolean~was~expected.
844 }
845 \let\markdownIfOption=\@@_if_option:nTF

```

### 2.2.2.1 Finalizing and Freezing the Cache

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

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

```

846 \@@_add_plain_tex_option:nnn
847 { frozenCache }
848 { boolean }
849 { false }

```

The standard usage of the above two options is as follows:

1. Remove the `cacheDir` cache directory with stale auxiliary cache files.
2. Enable the `finalizeCache` option.
4. Typeset the plain T<sub>E</sub>X document to populate and finalize the cache.
5. Enable the `frozenCache` option.
6. Publish the source code of the plain T<sub>E</sub>X document and the `cacheDir` directory.

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

The expansion of this macro must not contain quotation marks (") or backslash symbols (\). Mind that T<sub>E</sub>X engines tend to put quotation marks around `\jobname`, when it contains spaces.

```

850 \@@_add_plain_tex_option:nnn
851 { inputTempFileName }
852 { path }
853 { \jobname.markdown.in }

```

The `\markdownOptionOutputDir` macro sets the path to the directory that will contain the auxiliary cache files produced by the Lua implementation and also the auxiliary files produced by the plain T<sub>E</sub>X implementation. The option defaults to `.`

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

The path must be set to the same value as the `-output-directory` option of your T<sub>E</sub>X engine for the package to function correctly. We need this macro to make the Lua implementation aware where it should store the helper files. The same limitations apply here as in the case of the `inputTempFileName` macro.

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

```
854 \cs_generate_variant:Nn
855   \@@_add_plain_tex_option:nnn
856   { nnV }
```

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

```
857 \ExplSyntaxOff
858 \input lt3luabridge.tex
859 \ExplSyntaxOn
860 \bool_if:nTF
861   {
862     \cs_if_exist_p:N
863       \luabridge_tl_set:Nn &&
864     (
865       \int_compare_p:nNn
866         { \g_luabridge_method_int }
867         =
868         { \c_luabridge_method_directlua_int } ||
869       \sys_if_shell_unrestricted_p:
870     )
871   }
872   {
873     \luabridge_tl_set:Nn
874       \l_tmpa_tl
875       { print(os.getenv("TEXMF_OUTPUT_DIRECTORY") or ".") }
876   }
877   {
878     \tl_set:Nn
879       \l_tmpa_tl
880       { . }
881   }
882 \@@_add_plain_tex_option:nnV
883   { outputDir }
884   { path }
885   \l_tmpa_tl
```

### 2.2.2.3 No default token renderer prototypes

The Markdown package provides default definitions for token renderer prototypes using the `witiko/markdown/defaults` theme (see Section [sec:#themes](#)). Although these default definitions provide a useful starting point for authors, they use extra resources, especially with higher-level T<sub>E</sub>X formats such as L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt. Furthermore, the default definitions may change at any time, which may pose a problem for maintainers of Markdown themes and templates who may require a stable output.

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

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

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

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

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

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

```
886 \@@_add_plain_tex_option:nnn  
887   { plain }  
888   { boolean }  
889   { 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.

```

890 \@@_add_plain_tex_option:nnn
891   { noDefaults }
892   { boolean }
893   { 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 Section 3.2.5) or not. Notably, this enables the use of markdown when writing T<sub>E</sub>X package documentation using the Doc L<sup>A</sup>T<sub>E</sub>X package [7] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```

894 \seq_gput_right:Nn
895   \g_@@_plain_tex_options_seq
896   { stripPercentSigns }
897 \prop_gput:Nnn
898   \g_@@_plain_tex_option_types_prop
899   { stripPercentSigns }
900   { boolean }
901 \prop_gput:Nnx
902   \g_@@_default_plain_tex_options_prop
903   { stripPercentSigns }
904   { 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.

```

905 \cs_new:Nn
906   \@@_define_option_commands_and_keyvals:
907   {
908     \seq_map_inline:Nn
909       \g_@@_option_layers_seq

```



```

910     {
911       \seq_map_inline:cn
912       { g_@@_ ##1 _options_seq }
913       {
914         \@@_define_option_command:n
915         { #####1 }

```

To make it easier to copy-and-paste options from Pandoc [4] such as [fancy\\_lists](#), [header\\_attributes](#), and [pipe\\_tables](#), we accept snake\_case in addition to camel-Case variants of options. As a bonus, studies [5] also show that snake\_case is faster to read than camelCase.

```

916         \@@_with_various_cases:nn
917         { #####1 }
918         {
919           \@@_define_option_keyval:nnn
920           { ##1 }
921           { #####1 }
922           { #####1 }
923         }
924       }
925     }
926   }
927   \cs_new:Nn
928   \@@_define_option_command:n
929   {

```

Do not override options defined before loading the package.

```

930   \@@_option_tl_to_csname:nN
931   { #1 }
932   \l_tmpa_tl
933   \cs_if_exist:cF
934   { \l_tmpa_tl }
935   {
936     \@@_get_default_option_value:nN
937     { #1 }
938     \l_tmpa_tl
939     \@@_set_option_value:nV
940     { #1 }
941     \l_tmpa_tl
942   }
943 }
944 \cs_new:Nn
945 \@@_set_option_value:nn
946 {
947   \@@_define_option:n
948   { #1 }
949   \@@_get_option_type:nN
950   { #1 }

```

```

951     \l_tmpa_tl
952     \str_if_eq:NNTF
953     \c_@@_option_type_counter_tl
954     \l_tmpa_tl
955     {
956         \@@_option_tl_to_csname:nN
957         { #1 }
958         \l_tmpa_tl
959         \int_gset:cn
960         { \l_tmpa_tl }
961         { #2 }
962     }
963     {
964         \@@_option_tl_to_csname:nN
965         { #1 }
966         \l_tmpa_tl
967         \cs_set:cpn
968         { \l_tmpa_tl }
969         { #2 }
970     }
971 }
972 \cs_generate_variant:Nn
973 \@@_set_option_value:nn
974 { nV }
975 \cs_new:Nn
976 \@@_define_option:n
977 {
978     \@@_option_tl_to_csname:nN
979     { #1 }
980     \l_tmpa_tl
981     \cs_if_free:cT
982     { \l_tmpa_tl }
983     {
984         \@@_get_option_type:nN
985         { #1 }
986         \l_tmpb_tl
987         \str_if_eq:NNT
988         \c_@@_option_type_counter_tl
989         \l_tmpb_tl
990         {
991             \@@_option_tl_to_csname:nN
992             { #1 }
993             \l_tmpa_tl
994             \int_new:c
995             { \l_tmpa_tl }
996         }
997     }

```

```

998     }
999     \cs_new:Nn
1000     \@@_define_option_keyval:nnn
1001     {
1002         \prop_get:cnN
1003         { g_@@_ #1 _option_types_prop }
1004         { #2 }
1005         \l_tmpa_tl
1006         \str_if_eq:VTF
1007         \l_tmpa_tl
1008         \c_@@_option_type_boolean_tl
1009         {
1010             \keys_define:nn
1011             { markdown/options }
1012             {

```

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

```

1013             #3 .code:n = {
1014                 \tl_set:Nx
1015                 \l_tmpa_tl
1016                 {
1017                     \str_case:nnF
1018                     { ##1 }
1019                     {
1020                         { yes } { true }
1021                         { no } { false }
1022                     }
1023                     { ##1 }
1024                 }
1025                 \@@_set_option_value:nV
1026                 { #2 }
1027                 \l_tmpa_tl
1028             },
1029             #3 .default:n = { true },
1030         }
1031     }
1032     {
1033         \keys_define:nn
1034         { markdown/options }
1035         {
1036             #3 .code:n = {
1037                 \@@_set_option_value:nn
1038                 { #2 }
1039                 { ##1 }
1040             },
1041         }

```

```
1042     }
```

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

```
1043     \str_if_eq:VVT
1044     \l_tmpa_tl
1045     \c_@@_option_type_clist_tl
1046     {
1047         \tl_set:Nn
1048             \l_tmpa_tl
1049             { #3 }
1050         \tl_reverse:N
1051             \l_tmpa_tl
1052         \str_if_eq:enF
1053             {
1054                 \tl_head:V
1055                 \l_tmpa_tl
1056             }
1057         { s }
1058         {
1059             \msg_error:nnn
1060                 { markdown }
1061                 { malformed-name-for-clist-option }
1062                 { #3 }
1063         }
1064         \tl_set:Nx
1065             \l_tmpa_tl
1066             {
1067                 \tl_tail:V
1068                 \l_tmpa_tl
1069             }
1070         \tl_reverse:N
1071             \l_tmpa_tl
1072         \tl_put_right:Nn
1073             \l_tmpa_tl
1074             {
1075                 .code:n = {
1076                     \@@_get_option_value:nN
1077                         { #2 }
1078                     \l_tmpa_tl
1079                     \clist_set:NV
1080                         \l_tmpa_clist
1081                         { \l_tmpa_tl, { ##1 } }
1082                     \@@_set_option_value:nV
```

```

1083             { #2 }
1084             \l_tmpa_clist
1085         }
1086     }
1087     \keys_define:nV
1088     { markdown/options }
1089     \l_tmpa_tl
1090 }
1091 }
1092 \cs_generate_variant:Nn
1093 \clist_set:Nn
1094 { NV }
1095 \cs_generate_variant:Nn
1096 \keys_define:nn
1097 { nV }
1098 \cs_generate_variant:Nn
1099 \@@_set_option_value:nn
1100 { nV }
1101 \prg_generate_conditional_variant:Nnn
1102 \str_if_eq:nn
1103 { en }
1104 { F }
1105 \msg_new:nnn
1106 { markdown }
1107 { malformed-name-for-clist-option }
1108 {
1109     Clist-option-name~#1~does~not~end~with~-s.
1110 }

```

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

```

1111 \str_if_eq:VVT
1112 \c_@@_top_layer_tl
1113 \c_@@_option_layer_plain_tex_tl
1114 {
1115     \@@_define_option_commands_and_keyvals:
1116 }
1117 \ExplSyntaxOff

```

### 2.2.3 Themes

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

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

*name*⟩.tex, where the *munged theme name* is the *theme name* after the substitution of all forward slashes (/) for an underscore (\_). The theme name is *qualified* and contains no underscores. A theme name is qualified if and only if it contains at least one forward slash. Themes are inspired by the Beamer L<sup>A</sup>T<sub>E</sub>X package, which provides similar functionality with its `\usetheme` macro [8, Section 15.1].

Theme names must be qualified to minimize naming conflicts between different themes with a similar purpose. The preferred format of a theme name is *⟨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 T<sub>E</sub>X directory structure. For example, loading a theme named `witiko/beamer/MU` would load a T<sub>E</sub>X document package named `markdownthemewitiko_beamer_MU.tex`.

```

1118 \ExplSyntaxOn
1119 \keys_define:nn
1120   { markdown/options }
1121   {
1122     theme .code:n = {
1123       \@@_set_theme:n
1124       { #1 }
1125     },
1126     import .code:n = {
1127       \tl_set:Nn
1128       \l_tmpa_tl
1129       { #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.

```

1130     \tl_replace_all:NnV
1131     \l_tmpa_tl
1132     { / }
1133     \c_backslash_str
1134     \keys_set:nV
1135     { markdown/options/import }
1136     \l_tmpa_tl
1137   },
1138 }

```

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

```

1139 \seq_new:N
1140   \g_@@_themes_seq
1141 \tl_new:N
1142   \g_@@_current_theme_tl
1143 \tl_gset:Nn
1144   \g_@@_current_theme_tl
1145   { }
1146 \seq_gput_right:NV
1147   \g_@@_themes_seq
1148   \g_@@_current_theme_tl
1149 \cs_new:Nn
1150   \@@_set_theme:n
1151   {

```

First, we validate the theme name.

```

1152   \str_if_in:nnF
1153     { #1 }
1154     { / }
1155     {
1156       \msg_error:nnn
1157         { markdown }
1158         { unqualified-theme-name }
1159         { #1 }
1160     }
1161   \str_if_in:nnT
1162     { #1 }
1163     { _ }
1164     {
1165       \msg_error:nnn
1166         { markdown }
1167         { underscores-in-theme-name }
1168         { #1 }
1169     }

```

Next, we munge the theme name.

```

1170   \str_set:Nn
1171     \l_tmpa_str
1172     { #1 }
1173   \str_replace_all:Nnn
1174     \l_tmpa_str
1175     { / }
1176     { _ }

```

Finally, we load the theme.

```

1177   \tl_gset:Nn
1178     \g_@@_current_theme_tl
1179     { #1 / }
1180   \seq_gput_right:NV

```

```

1181     \g_@@_themes_seq
1182     \g_@@_current_theme_tl
1183     \@@_load_theme:nV
1184     { #1 }
1185     \l_tmpa_str
1186     \seq_gpop_right:NN
1187     \g_@@_themes_seq
1188     \l_tmpa_tl
1189     \seq_get_right:NN
1190     \g_@@_themes_seq
1191     \l_tmpa_tl
1192     \tl_gset:NV
1193     \g_@@_current_theme_tl
1194     \l_tmpa_tl
1195 }
1196 \msg_new:nnnn
1197 { markdown }
1198 { unqualified-theme-name }
1199 { Won't~load~theme~with~unqualified~name~#1 }
1200 { Theme~names~must~contain~at~least~one~forward~slash }
1201 \msg_new:nnnn
1202 { markdown }
1203 { underscores-in-theme-name }
1204 { Won't~load~theme~with~an~underscore~in~its~name~#1 }
1205 { Theme~names~must~not~contain~underscores~in~their~names }
1206 \cs_generate_variant:Nn
1207 \tl_replace_all:Nnn
1208 { NnV }
1209 \ExplSyntaxOff

```

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

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

```

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

```

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



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

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

## 2.2.4 Snippets

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

```

1210 \ExplSyntaxOn
1211 \prop_new:N
1212   \g_@@_snippets_prop
1213 \cs_new:Nn
1214   \@@_setup_snippet:nn
1215   {
1216     \tl_if_empty:nT
1217       { #1 }
1218     {
1219       \msg_error:nnn
1220         { markdown }
1221         { empty-snippet-name }
1222         { #1 }
1223     }
1224     \tl_set:NV
1225       \l_tmpa_tl
1226       \g_@@_current_theme_tl
1227     \tl_put_right:Nn
1228       \l_tmpa_tl
1229       { #1 }
1230     \@@_if_snippet_exists:nT
1231       { #1 }
1232     {
1233       \msg_warning:nnV
1234         { markdown }
1235         { redefined-snippet }
1236         \l_tmpa_tl
1237     }
1238     \prop_gput:NVn
1239       \g_@@_snippets_prop
1240       \l_tmpa_tl
1241       { #2 }
1242   }
1243 \cs_gset_eq:NN

```

```

1244 \markdownSetupSnippet
1245 \@@_setup_snippet:nn
1246 \msg_new:nnnn
1247 { markdown }
1248 { empty-snippet-name }
1249 { Empty-snippet-name~#1 }
1250 { Pick-a-non-empty-name-for-your-snippet }
1251 \msg_new:nnn
1252 { markdown }
1253 { redefined-snippet }
1254 { Redefined~snippet~#1 }

```

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

```

1255 \prg_new_conditional:Nnn
1256 \@@_if_snippet_exists:n
1257 { TF, T, F }
1258 {
1259   \tl_set:NV
1260     \l_tmpa_tl
1261     \g_@@_current_theme_tl
1262   \tl_put_right:Nn
1263     \l_tmpa_tl
1264     { #1 }
1265   \prop_get:NVNTF
1266     \g_@@_snippets_prop
1267     \l_tmpa_tl
1268     \l_tmpb_tl
1269     { \prg_return_true: }
1270     { \prg_return_false: }
1271 }
1272 \cs_gset_eq:NN
1273 \markdownIfSnippetExists
1274 \@@_if_snippet_exists:nTF

```

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

```

1275 \keys_define:nn
1276 { markdown/options }
1277 {
1278   snippet .code:n = {
1279     \tl_set:NV
1280       \l_tmpa_tl
1281       \g_@@_current_theme_tl
1282     \tl_put_right:Nn
1283       \l_tmpa_tl
1284       { #1 }
1285     \@@_if_snippet_exists:nTF
1286       { #1 }

```

```

1287     {
1288         \prop_get:NVN
1289         \g_@@_snippets_prop
1290         \l_tmpa_tl
1291         \l_tmpb_tl
1292         \@@_setup:V
1293         \l_tmpb_tl
1294     }
1295     {
1296         \msg_error:nnV
1297         { markdown }
1298         { undefined-snippet }
1299         \l_tmpa_tl
1300     }
1301 }
1302 }
1303 \msg_new:nnn
1304 { markdown }
1305 { undefined-snippet }
1306 { Can't~invoke~undefined~snippet~#1 }
1307 \cs_generate_variant:Nn
1308 \@@_setup:n
1309 { V }
1310 \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,
  },
}

```

```

1311 \ExplSyntaxOn
1312 \tl_new:N
1313   \l_@@_import_current_theme_tl
1314 \keys_define:nn
1315   { markdown/options/import }
1316   {

```

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

```

1317     unknown .default:n = {},
1318     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.

```

1319     \tl_set_eq:NN
1320     \l_@@_import_current_theme_tl
1321     \l_keys_key_str
1322     \tl_replace_all:NVN
1323     \l_@@_import_current_theme_tl
1324     \c_backslash_str
1325     { / }

```

Here, we import the snippets.

```

1326     \clist_map_inline:nn
1327     { #1 }
1328     {
1329         \regex_extract_once:nnNTF
1330         { ^(.*)\s+as\s+(.*)$ }
1331         { ##1 }
1332         \l_tmpa_seq
1333         {
1334             \seq_pop:NN
1335             \l_tmpa_seq
1336             \l_tmpa_tl
1337             \seq_pop:NN
1338             \l_tmpa_seq
1339             \l_tmpa_tl
1340             \seq_pop:NN
1341             \l_tmpa_seq
1342             \l_tmpb_tl
1343         }
1344         {
1345             \tl_set:Nn
1346             \l_tmpa_tl
1347             { ##1 }
1348             \tl_set:Nn
1349             \l_tmpb_tl
1350             { ##1 }
1351         }
1352         \tl_put_left:Nn
1353         \l_tmpa_tl
1354         { / }
1355         \tl_put_left:NV
1356         \l_tmpa_tl
1357         \l_@@_import_current_theme_tl
1358         \@@_setup_snippet:Vx
1359         \l_tmpb_tl
1360         { snippet = { \l_tmpa_tl } }

```

```
1361      }
```

Here, we load the theme.

```
1362      \@@_set_theme:V
1363      \l_@@_import_current_theme_tl
1364    },
1365  }
1366 \cs_generate_variant:Nn
1367   \tl_replace_all:Nnn
1368   { NVn }
1369 \cs_generate_variant:Nn
1370   \@@_set_theme:n
1371   { V }
1372 \cs_generate_variant:Nn
1373   \@@_setup_snippet:nn
1374   { 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.

```
1375 \ExplSyntaxOn
1376 \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.

```
1377 \prop_new:N \g_@@_renderer_arities_prop
1378 \ExplSyntaxOff
```

#### 2.2.5.1 Attribute Renderers

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

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

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

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

`\markdownRendererAttributeKeyValue` represents a HTML attribute in the form  $\langle key \rangle = \langle value \rangle$  that is neither an identifier nor a class name. The macro receives two attributes that correspond to the  $\langle key \rangle$  and the  $\langle value \rangle$ , respectively.

```

1379 \def\markdownRendererAttributeIdentifier{%
1380   \markdownRendererAttributeIdentifierPrototype}%
1381 \ExplSyntaxOn
1382 \seq_gput_right:Nn
1383   \g_@@_renderers_seq
1384   { attributeIdentifier }
1385 \prop_gput:Nnn
1386   \g_@@_renderer_arities_prop
1387   { attributeIdentifier }
1388   { 1 }
1389 \ExplSyntaxOff
1390 \def\markdownRendererAttributeClassName{%
1391   \markdownRendererAttributeClassNamePrototype}%
1392 \ExplSyntaxOn
1393 \seq_gput_right:Nn
1394   \g_@@_renderers_seq
1395   { attributeClassName }
1396 \prop_gput:Nnn
1397   \g_@@_renderer_arities_prop
1398   { attributeClassName }
1399   { 1 }
1400 \ExplSyntaxOff
1401 \def\markdownRendererAttributeKeyValue{%
1402   \markdownRendererAttributeKeyValuePrototype}%
1403 \ExplSyntaxOn
1404 \seq_gput_right:Nn
1405   \g_@@_renderers_seq
1406   { attributeKeyValue }
1407 \prop_gput:Nnn
1408   \g_@@_renderer_arities_prop
1409   { attributeKeyValue }
1410   { 2 }
1411 \ExplSyntaxOff

```

### 2.2.5.2 Block Quote Renderers



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

```

1412 \def\markdownRendererBlockQuoteBegin{%
1413   \markdownRendererBlockQuoteBeginPrototype}%
1414 \ExplSyntaxOn
1415 \seq_gput_right:Nn
1416   \g_@@_renderers_seq
1417   { blockQuoteBegin }
1418 \prop_gput:Nnn
1419   \g_@@_renderer_arities_prop
1420   { blockQuoteBegin }
1421   { 0 }
1422 \ExplSyntaxOff

```

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

```

1423 \def\markdownRendererBlockQuoteEnd{%
1424   \markdownRendererBlockQuoteEndPrototype}%
1425 \ExplSyntaxOn
1426 \seq_gput_right:Nn
1427   \g_@@_renderers_seq
1428   { blockQuoteEnd }
1429 \prop_gput:Nnn
1430   \g_@@_renderer_arities_prop
1431   { blockQuoteEnd }
1432   { 0 }
1433 \ExplSyntaxOff

```

### 2.2.5.3 Bracketed Spans Attribute Context Renderers

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

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

```

1434 \def\markdownRendererBracketedSpanAttributeContextBegin{%
1435   \markdownRendererBracketedSpanAttributeContextBeginPrototype}%
1436 \ExplSyntaxOn
1437 \seq_gput_right:Nn
1438   \g_@@_renderers_seq
1439   { bracketedSpanAttributeContextBegin }
1440 \prop_gput:Nnn
1441   \g_@@_renderer_arities_prop
1442   { bracketedSpanAttributeContextBegin }
1443   { 0 }
1444 \ExplSyntaxOff

```

```

1445 \def\markdownRendererBracketedSpanAttributeContextEnd{%
1446   \markdownRendererBracketedSpanAttributeContextEndPrototype}%
1447 \ExplSyntaxOn
1448 \seq_gput_right:Nn
1449   \g_@@_renderers_seq
1450   { bracketedSpanAttributeContextEnd }
1451 \prop_gput:Nnn
1452   \g_@@_renderer_arities_prop
1453   { bracketedSpanAttributeContextEnd }
1454   { 0 }
1455 \ExplSyntaxOff

```

#### 2.2.5.4 Bullet List Renderers

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

```

1456 \def\markdownRendererUlBegin{%
1457   \markdownRendererUlBeginPrototype}%
1458 \ExplSyntaxOn
1459 \seq_gput_right:Nn
1460   \g_@@_renderers_seq
1461   { ulBegin }
1462 \prop_gput:Nnn
1463   \g_@@_renderer_arities_prop
1464   { ulBegin }
1465   { 0 }
1466 \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.

```

1467 \def\markdownRendererUlBeginTight{%
1468   \markdownRendererUlBeginTightPrototype}%
1469 \ExplSyntaxOn
1470 \seq_gput_right:Nn
1471   \g_@@_renderers_seq
1472   { ulBeginTight }
1473 \prop_gput:Nnn
1474   \g_@@_renderer_arities_prop
1475   { ulBeginTight }
1476   { 0 }
1477 \ExplSyntaxOff

```

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

```

1478 \def\markdownRendererUListItem{%
1479   \markdownRendererUListItemPrototype}%
1480 \ExplSyntaxOn
1481 \seq_gput_right:Nn
1482   \g_@@_renderers_seq
1483   { ulItem }
1484 \prop_gput:Nnn
1485   \g_@@_renderer_arities_prop
1486   { ulItem }
1487   { 0 }
1488 \ExplSyntaxOff

```

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

```

1489 \def\markdownRendererUListItemEnd{%
1490   \markdownRendererUListItemEndPrototype}%
1491 \ExplSyntaxOn
1492 \seq_gput_right:Nn
1493   \g_@@_renderers_seq
1494   { ulItemEnd }
1495 \prop_gput:Nnn
1496   \g_@@_renderer_arities_prop
1497   { ulItemEnd }
1498   { 0 }
1499 \ExplSyntaxOff

```

The `\markdownRendererUlEnd` 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.

```

1500 \def\markdownRendererUlEnd{%
1501   \markdownRendererUlEndPrototype}%
1502 \ExplSyntaxOn
1503 \seq_gput_right:Nn
1504   \g_@@_renderers_seq
1505   { ulEnd }
1506 \prop_gput:Nnn
1507   \g_@@_renderer_arities_prop
1508   { ulEnd }
1509   { 0 }
1510 \ExplSyntaxOff

```

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

```

1511 \def\markdownRendererUlEndTight{%

```

```

1512 \markdownRendererUEndTightPrototype}%
1513 \ExplSyntaxOn
1514 \seq_gput_right:Nn
1515 \g_@@_renderers_seq
1516 { ulEndTight }
1517 \prop_gput:Nnn
1518 \g_@@_renderer_arities_prop
1519 { ulEndTight }
1520 { 0 }
1521 \ExplSyntaxOff

```

### 2.2.5.5 Citation Renderers

The `\markdownRendererCite` macro represents a string of one or more parenthetical citations. This macro will only be produced, when the `citations` option is enabled. The macro receives the parameter `{<number of citations>}` followed by `<suppress author> {<prenote>}{<postnote>}{<name>}` repeated `<number of citations>` times. The `<suppress author>` parameter is either the token `-`, when the author's name is to be suppressed, or `+` otherwise.

```

1522 \def\markdownRendererCite{%
1523 \markdownRendererCitePrototype}%
1524 \ExplSyntaxOn
1525 \seq_gput_right:Nn
1526 \g_@@_renderers_seq
1527 { cite }
1528 \prop_gput:Nnn
1529 \g_@@_renderer_arities_prop
1530 { cite }
1531 { 1 }
1532 \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.

```

1533 \def\markdownRendererTextCite{%
1534 \markdownRendererTextCitePrototype}%
1535 \ExplSyntaxOn
1536 \seq_gput_right:Nn
1537 \g_@@_renderers_seq
1538 { textCite }
1539 \prop_gput:Nnn
1540 \g_@@_renderer_arities_prop
1541 { textCite }
1542 { 1 }
1543 \ExplSyntaxOff

```

### 2.2.5.6 Code Block Renderers

The `\markdownRendererInputVerbatim` macro represents a code block. The macro receives a single argument that corresponds to the filename of a file containing the code block contents.

```
1544 \def\markdownRendererInputVerbatim{%
1545   \markdownRendererInputVerbatimPrototype}%
1546 \ExplSyntaxOn
1547 \seq_gput_right:Nn
1548   \g_@@_renderers_seq
1549   { inputVerbatim }
1550 \prop_gput:Nnn
1551   \g_@@_renderer_arities_prop
1552   { inputVerbatim }
1553   { 1 }
1554 \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.

```
1555 \def\markdownRendererInputFencedCode{%
1556   \markdownRendererInputFencedCodePrototype}%
1557 \ExplSyntaxOn
1558 \seq_gput_right:Nn
1559   \g_@@_renderers_seq
1560   { inputFencedCode }
1561 \prop_gput:Nnn
1562   \g_@@_renderer_arities_prop
1563   { inputFencedCode }
1564   { 3 }
1565 \ExplSyntaxOff
```

### 2.2.5.7 Code Span Renderer

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

```
1566 \def\markdownRendererCodeSpan{%
1567   \markdownRendererCodeSpanPrototype}%
1568 \ExplSyntaxOn
1569 \seq_gput_right:Nn
1570   \g_@@_renderers_seq
1571   { codeSpan }
1572 \prop_gput:Nnn
1573   \g_@@_renderer_arities_prop
1574   { codeSpan }
```

```

1575 { 1 }
1576 \ExplSyntaxOff

```

### 2.2.5.8 Code Span Attribute Context Renderers

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

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

```

1577 \def\markdownRendererCodeSpanAttributeContextBegin{%
1578   \markdownRendererCodeSpanAttributeContextBeginPrototype}%
1579 \ExplSyntaxOn
1580 \seq_gput_right:Nn
1581   \g_@@_renderers_seq
1582   { codeSpanAttributeContextBegin }
1583 \prop_gput:Nnn
1584   \g_@@_renderer_arities_prop
1585   { codeSpanAttributeContextBegin }
1586   { 0 }
1587 \ExplSyntaxOff
1588 \def\markdownRendererCodeSpanAttributeContextEnd{%
1589   \markdownRendererCodeSpanAttributeContextEndPrototype}%
1590 \ExplSyntaxOn
1591 \seq_gput_right:Nn
1592   \g_@@_renderers_seq
1593   { codeSpanAttributeContextEnd }
1594 \prop_gput:Nnn
1595   \g_@@_renderer_arities_prop
1596   { codeSpanAttributeContextEnd }
1597   { 0 }
1598 \ExplSyntaxOff

```

### 2.2.5.9 Content Block Renderers

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

```

1599 \def\markdownRendererContentBlock{%
1600   \markdownRendererContentBlockPrototype}%
1601 \ExplSyntaxOn
1602 \seq_gput_right:Nn
1603   \g_@@_renderers_seq
1604   { contentBlock }
1605 \prop_gput:Nnn

```

```

1606 \g_@@_renderer_arities_prop
1607 { contentBlock }
1608 { 4 }
1609 \ExplSyntaxOff

```

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

```

1610 \def\markdownRendererContentBlockOnlineImage{%
1611 \markdownRendererContentBlockOnlineImagePrototype}%
1612 \ExplSyntaxOn
1613 \seq_gput_right:Nn
1614 \g_@@_renderers_seq
1615 { contentBlockOnlineImage }
1616 \prop_gput:Nnn
1617 \g_@@_renderer_arities_prop
1618 { contentBlockOnlineImage }
1619 { 4 }
1620 \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>31</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 [3] is a good starting point.

```

1621 \def\markdownRendererContentBlockCode{%
1622 \markdownRendererContentBlockCodePrototype}%
1623 \ExplSyntaxOn
1624 \seq_gput_right:Nn
1625 \g_@@_renderers_seq
1626 { contentBlockCode }
1627 \prop_gput:Nnn
1628 \g_@@_renderer_arities_prop

```

---

<sup>31</sup> Filenames other than `markdown-languages.json` may be specified using the `contentBlocksLanguageMap` Lua option.

```

1629 { contentBlockCode }
1630 { 5 }
1631 \ExplSyntaxOff

```

### 2.2.5.10 Definition List Renderers

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

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

```

1632 \def\markdownRendererDlBegin{%
1633   \markdownRendererDlBeginPrototype}%
1634 \ExplSyntaxOn
1635 \seq_gput_right:Nn
1636   \g_@@_renderers_seq
1637   { dlBegin }
1638 \prop_gput:Nnn
1639   \g_@@_renderer_arities_prop
1640   { dlBegin }
1641   { 0 }
1642 \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.

```

1643 \def\markdownRendererDlBeginTight{%
1644   \markdownRendererDlBeginTightPrototype}%
1645 \ExplSyntaxOn
1646 \seq_gput_right:Nn
1647   \g_@@_renderers_seq
1648   { dlBeginTight }
1649 \prop_gput:Nnn
1650   \g_@@_renderer_arities_prop
1651   { dlBeginTight }
1652   { 0 }
1653 \ExplSyntaxOff

```

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

```

1654 \def\markdownRendererDlItem{%
1655   \markdownRendererDlItemPrototype}%
1656 \ExplSyntaxOn
1657 \seq_gput_right:Nn
1658   \g_@@_renderers_seq

```



```

1659 { dlItem }
1660 \prop_gput:Nnn
1661 \g_@@_renderer_arities_prop
1662 { dlItem }
1663 { 1 }
1664 \ExplSyntaxOff

```

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

```

1665 \def\markdownRendererDlItemEnd{%
1666   \markdownRendererDlItemEndPrototype}%
1667 \ExplSyntaxOn
1668 \seq_gput_right:Nn
1669   \g_@@_renderers_seq
1670   { dlItemEnd }
1671 \prop_gput:Nnn
1672   \g_@@_renderer_arities_prop
1673   { dlItemEnd }
1674   { 0 }
1675 \ExplSyntaxOff

```

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

```

1676 \def\markdownRendererDlDefinitionBegin{%
1677   \markdownRendererDlDefinitionBeginPrototype}%
1678 \ExplSyntaxOn
1679 \seq_gput_right:Nn
1680   \g_@@_renderers_seq
1681   { dlDefinitionBegin }
1682 \prop_gput:Nnn
1683   \g_@@_renderer_arities_prop
1684   { dlDefinitionBegin }
1685   { 0 }
1686 \ExplSyntaxOff

```

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

```

1687 \def\markdownRendererDlDefinitionEnd{%
1688   \markdownRendererDlDefinitionEndPrototype}%
1689 \ExplSyntaxOn
1690 \seq_gput_right:Nn
1691   \g_@@_renderers_seq
1692   { dlDefinitionEnd }
1693 \prop_gput:Nnn
1694   \g_@@_renderer_arities_prop
1695   { dlDefinitionEnd }
1696   { 0 }

```

```
1697 \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.

```
1698 \def\markdownRendererDlEnd{%
1699   \markdownRendererDlEndPrototype}%
1700 \ExplSyntaxOn
1701 \seq_gput_right:Nn
1702   \g_@@_renderers_seq
1703   { dlEnd }
1704 \prop_gput:Nnn
1705   \g_@@_renderer_arities_prop
1706   { dlEnd }
1707   { 0 }
1708 \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.

```
1709 \def\markdownRendererDlEndTight{%
1710   \markdownRendererDlEndTightPrototype}%
1711 \ExplSyntaxOn
1712 \seq_gput_right:Nn
1713   \g_@@_renderers_seq
1714   { dlEndTight }
1715 \prop_gput:Nnn
1716   \g_@@_renderer_arities_prop
1717   { dlEndTight }
1718   { 0 }
1719 \ExplSyntaxOff
```

#### 2.2.5.11 Ellipsis Renderer

The `\markdownRendererEllipsis` macro replaces any occurrence of ASCII ellipses in the input text. This macro will only be produced, when the `smartEllipses` option is enabled. The macro receives no arguments.

```
1720 \def\markdownRendererEllipsis{%
1721   \markdownRendererEllipsisPrototype}%
1722 \ExplSyntaxOn
1723 \seq_gput_right:Nn
1724   \g_@@_renderers_seq
1725   { ellipsis }
1726 \prop_gput:Nnn
1727   \g_@@_renderer_arities_prop
```

```

1728 { ellipsis }
1729 { 0 }
1730 \ExplSyntaxOff

```

### 2.2.5.12 Emphasis Renderers

The `\markdownRendererEmphasis` macro represents an emphasized span of text. The macro receives a single argument that corresponds to the emphasized span of text.

```

1731 \def\markdownRendererEmphasis{%
1732   \markdownRendererEmphasisPrototype}%
1733 \ExplSyntaxOn
1734 \seq_gput_right:Nn
1735   \g_@@_renderers_seq
1736   { emphasis }
1737 \prop_gput:Nnn
1738   \g_@@_renderer_arities_prop
1739   { emphasis }
1740   { 1 }
1741 \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.

```

1742 \def\markdownRendererStrongEmphasis{%
1743   \markdownRendererStrongEmphasisPrototype}%
1744 \ExplSyntaxOn
1745 \seq_gput_right:Nn
1746   \g_@@_renderers_seq
1747   { strongEmphasis }
1748 \prop_gput:Nnn
1749   \g_@@_renderer_arities_prop
1750   { strongEmphasis }
1751   { 1 }
1752 \ExplSyntaxOff

```

### 2.2.5.13 Fenced Code Attribute Context Renderers

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

```

1753 \def\markdownRendererFencedCodeAttributeContextBegin{%
1754   \markdownRendererFencedCodeAttributeContextBeginPrototype}%
1755 \ExplSyntaxOn
1756 \seq_gput_right:Nn

```

```

1757 \g_@@_renderers_seq
1758 { fencedCodeAttributeContextBegin }
1759 \prop_gput:Nnn
1760 \g_@@_renderer_arities_prop
1761 { fencedCodeAttributeContextBegin }
1762 { 0 }
1763 \ExplSyntaxOff
1764 \def\markdownRendererFencedCodeAttributeContextEnd{%
1765 \markdownRendererFencedCodeAttributeContextEndPrototype}%
1766 \ExplSyntaxOn
1767 \seq_gput_right:Nn
1768 \g_@@_renderers_seq
1769 { fencedCodeAttributeContextEnd }
1770 \prop_gput:Nnn
1771 \g_@@_renderer_arities_prop
1772 { fencedCodeAttributeContextEnd }
1773 { 0 }
1774 \ExplSyntaxOff

```

#### 2.2.5.14 Fenced Div Attribute Context Renderers

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

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

```

1775 \def\markdownRendererFencedDivAttributeContextBegin{%
1776 \markdownRendererFencedDivAttributeContextBeginPrototype}%
1777 \ExplSyntaxOn
1778 \seq_gput_right:Nn
1779 \g_@@_renderers_seq
1780 { fencedDivAttributeContextBegin }
1781 \prop_gput:Nnn
1782 \g_@@_renderer_arities_prop
1783 { fencedDivAttributeContextBegin }
1784 { 0 }
1785 \ExplSyntaxOff
1786 \def\markdownRendererFencedDivAttributeContextEnd{%
1787 \markdownRendererFencedDivAttributeContextEndPrototype}%
1788 \ExplSyntaxOn
1789 \seq_gput_right:Nn
1790 \g_@@_renderers_seq
1791 { fencedDivAttributeContextEnd }
1792 \prop_gput:Nnn
1793 \g_@@_renderer_arities_prop
1794 { fencedDivAttributeContextEnd }
1795 { 0 }
1796 \ExplSyntaxOff

```

### 2.2.5.15 Header Attribute Context Renderers

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

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

```
1797 \def\markdownRendererHeaderAttributeContextBegin{%
1798   \markdownRendererHeaderAttributeContextBeginPrototype}%
1799 \ExplSyntaxOn
1800 \seq_gput_right:Nn
1801   \g_@@_renderers_seq
1802   { headerAttributeContextBegin }
1803 \prop_gput:Nnn
1804   \g_@@_renderer_arities_prop
1805   { headerAttributeContextBegin }
1806   { 0 }
1807 \ExplSyntaxOff
1808 \def\markdownRendererHeaderAttributeContextEnd{%
1809   \markdownRendererHeaderAttributeContextEndPrototype}%
1810 \ExplSyntaxOn
1811 \seq_gput_right:Nn
1812   \g_@@_renderers_seq
1813   { headerAttributeContextEnd }
1814 \prop_gput:Nnn
1815   \g_@@_renderer_arities_prop
1816   { headerAttributeContextEnd }
1817   { 0 }
1818 \ExplSyntaxOff
```

### 2.2.5.16 Heading Renderers

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

```
1819 \def\markdownRendererHeadingOne{%
1820   \markdownRendererHeadingOnePrototype}%
1821 \ExplSyntaxOn
1822 \seq_gput_right:Nn
1823   \g_@@_renderers_seq
1824   { headingOne }
1825 \prop_gput:Nnn
1826   \g_@@_renderer_arities_prop
1827   { headingOne }
1828   { 1 }
1829 \ExplSyntaxOff
```

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

```

1830 \def\markdownRendererHeadingTwo{%
1831   \markdownRendererHeadingTwoPrototype}%
1832 \ExplSyntaxOn
1833 \seq_gput_right:Nn
1834   \g_@@_renderers_seq
1835   { headingTwo }
1836 \prop_gput:Nnn
1837   \g_@@_renderer_arities_prop
1838   { headingTwo }
1839   { 1 }
1840 \ExplSyntaxOff

```

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

```

1841 \def\markdownRendererHeadingThree{%
1842   \markdownRendererHeadingThreePrototype}%
1843 \ExplSyntaxOn
1844 \seq_gput_right:Nn
1845   \g_@@_renderers_seq
1846   { headingThree }
1847 \prop_gput:Nnn
1848   \g_@@_renderer_arities_prop
1849   { headingThree }
1850   { 1 }
1851 \ExplSyntaxOff

```

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

```

1852 \def\markdownRendererHeadingFour{%
1853   \markdownRendererHeadingFourPrototype}%
1854 \ExplSyntaxOn
1855 \seq_gput_right:Nn
1856   \g_@@_renderers_seq
1857   { headingFour }
1858 \prop_gput:Nnn
1859   \g_@@_renderer_arities_prop
1860   { headingFour }
1861   { 1 }
1862 \ExplSyntaxOff

```

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

```

1863 \def\markdownRendererHeadingFive{%
1864   \markdownRendererHeadingFivePrototype}%

```

```

1865 \ExplSyntaxOn
1866 \seq_gput_right:Nn
1867   \g_@@_renderers_seq
1868   { headingFive }
1869 \prop_gput:Nnn
1870   \g_@@_renderer_arities_prop
1871   { headingFive }
1872   { 1 }
1873 \ExplSyntaxOff

```

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

```

1874 \def\markdownRendererHeadingSix{%
1875   \markdownRendererHeadingSixPrototype}%
1876 \ExplSyntaxOn
1877 \seq_gput_right:Nn
1878   \g_@@_renderers_seq
1879   { headingSix }
1880 \prop_gput:Nnn
1881   \g_@@_renderer_arities_prop
1882   { headingSix }
1883   { 1 }
1884 \ExplSyntaxOff

```

#### 2.2.5.17 Inline HTML Comment Renderer

The `\markdownRendererInlineHtmlComment` macro represents the contents of an inline HTML comment. This macro will only be produced, when the `html` option is enabled. The macro receives a single argument that corresponds to the contents of the HTML comment.

```

1885 \def\markdownRendererInlineHtmlComment{%
1886   \markdownRendererInlineHtmlCommentPrototype}%
1887 \ExplSyntaxOn
1888 \seq_gput_right:Nn
1889   \g_@@_renderers_seq
1890   { inlineHtmlComment }
1891 \prop_gput:Nnn
1892   \g_@@_renderer_arities_prop
1893   { inlineHtmlComment }
1894   { 1 }
1895 \ExplSyntaxOff

```

#### 2.2.5.18 HTML Tag and Element Renderers

The `\markdownRendererInlineHtmlTag` macro represents an opening, closing, or empty inline HTML tag. This macro will only be produced, when the `html` option is

enabled. The macro receives a single argument that corresponds to the contents of the HTML tag.

The `\markdownRendererInputBlockHtmlElement` macro represents a block HTML element. This macro will only be produced, when the `html` option is enabled. The macro receives a single argument that filename of a file containing the contents of the HTML element.

```

1896 \def\markdownRendererInlineHtmlTag{%
1897   \markdownRendererInlineHtmlTagPrototype}%
1898 \ExplSyntaxOn
1899 \seq_gput_right:Nn
1900   \g_@@_renderers_seq
1901   { inlineHtmlTag }
1902 \prop_gput:Nnn
1903   \g_@@_renderer_arities_prop
1904   { inlineHtmlTag }
1905   { 1 }
1906 \ExplSyntaxOff
1907 \def\markdownRendererInputBlockHtmlElement{%
1908   \markdownRendererInputBlockHtmlElementPrototype}%
1909 \ExplSyntaxOn
1910 \seq_gput_right:Nn
1911   \g_@@_renderers_seq
1912   { inputBlockHtmlElement }
1913 \prop_gput:Nnn
1914   \g_@@_renderer_arities_prop
1915   { inputBlockHtmlElement }
1916   { 1 }
1917 \ExplSyntaxOff

```

### 2.2.5.19 Image Renderer

The `\markdownRendererImage` macro represents an image. It receives four arguments: the label, the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the link.

```

1918 \def\markdownRendererImage{%
1919   \markdownRendererImagePrototype}%
1920 \ExplSyntaxOn
1921 \seq_gput_right:Nn
1922   \g_@@_renderers_seq
1923   { image }
1924 \prop_gput:Nnn
1925   \g_@@_renderer_arities_prop
1926   { image }
1927   { 4 }
1928 \ExplSyntaxOff

```



### 2.2.5.20 Image Attribute Context Renderers

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

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

```
1929 \def\markdownRendererImageAttributeContextBegin{%
1930   \markdownRendererImageAttributeContextBeginPrototype}%
1931 \ExplSyntaxOn
1932 \seq_gput_right:Nn
1933   \g_@@_renderers_seq
1934   { imageAttributeContextBegin }
1935 \prop_gput:Nnn
1936   \g_@@_renderer_arities_prop
1937   { imageAttributeContextBegin }
1938   { 0 }
1939 \ExplSyntaxOff
1940 \def\markdownRendererImageAttributeContextEnd{%
1941   \markdownRendererImageAttributeContextEndPrototype}%
1942 \ExplSyntaxOn
1943 \seq_gput_right:Nn
1944   \g_@@_renderers_seq
1945   { imageAttributeContextEnd }
1946 \prop_gput:Nnn
1947   \g_@@_renderer_arities_prop
1948   { imageAttributeContextEnd }
1949   { 0 }
1950 \ExplSyntaxOff
```

### 2.2.5.21 Interblock Separator Renderers

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

```
1951 \def\markdownRendererInterblockSeparator{%
1952   \markdownRendererInterblockSeparatorPrototype}%
1953 \ExplSyntaxOn
1954 \seq_gput_right:Nn
1955   \g_@@_renderers_seq
1956   { interblockSeparator }
1957 \prop_gput:Nnn
1958   \g_@@_renderer_arities_prop
1959   { interblockSeparator }
1960   { 0 }
1961 \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.

```

1962 \def\markdownRendererParagraphSeparator{%
1963   \markdownRendererParagraphSeparatorPrototype}%
1964 \ExplSyntaxOn
1965 \seq_gput_right:Nn
1966   \g_@@_renderers_seq
1967   { paragraphSeparator }
1968 \prop_gput:Nnn
1969   \g_@@_renderer_arities_prop
1970   { paragraphSeparator }
1971   { 0 }
1972 \ExplSyntaxOff

```

### 2.2.5.22 Line Block Renderers

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

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

```

1973 \def\markdownRendererLineBlockBegin{%
1974   \markdownRendererLineBlockBeginPrototype}%
1975 \ExplSyntaxOn
1976 \seq_gput_right:Nn
1977   \g_@@_renderers_seq
1978   { lineBlockBegin }
1979 \prop_gput:Nnn
1980   \g_@@_renderer_arities_prop
1981   { lineBlockBegin }
1982   { 0 }
1983 \ExplSyntaxOff
1984 \def\markdownRendererLineBlockEnd{%
1985   \markdownRendererLineBlockEndPrototype}%
1986 \ExplSyntaxOn
1987 \seq_gput_right:Nn
1988   \g_@@_renderers_seq
1989   { lineBlockEnd }
1990 \prop_gput:Nnn
1991   \g_@@_renderer_arities_prop
1992   { lineBlockEnd }
1993   { 0 }
1994 \ExplSyntaxOff

```

### 2.2.5.23 Line Break Renderers

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

```
1995 \def\markdownRendererSoftLineBreak{%
1996   \markdownRendererSoftLineBreakPrototype}%
1997 \ExplSyntaxOn
1998 \seq_gput_right:Nn
1999   \g_@@_renderers_seq
2000   { softLineBreak }
2001 \prop_gput:Nnn
2002   \g_@@_renderer_arities_prop
2003   { softLineBreak }
2004   { 0 }
2005 \ExplSyntaxOff
```

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

```
2006 \def\markdownRendererHardLineBreak{%
2007   \markdownRendererHardLineBreakPrototype}%
2008 \ExplSyntaxOn
2009 \seq_gput_right:Nn
2010   \g_@@_renderers_seq
2011   { hardLineBreak }
2012 \prop_gput:Nnn
2013   \g_@@_renderer_arities_prop
2014   { hardLineBreak }
2015   { 0 }
2016 \ExplSyntaxOff
```

### 2.2.5.24 Link Renderer

The `\markdownRendererLink` macro represents a hyperlink. It receives four arguments: the label, the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the link.

```
2017 \def\markdownRendererLink{%
2018   \markdownRendererLinkPrototype}%
2019 \ExplSyntaxOn
2020 \seq_gput_right:Nn
2021   \g_@@_renderers_seq
2022   { link }
2023 \prop_gput:Nnn
2024   \g_@@_renderer_arities_prop
2025   { link }
2026   { 4 }
2027 \ExplSyntaxOff
```

### 2.2.5.25 Link Attribute Context Renderers

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

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

```
2028 \def\markdownRendererLinkAttributeContextBegin{%
2029   \markdownRendererLinkAttributeContextBeginPrototype}%
2030 \ExplSyntaxOn
2031 \seq_gput_right:Nn
2032   \g_@@_renderers_seq
2033   { linkAttributeContextBegin }
2034 \prop_gput:Nnn
2035   \g_@@_renderer_arities_prop
2036   { linkAttributeContextBegin }
2037   { 0 }
2038 \ExplSyntaxOff
2039 \def\markdownRendererLinkAttributeContextEnd{%
2040   \markdownRendererLinkAttributeContextEndPrototype}%
2041 \ExplSyntaxOn
2042 \seq_gput_right:Nn
2043   \g_@@_renderers_seq
2044   { linkAttributeContextEnd }
2045 \prop_gput:Nnn
2046   \g_@@_renderer_arities_prop
2047   { linkAttributeContextEnd }
2048   { 0 }
2049 \ExplSyntaxOff
```

### 2.2.5.26 Marked Text Renderer

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

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

```
2050 \def\markdownRendererMark{%
2051   \markdownRendererMarkPrototype}%
2052 \ExplSyntaxOn
2053 \seq_gput_right:Nn
2054   \g_@@_renderers_seq
2055   { mark }
2056 \prop_gput:Nnn
2057   \g_@@_renderer_arities_prop
2058   { mark }
2059   { 1 }
2060 \ExplSyntaxOff
```

### 2.2.5.27 Markdown Document Renderers

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

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

```
2061 \def\markdownRendererDocumentBegin{%
2062   \markdownRendererDocumentBeginPrototype}%
2063 \ExplSyntaxOn
2064 \seq_gput_right:Nn
2065   \g_@@_renderers_seq
2066   { documentBegin }
2067 \prop_gput:Nnn
2068   \g_@@_renderer_arities_prop
2069   { documentBegin }
2070   { 0 }
2071 \ExplSyntaxOff
2072 \def\markdownRendererDocumentEnd{%
2073   \markdownRendererDocumentEndPrototype}%
2074 \ExplSyntaxOn
2075 \seq_gput_right:Nn
2076   \g_@@_renderers_seq
2077   { documentEnd }
2078 \prop_gput:Nnn
2079   \g_@@_renderer_arities_prop
2080   { documentEnd }
2081   { 0 }
2082 \ExplSyntaxOff
```

### 2.2.5.28 Non-Breaking Space Renderer

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

```
2083 \def\markdownRendererNbsp{%
2084   \markdownRendererNbspPrototype}%
2085 \ExplSyntaxOn
2086 \seq_gput_right:Nn
2087   \g_@@_renderers_seq
2088   { nbsp }
2089 \prop_gput:Nnn
2090   \g_@@_renderer_arities_prop
2091   { nbsp }
2092   { 0 }
2093 \ExplSyntaxOff
```

### 2.2.5.29 Note Renderer

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

```
2094 \def\markdownRendererNote{%
2095   \markdownRendererNotePrototype}%
2096 \ExplSyntaxOn
2097 \seq_gput_right:Nn
2098   \g_@@_renderers_seq
2099   { note }
2100 \prop_gput:Nnn
2101   \g_@@_renderer_arities_prop
2102   { note }
2103   { 1 }
2104 \ExplSyntaxOff
```

### 2.2.5.30 Ordered List Renderers

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

```
2105 \def\markdownRendererOlBegin{%
2106   \markdownRendererOlBeginPrototype}%
2107 \ExplSyntaxOn
2108 \seq_gput_right:Nn
2109   \g_@@_renderers_seq
2110   { olBegin }
2111 \prop_gput:Nnn
2112   \g_@@_renderer_arities_prop
2113   { olBegin }
2114   { 0 }
2115 \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.

```
2116 \def\markdownRendererOlBeginTight{%
2117   \markdownRendererOlBeginTightPrototype}%
2118 \ExplSyntaxOn
2119 \seq_gput_right:Nn
2120   \g_@@_renderers_seq
2121   { olBeginTight }
2122 \prop_gput:Nnn
```

```

2123 \g_@@_renderer_arities_prop
2124 { olBeginTight }
2125 { 0 }
2126 \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`).

```

2127 \def\markdownRendererFancyOlBegin{%
2128   \markdownRendererFancyOlBeginPrototype}%
2129 \ExplSyntaxOn
2130 \seq_gput_right:Nn
2131   \g_@@_renderers_seq
2132   { fancyOlBegin }
2133 \prop_gput:Nnn
2134   \g_@@_renderer_arities_prop
2135   { fancyOlBegin }
2136   { 2 }
2137 \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.

```

2138 \def\markdownRendererFancyOlBeginTight{%
2139   \markdownRendererFancyOlBeginTightPrototype}%
2140 \ExplSyntaxOn
2141 \seq_gput_right:Nn
2142   \g_@@_renderers_seq
2143   { fancyOlBeginTight }
2144 \prop_gput:Nnn
2145   \g_@@_renderer_arities_prop
2146   { fancyOlBeginTight }
2147   { 2 }
2148 \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.

```

2149 \def\markdownRendererOlItem{%

```

```

2150 \markdownRendererOlItemPrototype}%
2151 \ExplSyntaxOn
2152 \seq_gput_right:Nn
2153 \g_@@_renderers_seq
2154 { olItem }
2155 \prop_gput:Nnn
2156 \g_@@_renderer_arities_prop
2157 { olItem }
2158 { 0 }
2159 \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.

```

2160 \def\markdownRendererOlItemEnd{%
2161 \markdownRendererOlItemEndPrototype}%
2162 \ExplSyntaxOn
2163 \seq_gput_right:Nn
2164 \g_@@_renderers_seq
2165 { olItemEnd }
2166 \prop_gput:Nnn
2167 \g_@@_renderer_arities_prop
2168 { olItemEnd }
2169 { 0 }
2170 \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.

```

2171 \def\markdownRendererOlItemWithNumber{%
2172 \markdownRendererOlItemWithNumberPrototype}%
2173 \ExplSyntaxOn
2174 \seq_gput_right:Nn
2175 \g_@@_renderers_seq
2176 { olItemWithNumber }
2177 \prop_gput:Nnn
2178 \g_@@_renderer_arities_prop
2179 { olItemWithNumber }
2180 { 1 }
2181 \ExplSyntaxOff

```

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

```

2182 \def\markdownRendererFancyOlItem{%

```



```

2183 \markdownRendererFancyOliItemPrototype}%
2184 \ExplSyntaxOn
2185 \seq_gput_right:Nn
2186 \g_@@_renderers_seq
2187 { fancyOliItem }
2188 \prop_gput:Nnn
2189 \g_@@_renderer_arities_prop
2190 { fancyOliItem }
2191 { 0 }
2192 \ExplSyntaxOff

```

The `\markdownRendererFancyOliItemEnd` 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.

```

2193 \def\markdownRendererFancyOliItemEnd{%
2194 \markdownRendererFancyOliItemEndPrototype}%
2195 \ExplSyntaxOn
2196 \seq_gput_right:Nn
2197 \g_@@_renderers_seq
2198 { fancyOliItemEnd }
2199 \prop_gput:Nnn
2200 \g_@@_renderer_arities_prop
2201 { fancyOliItemEnd }
2202 { 0 }
2203 \ExplSyntaxOff

```

The `\markdownRendererFancyOliItemWithNumber` 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.

```

2204 \def\markdownRendererFancyOliItemWithNumber{%
2205 \markdownRendererFancyOliItemWithNumberPrototype}%
2206 \ExplSyntaxOn
2207 \seq_gput_right:Nn
2208 \g_@@_renderers_seq
2209 { fancyOliItemWithNumber }
2210 \prop_gput:Nnn
2211 \g_@@_renderer_arities_prop
2212 { fancyOliItemWithNumber }
2213 { 1 }
2214 \ExplSyntaxOff

```

The `\markdownRendererOliEnd` 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.

```

2215 \def\markdownRendererOlEnd{%
2216   \markdownRendererOlEndPrototype}%
2217 \ExplSyntaxOn
2218 \seq_gput_right:Nn
2219   \g_@@_renderers_seq
2220   { olEnd }
2221 \prop_gput:Nnn
2222   \g_@@_renderer_arities_prop
2223   { olEnd }
2224   { 0 }
2225 \ExplSyntaxOff

```

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

```

2226 \def\markdownRendererOlEndTight{%
2227   \markdownRendererOlEndTightPrototype}%
2228 \ExplSyntaxOn
2229 \seq_gput_right:Nn
2230   \g_@@_renderers_seq
2231   { olEndTight }
2232 \prop_gput:Nnn
2233   \g_@@_renderer_arities_prop
2234   { olEndTight }
2235   { 0 }
2236 \ExplSyntaxOff

```

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

```

2237 \def\markdownRendererFancyOlEnd{%
2238   \markdownRendererFancyOlEndPrototype}%
2239 \ExplSyntaxOn
2240 \seq_gput_right:Nn
2241   \g_@@_renderers_seq
2242   { fancyOlEnd }
2243 \prop_gput:Nnn
2244   \g_@@_renderer_arities_prop
2245   { fancyOlEnd }
2246   { 0 }
2247 \ExplSyntaxOff

```

The `\markdownRendererFancyOlEndTight` macro represents the end of a fancy ordered list that contains no item with several paragraphs of text (the list is tight).

This macro will only be produced, when the `fancyLists` and `tightLists` options are enabled. The macro receives no arguments.

```

2248 \def\markdownRendererFancyO1EndTight{%
2249   \markdownRendererFancyO1EndTightPrototype}%
2250 \ExplSyntaxOn
2251 \seq_gput_right:Nn
2252   \g_@@_renderers_seq
2253   { fancyO1EndTight }
2254 \prop_gput:Nnn
2255   \g_@@_renderer_arities_prop
2256   { fancyO1EndTight }
2257   { 0 }
2258 \ExplSyntaxOff

```

### 2.2.5.31 Raw Content Renderers

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

```

2259 \def\markdownRendererInputRawInline{%
2260   \markdownRendererInputRawInlinePrototype}%
2261 \ExplSyntaxOn
2262 \seq_gput_right:Nn
2263   \g_@@_renderers_seq
2264   { inputRawInline }
2265 \prop_gput:Nnn
2266   \g_@@_renderer_arities_prop
2267   { inputRawInline }
2268   { 2 }
2269 \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.

```

2270 \def\markdownRendererInputRawBlock{%
2271   \markdownRendererInputRawBlockPrototype}%
2272 \ExplSyntaxOn
2273 \seq_gput_right:Nn
2274   \g_@@_renderers_seq
2275   { inputRawBlock }
2276 \prop_gput:Nnn
2277   \g_@@_renderer_arities_prop
2278   { inputRawBlock }
2279   { 2 }

```

```
2280 \ExplSyntaxOff
```

### 2.2.5.32 Section Renderers

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

```
2281 \def\markdownRendererSectionBegin{%
2282   \markdownRendererSectionBeginPrototype}%
2283 \ExplSyntaxOn
2284 \seq_gput_right:Nn
2285   \g_@@_renderers_seq
2286   { sectionBegin }
2287 \prop_gput:Nnn
2288   \g_@@_renderer_arities_prop
2289   { sectionBegin }
2290   { 0 }
2291 \ExplSyntaxOff
2292 \def\markdownRendererSectionEnd{%
2293   \markdownRendererSectionEndPrototype}%
2294 \ExplSyntaxOn
2295 \seq_gput_right:Nn
2296   \g_@@_renderers_seq
2297   { sectionEnd }
2298 \prop_gput:Nnn
2299   \g_@@_renderer_arities_prop
2300   { sectionEnd }
2301   { 0 }
2302 \ExplSyntaxOff
```

### 2.2.5.33 Replacement Character Renderers

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

```
2303 \def\markdownRendererReplacementCharacter{%
2304   \markdownRendererReplacementCharacterPrototype}%
2305 \ExplSyntaxOn
2306 \seq_gput_right:Nn
2307   \g_@@_renderers_seq
2308   { replacementCharacter }
2309 \prop_gput:Nnn
2310   \g_@@_renderer_arities_prop
2311   { replacementCharacter }
2312   { 0 }
2313 \ExplSyntaxOff
```

### 2.2.5.34 Special Character Renderers

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

```

2314 \def\markdownRendererLeftBrace{%
2315   \markdownRendererLeftBracePrototype}%
2316 \ExplSyntaxOn
2317 \seq_gput_right:Nn
2318   \g_@@_renderers_seq
2319   { leftBrace }
2320 \prop_gput:Nnn
2321   \g_@@_renderer_arities_prop
2322   { leftBrace }
2323   { 0 }
2324 \ExplSyntaxOff
2325 \def\markdownRendererRightBrace{%
2326   \markdownRendererRightBracePrototype}%
2327 \ExplSyntaxOn
2328 \seq_gput_right:Nn
2329   \g_@@_renderers_seq
2330   { rightBrace }
2331 \prop_gput:Nnn
2332   \g_@@_renderer_arities_prop
2333   { rightBrace }
2334   { 0 }
2335 \ExplSyntaxOff
2336 \def\markdownRendererDollarSign{%
2337   \markdownRendererDollarSignPrototype}%
2338 \ExplSyntaxOn
2339 \seq_gput_right:Nn
2340   \g_@@_renderers_seq
2341   { dollarSign }
2342 \prop_gput:Nnn
2343   \g_@@_renderer_arities_prop
2344   { dollarSign }
2345   { 0 }
2346 \ExplSyntaxOff
2347 \def\markdownRendererPercentSign{%
2348   \markdownRendererPercentSignPrototype}%
2349 \ExplSyntaxOn
2350 \seq_gput_right:Nn
2351   \g_@@_renderers_seq
2352   { percentSign }
2353 \prop_gput:Nnn
2354   \g_@@_renderer_arities_prop
2355   { percentSign }
2356   { 0 }

```

```

2357 \ExplSyntaxOff
2358 \def\markdownRendererAmpersand{%
2359   \markdownRendererAmpersandPrototype}%
2360 \ExplSyntaxOn
2361 \seq_gput_right:Nn
2362   \g_@@_renderers_seq
2363   { ampersand }
2364 \prop_gput:Nnn
2365   \g_@@_renderer_arities_prop
2366   { ampersand }
2367   { 0 }
2368 \ExplSyntaxOff
2369 \def\markdownRendererUnderscore{%
2370   \markdownRendererUnderscorePrototype}%
2371 \ExplSyntaxOn
2372 \seq_gput_right:Nn
2373   \g_@@_renderers_seq
2374   { underscore }
2375 \prop_gput:Nnn
2376   \g_@@_renderer_arities_prop
2377   { underscore }
2378   { 0 }
2379 \ExplSyntaxOff
2380 \def\markdownRendererHash{%
2381   \markdownRendererHashPrototype}%
2382 \ExplSyntaxOn
2383 \seq_gput_right:Nn
2384   \g_@@_renderers_seq
2385   { hash }
2386 \prop_gput:Nnn
2387   \g_@@_renderer_arities_prop
2388   { hash }
2389   { 0 }
2390 \ExplSyntaxOff
2391 \def\markdownRendererCircumflex{%
2392   \markdownRendererCircumflexPrototype}%
2393 \ExplSyntaxOn
2394 \seq_gput_right:Nn
2395   \g_@@_renderers_seq
2396   { circumflex }
2397 \prop_gput:Nnn
2398   \g_@@_renderer_arities_prop
2399   { circumflex }
2400   { 0 }
2401 \ExplSyntaxOff
2402 \def\markdownRendererBackslash{%
2403   \markdownRendererBackslashPrototype}%

```

```

2404 \ExplSyntaxOn
2405 \seq_gput_right:Nn
2406   \g_@@_renderers_seq
2407   { backslash }
2408 \prop_gput:Nnn
2409   \g_@@_renderer_arities_prop
2410   { backslash }
2411   { 0 }
2412 \ExplSyntaxOff
2413 \def\markdownRendererTilde{%
2414   \markdownRendererTildePrototype}%
2415 \ExplSyntaxOn
2416 \seq_gput_right:Nn
2417   \g_@@_renderers_seq
2418   { tilde }
2419 \prop_gput:Nnn
2420   \g_@@_renderer_arities_prop
2421   { tilde }
2422   { 0 }
2423 \ExplSyntaxOff
2424 \def\markdownRendererPipe{%
2425   \markdownRendererPipePrototype}%
2426 \ExplSyntaxOn
2427 \seq_gput_right:Nn
2428   \g_@@_renderers_seq
2429   { pipe }
2430 \prop_gput:Nnn
2431   \g_@@_renderer_arities_prop
2432   { pipe }
2433   { 0 }
2434 \ExplSyntaxOff

```

### 2.2.5.35 Strike-Through Renderer

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

```

2435 \def\markdownRendererStrikeThrough{%
2436   \markdownRendererStrikeThroughPrototype}%
2437 \ExplSyntaxOn
2438 \seq_gput_right:Nn
2439   \g_@@_renderers_seq
2440   { strikeThrough }
2441 \prop_gput:Nnn
2442   \g_@@_renderer_arities_prop
2443   { strikeThrough }

```

```

2444 { 1 }
2445 \ExplSyntaxOff

```

### 2.2.5.36 Subscript Renderer

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

```

2446 \def\markdownRendererSubscript{%
2447   \markdownRendererSubscriptPrototype}%
2448 \ExplSyntaxOn
2449 \seq_gput_right:Nn
2450   \g_@@_renderers_seq
2451   { subscript }
2452 \prop_gput:Nnn
2453   \g_@@_renderer_arities_prop
2454   { subscript }
2455   { 1 }

```

### 2.2.5.37 Superscript Renderer

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

```

2456 \def\markdownRendererSuperscript{%
2457   \markdownRendererSuperscriptPrototype}%
2458 \ExplSyntaxOn
2459 \seq_gput_right:Nn
2460   \g_@@_renderers_seq
2461   { superscript }
2462 \prop_gput:Nnn
2463   \g_@@_renderer_arities_prop
2464   { superscript }
2465   { 1 }
2466 \ExplSyntaxOff

```

### 2.2.5.38 Table Attribute Context Renderers

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

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

```

2467 \def\markdownRendererTableAttributeContextBegin{%
2468   \markdownRendererTableAttributeContextBeginPrototype}%
2469 \ExplSyntaxOn

```



```

2470 \seq_gput_right:Nn
2471   \g_@@_renderers_seq
2472   { tableAttributeContextBegin }
2473 \prop_gput:Nnn
2474   \g_@@_renderer_arities_prop
2475   { tableAttributeContextBegin }
2476   { 0 }
2477 \ExplSyntaxOff
2478 \def\markdownRendererTableAttributeContextEnd{%
2479   \markdownRendererTableAttributeContextEndPrototype}%
2480 \ExplSyntaxOn
2481 \seq_gput_right:Nn
2482   \g_@@_renderers_seq
2483   { tableAttributeContextEnd }
2484 \prop_gput:Nnn
2485   \g_@@_renderer_arities_prop
2486   { tableAttributeContextEnd }
2487   { 0 }
2488 \ExplSyntaxOff

```

### 2.2.5.39 Table Renderer

The `\markdownRendererTable` macro represents a table. This macro will only be produced, when the `pipeTables` option is enabled. The macro receives the parameters `{<caption>}{<number of rows>}{<number of columns>}` followed by `{<alignments>}` and then by `{<row>}` repeated `<number of rows>` times, where `<row>` is `{<column>}` repeated `<number of columns>` times, `<alignments>` is `<alignment>` repeated `<number of columns>` times, and `<alignment>` is one of the following:

- **d** – The corresponding column has an unspecified (default) alignment.
- **l** – The corresponding column is left-aligned.
- **c** – The corresponding column is centered.
- **r** – The corresponding column is right-aligned.

```

2489 \def\markdownRendererTable{%
2490   \markdownRendererTablePrototype}%
2491 \ExplSyntaxOn
2492 \seq_gput_right:Nn
2493   \g_@@_renderers_seq
2494   { table }
2495 \prop_gput:Nnn
2496   \g_@@_renderer_arities_prop
2497   { table }
2498   { 3 }
2499 \ExplSyntaxOff

```

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

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

```
2500 \def\markdownRendererInlineMath{%
2501   \markdownRendererInlineMathPrototype}%
2502 \ExplSyntaxOn
2503 \seq_gput_right:Nn
2504   \g_@@_renderers_seq
2505   { inlineMath }
2506 \prop_gput:Nnn
2507   \g_@@_renderer_arities_prop
2508   { inlineMath }
2509   { 1 }
2510 \ExplSyntaxOff
2511 \def\markdownRendererDisplayMath{%
2512   \markdownRendererDisplayMathPrototype}%
2513 \ExplSyntaxOn
2514 \seq_gput_right:Nn
2515   \g_@@_renderers_seq
2516   { displayMath }
2517 \prop_gput:Nnn
2518   \g_@@_renderer_arities_prop
2519   { displayMath }
2520   { 1 }
2521 \ExplSyntaxOff
```

#### 2.2.5.41 Thematic Break Renderer

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

```
2522 \def\markdownRendererThematicBreak{%
2523   \markdownRendererThematicBreakPrototype}%
2524 \ExplSyntaxOn
2525 \seq_gput_right:Nn
2526   \g_@@_renderers_seq
2527   { thematicBreak }
2528 \prop_gput:Nnn
2529   \g_@@_renderer_arities_prop
2530   { thematicBreak }
2531   { 0 }
2532 \ExplSyntaxOff
```

#### 2.2.5.42 Tickbox Renderers

The macros named `\markdownRendererTickedBox`, `\markdownRendererHalfTickedBox`, and `\markdownRendererUntickedBox` represent ticked and unticked boxes, respectively. These macros will either be produced, when the `taskLists` option is enabled, or when the Ballot Box with X (☒, U+2612), Hourglass (⌚, U+231B) or Ballot Box (☐, U+2610) Unicode characters are encountered in the markdown input, respectively.

```

2533 \def\markdownRendererTickedBox{%
2534   \markdownRendererTickedBoxPrototype}%
2535 \ExplSyntaxOn
2536 \seq_gput_right:Nn
2537   \g_@@_renderers_seq
2538   { tickedBox }
2539 \prop_gput:Nnn
2540   \g_@@_renderer_arities_prop
2541   { tickedBox }
2542   { 0 }
2543 \ExplSyntaxOff
2544 \def\markdownRendererHalfTickedBox{%
2545   \markdownRendererHalfTickedBoxPrototype}%
2546 \ExplSyntaxOn
2547 \seq_gput_right:Nn
2548   \g_@@_renderers_seq
2549   { halfTickedBox }
2550 \prop_gput:Nnn
2551   \g_@@_renderer_arities_prop
2552   { halfTickedBox }
2553   { 0 }
2554 \ExplSyntaxOff
2555 \def\markdownRendererUntickedBox{%
2556   \markdownRendererUntickedBoxPrototype}%
2557 \ExplSyntaxOn
2558 \seq_gput_right:Nn
2559   \g_@@_renderers_seq
2560   { untickedBox }
2561 \prop_gput:Nnn
2562   \g_@@_renderer_arities_prop
2563   { untickedBox }
2564   { 0 }
2565 \ExplSyntaxOff

```

#### 2.2.5.43 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.

```

2566 \def\markdownRendererJekyllDataBegin{%
2567   \markdownRendererJekyllDataBeginPrototype}%

```

```

2568 \ExplSyntaxOn
2569 \seq_gput_right:Nn
2570   \g_@@_renderers_seq
2571   { jekyllDataBegin }
2572 \prop_gput:Nnn
2573   \g_@@_renderer_arities_prop
2574   { jekyllDataBegin }
2575   { 0 }
2576 \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.

```

2577 \def\markdownRendererJekyllDataEnd{%
2578   \markdownRendererJekyllDataEndPrototype}%
2579 \ExplSyntaxOn
2580 \seq_gput_right:Nn
2581   \g_@@_renderers_seq
2582   { jekyllDataEnd }
2583 \prop_gput:Nnn
2584   \g_@@_renderer_arities_prop
2585   { jekyllDataEnd }
2586   { 0 }
2587 \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.

```

2588 \def\markdownRendererJekyllDataMappingBegin{%
2589   \markdownRendererJekyllDataMappingBeginPrototype}%
2590 \ExplSyntaxOn
2591 \seq_gput_right:Nn
2592   \g_@@_renderers_seq
2593   { jekyllDataMappingBegin }
2594 \prop_gput:Nnn
2595   \g_@@_renderer_arities_prop
2596   { jekyllDataMappingBegin }
2597   { 2 }
2598 \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.

```

2599 \def\markdownRendererJekyllDataMappingEnd{%

```

```

2600 \markdownRendererJekyllDataMappingEndPrototype}%
2601 \ExplSyntaxOn
2602 \seq_gput_right:Nn
2603 \g_@@_renderers_seq
2604 { jekyllDataMappingEnd }
2605 \prop_gput:Nnn
2606 \g_@@_renderer_arities_prop
2607 { jekyllDataMappingEnd }
2608 { 0 }
2609 \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.

```

2610 \def\markdownRendererJekyllDataSequenceBegin{%
2611 \markdownRendererJekyllDataSequenceBeginPrototype}%
2612 \ExplSyntaxOn
2613 \seq_gput_right:Nn
2614 \g_@@_renderers_seq
2615 { jekyllDataSequenceBegin }
2616 \prop_gput:Nnn
2617 \g_@@_renderer_arities_prop
2618 { jekyllDataSequenceBegin }
2619 { 2 }
2620 \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.

```

2621 \def\markdownRendererJekyllDataSequenceEnd{%
2622 \markdownRendererJekyllDataSequenceEndPrototype}%
2623 \ExplSyntaxOn
2624 \seq_gput_right:Nn
2625 \g_@@_renderers_seq
2626 { jekyllDataSequenceEnd }
2627 \prop_gput:Nnn
2628 \g_@@_renderer_arities_prop
2629 { jekyllDataSequenceEnd }
2630 { 0 }
2631 \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.

```
2632 \def\markdownRendererJekyllDataBoolean{%
2633   \markdownRendererJekyllDataBooleanPrototype}%
2634 \ExplSyntaxOn
2635 \seq_gput_right:Nn
2636   \g_@@_renderers_seq
2637   { jekyllDataBoolean }
2638 \prop_gput:Nnn
2639   \g_@@_renderer_arities_prop
2640   { jekyllDataBoolean }
2641   { 2 }
2642 \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.

```
2643 \def\markdownRendererJekyllDataNumber{%
2644   \markdownRendererJekyllDataNumberPrototype}%
2645 \ExplSyntaxOn
2646 \seq_gput_right:Nn
2647   \g_@@_renderers_seq
2648   { jekyllDataNumber }
2649 \prop_gput:Nnn
2650   \g_@@_renderer_arities_prop
2651   { jekyllDataNumber }
2652   { 2 }
2653 \ExplSyntaxOff
```

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

```
2654 \def\markdownRendererJekyllDataString{%
2655   \markdownRendererJekyllDataStringPrototype}%
2656 \ExplSyntaxOn
2657 \seq_gput_right:Nn
2658   \g_@@_renderers_seq
2659   { jekyllDataString }
2660 \prop_gput:Nnn
2661   \g_@@_renderer_arities_prop
2662   { jekyllDataString }
2663   { 2 }
2664 \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.

```

2665 \def\markdownRendererJekyllDataEmpty{%
2666   \markdownRendererJekyllDataEmptyPrototype}%
2667 \ExplSyntaxOn
2668 \seq_gput_right:Nn
2669   \g_@@_renderers_seq
2670   { jekyllDataEmpty }
2671 \prop_gput:Nnn
2672   \g_@@_renderer_arities_prop
2673   { jekyllDataEmpty }
2674   { 1 }
2675 \ExplSyntaxOff

```

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

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

```

2676 \ExplSyntaxOn
2677 \cs_new:Nn \@@_define_renderers:
2678   {
2679     \seq_map_function:NN
2680       \g_@@_renderers_seq
2681       \@@_define_renderer:n
2682   }
2683 \cs_new:Nn \@@_define_renderer:n
2684   {
2685     \@@_renderer_tl_to_csname:nN
2686       { #1 }
2687       \l_tmpa_tl
2688     \prop_get:NnN
2689       \g_@@_renderer_arities_prop
2690       { #1 }
2691       \l_tmpb_tl
2692     \@@_define_renderer:ncV
2693       { #1 }
2694       { \l_tmpa_tl }
2695       \l_tmpb_tl
2696   }

```

```

2697 \cs_new:Nn \@@_renderer_tl_to_csname:nN
2698 {
2699     \tl_set:Nn
2700         \l_tmpa_tl
2701         { \str_uppercase:n { #1 } }
2702     \tl_set:Nx
2703         #2
2704         {
2705             markdownRenderer
2706             \tl_head:f { \l_tmpa_tl }
2707             \tl_tail:n { #1 }
2708         }
2709 }
2710 \tl_new:N
2711     \l_@@_renderer_definition_tl
2712 \cs_new:Nn \@@_define_renderer:nNn
2713 {
2714     \keys_define:nn
2715         { markdown/options/renderers }
2716         {
2717             #1 .code:n = {
2718                 \tl_set:Nn
2719                     \l_@@_renderer_definition_tl
2720                     { ##1 }
2721                 \regex_replace_all:nnN
2722                     { \cP\#0 }
2723                     { #1 }
2724                 \l_@@_renderer_definition_tl
2725                 \cs_generate_from_arg_count:NNnV
2726                     #2
2727                     \cs_set:Npn
2728                     { #3 }
2729                     \l_@@_renderer_definition_tl
2730             },
2731         }
2732 }
2733 \cs_generate_variant:Nn
2734     \@@_define_renderer:nNn
2735     { ncV }
2736 \cs_generate_variant:Nn
2737     \cs_generate_from_arg_count:NNnn
2738     { NNnV }
2739 \keys_define:nn
2740     { markdown/options }
2741     {
2742         renderers .code:n = {
2743             \keys_set:nn

```



```

2744         { markdown/options/renderers }
2745         { #1 }
2746     },
2747 }
2748 \ExplSyntaxOff

```

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.
  }
}

```

In addition to exact token renderer names, we also support wildcards and enumerations 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 parameter `#0`:

```

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

```

```

2749 \ExplSyntaxOn
2750 \prop_new:N
2751   \g_@@_glob_cache_prop
2752 \tl_new:N
2753   \l_@@_current_glob_tl
2754 \cs_new:Nn
2755   \@@_glob_seq:nnN
2756   {
2757     \tl_set:Nn
2758       \l_@@_current_glob_tl
2759       { ^ #1 $ }
2760     \prop_get:NeNTF
2761       \g_@@_glob_cache_prop
2762       { #2 / \l_@@_current_glob_tl }
2763     \l_tmpa_clist
2764     {
2765       \seq_set_from_clist:NN
2766         #3
2767       \l_tmpa_clist
2768     }
2769     {
2770       \seq_clear:N
2771         #3
2772       \regex_replace_all:nnN
2773         { \* }
2774         { .* }
2775         \l_@@_current_glob_tl
2776       \regex_set:NV
2777         \l_tmpa_regex
2778         \l_@@_current_glob_tl
2779       \seq_map_inline:cn
2780         { #2 }
2781         {
2782           \regex_match:NnT
2783             \l_tmpa_regex
2784             { ##1 }
2785           {
2786             \seq_put_right:Nn
2787               #3
2788               { ##1 }
2789           }
2790         }
2791       \clist_set_from_seq:NN
2792         \l_tmpa_clist
2793         #3
2794       \prop_gput:NeV
2795         \g_@@_glob_cache_prop

```

```

2796         { #2 / \l_@@_current_glob_tl }
2797         \l_tmpa_clist
2798     }
2799 }
2800 % TODO: Remove in TeX Live 2023.
2801 \prg_generate_conditional_variant:Nnn
2802   \prop_get:NnN
2803   { NeN }
2804   { TF }
2805 \cs_generate_variant:Nn
2806   \regex_set:Nn
2807   { NV }
2808 \cs_generate_variant:Nn
2809   \prop_gput:Nnn
2810   { NeV }
2811 \seq_new:N
2812   \l_@@_renderer_glob_results_seq
2813 \keys_define:nn
2814   { markdown/options/renderers }
2815   {
2816     unknown .code:n = {
2817       \@@_glob_seq:VnN
2818       \l_keys_key_str
2819       { g_@@_renderers_seq }
2820       \l_@@_renderer_glob_results_seq
2821       \seq_if_empty:NTF
2822       \l_@@_renderer_glob_results_seq
2823       {
2824         \msg_error:nnV
2825         { markdown }
2826         { undefined-renderer }
2827         \l_keys_key_str
2828       }
2829     }
2830     \tl_set:Nn
2831       \l_@@_renderer_definition_tl
2832       { #1 }
2833     \seq_map_inline:Nn
2834       \l_@@_renderer_glob_results_seq
2835       {
2836         \@@_renderer_tl_to_csname:nN
2837         { ##1 }
2838         \l_tmpa_tl
2839         \prop_get:NnN
2840         \g_@@_renderer_arities_prop
2841         { ##1 }
2842         \l_tmpb_tl

```

```

2843         \int_set:Nn
2844         \l_tmpa_int
2845         \l_tmpb_tl
2846     \tl_set:NV
2847         \l_tmpb_tl
2848         \l_@@_renderer_definition_tl
2849     \regex_replace_all:nnN
2850         { \cP\#0 }
2851         { ##1 }
2852         \l_tmpb_tl
2853     \cs_generate_from_arg_count:cNVV
2854         { \l_tmpa_tl }
2855     \cs_set:Npn
2856         \l_tmpa_int
2857         \l_tmpb_tl
2858     }
2859 }
2860 },
2861 }
2862 \msg_new:nnn
2863 { markdown }
2864 { undefined-renderer }
2865 {
2866     Renderer~#1~is~undefined.
2867 }
2868 \cs_generate_variant:Nn
2869 \@@_glob_seq:nnN
2870 { VnN }
2871 \cs_generate_variant:Nn
2872 \cs_generate_from_arg_count:NNnn
2873 { cNVV }
2874 \cs_generate_variant:Nn
2875 \msg_error:nnn
2876 { nnV }

```

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.

```

2877 \str_if_eq:VVT
2878 \c_@@_top_layer_tl
2879 \c_@@_option_layer_plain_tex_tl
2880 {
2881     \@@_define_renderers:
2882 }
2883 \ExplSyntaxOff

```

## 2.2.6 Token Renderer Prototypes

### 2.2.6.1 YAML Metadata Renderer Prototypes

By default, the renderer prototypes for YAML metadata provide a high-level interface that can be programmed using the `markdown/jekyllData` key-values from the `l3keys` module of the `LATEX3` kernel.

```
2884 \ExplSyntaxOn
2885 \keys_define:nn
2886   { markdown/jekyllData }
2887   { }
2888 \ExplSyntaxOff
```

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

```
2889 \ExplSyntaxOn
2890 \@@_with_various_cases:nn
2891   { jekyllDataRenderers }
2892   {
2893     \keys_define:nn
2894       { markdown/options }
2895       {
2896         #1 .code:n = {
2897           \tl_set:Nn
2898             \l_tmpa_tl
2899             { ##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.

```
2900         \tl_replace_all:NnV
2901         \l_tmpa_tl
2902         { / }
2903         \c_backslash_str
2904         \keys_set:nV
2905         { markdown/options/jekyll-data-renderers }
2906         \l_tmpa_tl
2907       },
2908     }
2909   }
2910 \keys_define:nn
2911   { markdown/options/jekyll-data-renderers }
2912   {
2913     unknown .code:n = {
2914       \tl_set_eq:NN
2915       \l_tmpa_tl

```

```

2916     \l_keys_key_str
2917     \tl_replace_all:NVn
2918     \l_tmpa_tl
2919     \c_backslash_str
2920     { / }
2921     \tl_put_right:Nn
2922     \l_tmpa_tl
2923     {
2924         .code:n = { #1 }
2925     }
2926     \keys_define:nV
2927     { markdown/jekyllData }
2928     \l_tmpa_tl
2929 }
2930 }
2931 \cs_generate_variant:Nn
2932 \keys_define:nn
2933 { nV }
2934 \ExplSyntaxOff

```

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

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

```

2935 \ExplSyntaxOn
2936 \cs_new:Nn \@@_define_renderer_prototypes:
2937 {
2938     \seq_map_function:NN
2939     \g_@@_renderers_seq
2940     \@@_define_renderer_prototype:n
2941 }
2942 \cs_new:Nn \@@_define_renderer_prototype:n
2943 {
2944     \@@_renderer_prototype_tl_to_csname:nN
2945     { #1 }
2946     \l_tmpa_tl
2947     \prop_get:NnN
2948     \g_@@_renderer_arities_prop
2949     { #1 }
2950     \l_tmpb_tl
2951     \@@_define_renderer_prototype:ncV
2952     { #1 }
2953     { \l_tmpa_tl }

```

```

2954     \l_tmpb_tl
2955   }
2956 \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
2957 {
2958   \tl_set:Nn
2959     \l_tmpa_tl
2960     { \str_uppercase:n { #1 } }
2961   \tl_set:Nx
2962     #2
2963     {
2964       markdownRenderer
2965       \tl_head:f { \l_tmpa_tl }
2966       \tl_tail:n { #1 }
2967       Prototype
2968     }
2969 }
2970 \tl_new:N
2971   \l_@@_renderer_prototype_definition_tl
2972 \cs_new:Nn \@@_define_renderer_prototype:nNn
2973 {
2974   \keys_define:nn
2975     { markdown/options/renderer-prototypes }
2976     {
2977       #1 .code:n = {
2978         \tl_set:Nn
2979           \l_@@_renderer_prototype_definition_tl
2980           { ##1 }
2981         \regex_replace_all:nnN
2982           { \cP\#0 }
2983           { #1 }
2984         \l_@@_renderer_prototype_definition_tl
2985         \cs_generate_from_arg_count:NNnV
2986           #2
2987           \cs_set:Npn
2988             { #3 }
2989             \l_@@_renderer_prototype_definition_tl
2990       },
2991     }

```

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

```

2992 \cs_if_free:NT
2993   #2
2994   {
2995     \cs_generate_from_arg_count:NNnn
2996       #2
2997       \cs_set:Npn

```

```

2998         { #3 }
2999     { }
3000 }
3001 }
3002 \cs_generate_variant:Nn
3003   \@@_define_renderer_prototype:nNn
3004   { ncV }
3005 \ExplSyntaxOff

```

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.
  }
}

```

In addition to exact token renderer 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 parameter `#0`:



```

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

```

```

3006 \ExplSyntaxOn
3007 \seq_new:N
3008   \l_@@_renderer_prototype_glob_results_seq
3009 \keys_define:nn
3010 { markdown/options/renderer-prototypes }
3011 {
3012   unknown .code:n = {
3013     \@@_glob_seq:VnN
3014     \l_keys_key_str
3015     { g_@@_renderers_seq }
3016     \l_@@_renderer_prototype_glob_results_seq
3017     \seq_if_empty:NTF
3018     \l_@@_renderer_prototype_glob_results_seq
3019     {
3020       \msg_error:nnV
3021       { markdown }
3022       { undefined-renderer-prototype }
3023       \l_keys_key_str
3024     }
3025   }
3026   \tl_set:Nn
3027     \l_@@_renderer_prototype_definition_tl
3028     { #1 }
3029   \seq_map_inline:Nn
3030     \l_@@_renderer_prototype_glob_results_seq
3031     {
3032       \@@_renderer_prototype_tl_to_csname:nN
3033       { ##1 }
3034       \l_tmpa_tl
3035       \prop_get:NnN
3036         \g_@@_renderer_arities_prop
3037         { ##1 }
3038         \l_tmpb_tl
3039       \int_set:Nn
3040         \l_tmpa_int
3041         \l_tmpb_tl
3042       \tl_set:NV
3043         \l_tmpb_tl
3044         \l_@@_renderer_prototype_definition_tl

```

```

3045         \regex_replace_all:nnN
3046         { \cP\#0 }
3047         { ##1 }
3048         \l_tmpb_tl
3049         \cs_generate_from_arg_count:cNVV
3050         { \l_tmpa_tl }
3051         \cs_set:Npn
3052         \l_tmpa_int
3053         \l_tmpb_tl
3054     }
3055 }
3056 },
3057 }
3058 \msg_new:nnn
3059 { markdown }
3060 { undefined-renderer-prototype }
3061 {
3062     Renderer~prototype~#1~is~undefined.
3063 }
3064 \@@_with_various_cases:nn
3065 { rendererPrototypes }
3066 {
3067     \keys_define:nn
3068     { markdown/options }
3069     {
3070         #1 .code:n = {
3071             \keys_set:nn
3072             { markdown/options/renderer-prototypes }
3073             { ##1 }
3074         },
3075     }
3076 }

```

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

```

3077 \str_if_eq:VVT
3078 \c_@@_top_layer_tl
3079 \c_@@_option_layer_plain_tex_tl
3080 {
3081     \@@_define_renderer_prototypes:
3082 }
3083 \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  $\text{\TeX}$  engine that does not support direct Lua access is starting to buffer a text. The plain  $\text{\TeX}$  implementation changes the category code of plain  $\text{\TeX}$  special characters to *other*, but there may be other active characters that may break the output. This macro should temporarily change the category of these to *other*.

```
3084 \let\markdownMakeOther\relax
```

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

```
3085 \let\markdownReadAndConvert\relax
```

```
3086 \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*.

```
3087 \catcode`\|=0\catcode`\=12%
```

```
3088 |gdef|markdownBegin{%
```

```
3089   |markdownReadAndConvert{\markdownEnd}%
```

```
3090                               {\|markdownEnd}}%
```

```
3091 |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.

```
3092 \ExplSyntaxOn
```

```
3093 \keys_define:nn
```

```
3094   { markdown/options }
```

```
3095   {
```

```
3096     code .code:n = { #1 },
```

```

3097   }
3098   \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.

```

3099 \ExplSyntaxOn
3100 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
3101 \cs_generate_variant:Nn
3102   \tl_const:Nn
3103   { NV }
3104 \tl_if_exist:NF
3105   \c_@@_top_layer_tl
3106   {
3107     \tl_const:NV
3108       \c_@@_top_layer_tl
3109       \c_@@_option_layer_latex_tl
3110   }
3111 \ExplSyntaxOff
3112 \input markdown/markdown

```

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

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

where *<options>* are the L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.2). Note that *<options>* inside the `\usepackage` macro may not set the `markdownRenderers` (see Section 2.2.5.44) 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 2.3.3) can be used, the extended variant that can load multiple themes and import snippets from them (see Section 2.2.4) cannot. This limitation is due to the way L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> parses package options.

### 2.3.1 Typesetting Markdown

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

The `markdown` and `markdown*` L<sup>A</sup>T<sub>E</sub>X environments are used to typeset markdown document fragments. Both L<sup>A</sup>T<sub>E</sub>X environments accept L<sup>A</sup>T<sub>E</sub>X interface options (see section 2.3.2) as the only argument. This argument is optional for the `markdown` environment and mandatory for the `markdown*` environment.

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

```
3113 \newenvironment{markdown}\relax\relax
3114 \newenvironment{markdown*}[1]\relax\relax
```

You may prepend your own code to the `\markdown` macro and append your own code to the `\markdownEnd` macro to produce special effects before and after the `markdown` L<sup>A</sup>T<sub>E</sub>X environment (and likewise for the starred version).

Note that the `markdown` and `markdown*` L<sup>A</sup>T<sub>E</sub>X environments are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros exposed by the plain T<sub>E</sub>X interface.

The following example L<sup>A</sup>T<sub>E</sub>X code showcases the usage of the `markdown` and `markdown*` environments:

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

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 2.3.2) 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:

|                                                      |
|------------------------------------------------------|
| <code>\documentclass{article}</code>                 |
| <code>\usepackage{markdown}</code>                   |
| <code>\begin{document}</code>                        |
| <code>\markdownInput[smartEllipses]{hello.md}</code> |
| <code>\end{document}</code>                          |

### 2.3.2 Options

The  $\LaTeX$  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.

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

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

#### 2.3.2.1 Finalizing and Freezing the Cache

To ensure compatibility with the `minted` package [9, Section 5.1], which supports the `finalizcache` and `frozenscache` package options with similar semantics to the `finalizeCache` and `frozenCache` plain  $\TeX$  options, the Markdown package also recognizes these as aliases and accepts them as document class options. By passing `finalizcache` and `frozenscache` as document class options, you may conveniently control the behavior of both packages at once:

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

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

```
3115 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
3116 \DeclareOption{frozenscache}{\markdownSetup{frozenCache}}
```

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

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

```
3117 \ExplSyntaxOn
3118 \str_if_eq:VVT
3119   \c_@@_top_layer_tl
3120   \c_@@_option_layer_latex_tl
```

```

3121 {
3122   \@@_define_option_commands_and_keyvals:
3123   \@@_define_renderers:
3124   \@@_define_renderer_prototypes:
3125 }
3126 \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,
}

```

### 2.3.3 Themes

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

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

If the  $\text{\LaTeX}$  option with keys `theme` or `import` is (repeatedly) specified in the `\usepackage` macro, the loading of the theme(s) will be postponed in first-in-first-out order until after the Markdown  $\text{\LaTeX}$  package has been loaded. Otherwise, the theme(s) will be loaded immediately. For example, there is a theme named `witiko/dot`, which typesets fenced code blocks with the `dot` infostring as images of directed graphs rendered by the Graphviz tools. The following code would first load the Markdown package, then the `markdownthemewitiko_beamer_MU.sty`  $\text{\LaTeX}$  package, and finally the `markdownthemewitiko_dot.sty`  $\text{\LaTeX}$  package:

```

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

```

```

3127 \newif\ifmarkdownLaTeXLoaded
3128 \markdownLaTeXLoadedfalse

```

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

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

**witiko/dot** A theme that typesets fenced code blocks with the `dot ...` infostring as images of directed graphs rendered by the Graphviz tools. The right tail of the infostring is used as the image title.

```

\documentclass{article}
\usepackage[import=witiko/dot]{markdown}
\setkeys{Gin}{
  width = \columnwidth,
  height = 0.65\paperheight,
  keepaspectratio}
\begin{document}
\begin{markdown}
``` dot Various formats of mathematical formulae
digraph tree {
  margin = 0;
  rankdir = "LR";

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

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

```

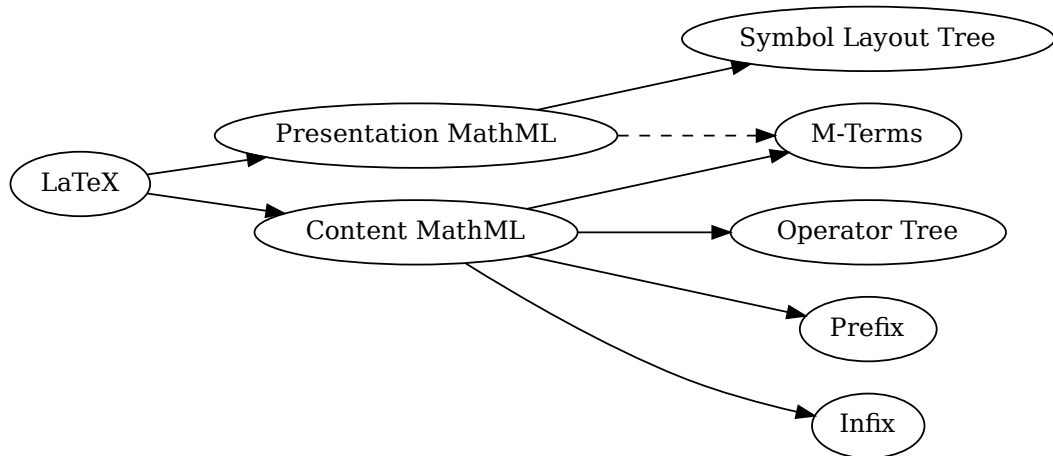


```

\end{markdown}
\end{document}

```

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



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

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

3129 \ProvidesPackage{markdownthemewitiko\_dot}[2021/03/09]%

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

```

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

```

Typesetting the above document produces the output shown in Figure 5. The theme requires the catchfile  $\text{\LaTeX}$  package and a Unix-like operating system with GNU Coreutils `md5sum` and either GNU Wget or cURL installed. 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 5: The banner of the Markdown package**

theme also requires shell access unless the `frozenCache` plain TeX option is enabled.

```
3130 \ProvidesPackage{markdownthemewitiko_graphicx_http}[2021/03/22]%
```

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

```

3131 \AtEndOfPackage{
3132   \markdownLaTeXLoadedtrue

```

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

```

3133   \markdownIfOption{noDefaults}{-}{-}{
3134     \markdownSetup{theme=witiko/markdown/defaults}
3135   }
3136 }

```

```
3137 \ProvidesPackage{markdownthemewitiko_markdown_defaults}[2024/01/03]%
```

Please, see Section 3.3.4 for implementation details of the built-in LaTeX themes.

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

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

```
3138 \ExplSyntaxOn
3139 \tl_const:Nn \c_@@_option_layer_context_tl { context }
3140 \cs_generate_variant:Nn
3141   \tl_const:Nn
3142   { NV }
3143 \tl_if_exist:NF
3144   \c_@@_top_layer_tl
3145   {
3146     \tl_const:NV
3147       \c_@@_top_layer_tl
3148       \c_@@_option_layer_context_tl
3149   }
3150 \ExplSyntaxOff
```

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

```
3151 \writestatus{loading}{ConTeXt User Module / markdown}%
3152 \startmodule[markdown]
3153 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
3154   \do\#\do\^\do\_do\%do\~}%
3155 \input markdown/markdown
```

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

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

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

### 2.4.1 Typesetting Markdown

The interface exposes the `\startmarkdown` and `\stopmarkdown` macro pair for the typesetting of a markdown document fragment, and defines the `\inputmarkdown` macro.

```
3156 \let\startmarkdown\relax
3157 \let\stopmarkdown\relax
3158 \let\inputmarkdown\relax
```

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

Note that the `\startmarkdown` and `\stopmarkdown` macros are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros exposed by the plain `TeX` interface.

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

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

The `\inputmarkdown` macro accepts a single mandatory parameter containing the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain `TeX`. Unlike the `\markdownInput` macro provided by the plain `TeX` interface, this macro also accepts `ConTeXt` interface options (see Section 2.4.2) as its optional argument. These options will only influence this markdown document.

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

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

## 2.4.2 Options

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

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

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

3159 \ExplSyntaxOn

```

3160 \cs_new:Npn
3161   \setupmarkdown
3162   [ #1 ]
3163   {
3164     \@@_setup:n
3165     { #1 }
3166   }
3167 \ExplSyntaxOff

```

### 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.

```

3168 \ExplSyntaxOn
3169 \cs_new:Nn \@@_caseless:N
3170   {
3171     \regex_replace_all:nnN
3172     { ([a-z])([A-Z]) }
3173     { \1 \c { str_lowercase:n } \cB\{ \2 \cE\} }
3174     #1
3175     \tl_set:Nx
3176     #1
3177     { #1 }
3178   }
3179 \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.

```

3180 \str_if_eq:VVT
3181   \c_@@_top_layer_tl
3182   \c_@@_option_layer_context_tl
3183   {
3184     \@@_define_option_commands_and_keyvals:
3185     \@@_define_renderers:
3186     \@@_define_renderer_prototypes:
3187   }
3188 \ExplSyntaxOff

```

### 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]
```

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.

```
3189 \startmodule[markdownthemewitiko_markdown_defaults]
3190 \unprotect
```

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

## 3 Implementation

This part of the documentation describes the implementation of the interfaces exposed by the package (see Section 2) and is aimed at the developers of the package, as well as the curious users.

Figure 1 shows the high-level structure of the Markdown package: The translation from markdown to T<sub>E</sub>X *token renderers* is performed by the Lua layer. The plain T<sub>E</sub>X layer provides default definitions for the token renderers. The L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt layers correct idiosyncrasies of the respective T<sub>E</sub>X formats, and provide format-specific default definitions for the token renderers.

### 3.1 Lua Implementation

The Lua implementation implements `writer` and `reader` objects, which provide the conversion from markdown to plain T<sub>E</sub>X, and `extensions` objects, which provide syntax extensions for the `writer` and `reader` objects.

The Lunamark Lua module implements writers for the conversion to various other formats, such as DocBook, Groff, or HTML. These were stripped from the module

and the remaining markdown reader and plain T<sub>E</sub>X writer were hidden behind the converter functions exposed by the Lua interface (see Section 2.1).

```

3191 local upper, format, length =
3192     string.upper, string.format, string.len
3193 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, Cp, any =
3194     lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
3195     lpeg.Cmt, lpeg.Cc, lpeg.Ct, lpeg.B, lpeg.Cs, lpeg.Cp, lpeg.P(1)

```

### 3.1.1 Utility Functions

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

```

3196 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.

```

3197 function util.err(msg, exit_code)
3198     io.stderr:write("markdown.lua: " .. msg .. "\n")
3199     os.exit(exit_code or 1)
3200 end

```

The `util.cache` method computes the digest of `string` and `salt`, adds the `suffix` and looks into the directory `dir`, whether a file with such a name exists. If it does not, it gets created with `transform(string)` as its content. The filename is then returned.

```

3201 function util.cache(dir, string, salt, transform, suffix)
3202     local digest = md5.sumhexa(string .. (salt or ""))
3203     local name = util.pathname(dir, digest .. suffix)
3204     local file = io.open(name, "r")
3205     if file == nil then -- If no cache entry exists, then create a new one.
3206         file = assert(io.open(name, "w"),
3207             [[Could not open file ]] .. name .. [[ for writing]])
3208         local result = string
3209         if transform ~= nil then
3210             result = transform(result)
3211         end
3212         assert(file:write(result))
3213         assert(file:close())
3214     end
3215     return name
3216 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.

```

3217 function util.cache_verbatim(dir, string)
3218   local name = util.cache(dir, string, nil, nil, ".verbatim")
3219   return name
3220 end

```

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

```

3221 function util.table_copy(t)
3222   local u = { }
3223   for k, v in pairs(t) do u[k] = v end
3224   return setmetatable(u, getmetatable(t))
3225 end

```

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

```

3226 function util.encode_json_string(s)
3227   s = s:gsub([[\\]], [[\\]])
3228   s = s:gsub([["]], [[\"]])
3229   return [["]] .. s .. [["]]
3230 end

```

The `util.lookup_files` method looks up files with filename `f` and returns their paths. Further options for the Kpathsea library can be specified in table `options`. [1, Section 10.7.4]

```

3231 function util.lookup_files(f, options)
3232   return kpse.lookup(f, options)
3233 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 [10, Chapter 21].

```

3234 function util.expand_tabs_in_line(s, tabstop)
3235   local tab = tabstop or 4
3236   local corr = 0
3237   return (s:gsub(")\t", function(p)
3238     local sp = tab - (p - 1 + corr) % tab
3239     corr = corr - 1 + sp
3240     return string.rep(" ", sp)
3241   end))
3242 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.

```

3243 function util.walk(t, f)
3244   local typ = type(t)
3245   if typ == "string" then
3246     f(t)

```



```

3247 elseif typ == "table" then
3248     local i = 1
3249     local n
3250     n = t[i]
3251     while n do
3252         util.walk(n, f)
3253         i = i + 1
3254         n = t[i]
3255     end
3256 elseif typ == "function" then
3257     local ok, val = pcall(t)
3258     if ok then
3259         util.walk(val, f)
3260     end
3261 else
3262     f(tostring(t))
3263 end
3264 end

```

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

```

3265 function util.flatten(ary)
3266     local new = {}
3267     for _,v in ipairs(ary) do
3268         if type(v) == "table" then
3269             for _,w in ipairs(util.flatten(v)) do
3270                 new[#new + 1] = w
3271             end
3272         else
3273             new[#new + 1] = v
3274         end
3275     end
3276     return new
3277 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.

```

3278 function util.rope_to_string(rope)
3279     local buffer = {}
3280     util.walk(rope, function(x) buffer[#buffer + 1] = x end)
3281     return table.concat(buffer)
3282 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.

```

3283 function util.rope_last(rope)
3284     if #rope == 0 then
3285         return nil

```

```

3286     else
3287         local l = rope[#rope]
3288         if type(l) == "table" then
3289             return util.rope_last(l)
3290         else
3291             return l
3292         end
3293     end
3294 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  $1 \leq i \leq \#ary$ .

```

3295 function util.intersperse(ary, x)
3296     local new = {}
3297     local l = #ary
3298     for i,v in ipairs(ary) do
3299         local n = #new
3300         new[n + 1] = v
3301         if i ~= l then
3302             new[n + 2] = x
3303         end
3304     end
3305     return new
3306 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  $1 \leq i \leq \#ary$ .

```

3307 function util.map(ary, f)
3308     local new = {}
3309     for i,v in ipairs(ary) do
3310         new[i] = f(v)
3311     end
3312     return new
3313 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.

```

3314 function util.escaper(char_escapes, string_escapes)

```

Build a string of escapable characters.

```

3315     local char_escapes_list = ""
3316     for i,_ in pairs(char_escapes) do
3317         char_escapes_list = char_escapes_list .. i
3318     end

```

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

```
3319 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.

```
3320 if string_escapes then
3321   for k,v in pairs(string_escapes) do
3322     escapable = P(k) / v + escapable
3323   end
3324 end
```

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

```
3325 local escape_string = Cs((escapable + any)^0)
```

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

```
3326 return function(s)
3327   return lpeg.match(escape_string, s)
3328 end
3329 end
```

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

```
3330 function util.pathname(dir, file)
3331   if #dir == 0 then
3332     return file
3333   else
3334     return dir .. "/" .. file
3335   end
3336 end
```

### 3.1.2 HTML Entities

This section documents the HTML entities recognized by the markdown reader. These functions are encapsulated in the `entities` object. The functions were originally located in the `lunamark/entities.lua` file in the Lunamark Lua module.

```
3337 local entities = {}
3338
```

```

3339 local character_entities = {
3340     ["Tab"] = 9,
3341     ["NewLine"] = 10,
3342     ["excl"] = 33,
3343     ["QUOT"] = 34,
3344     ["quot"] = 34,
3345     ["num"] = 35,
3346     ["dollar"] = 36,
3347     ["percnt"] = 37,
3348     ["AMP"] = 38,
3349     ["amp"] = 38,
3350     ["apos"] = 39,
3351     ["lpar"] = 40,
3352     ["rpar"] = 41,
3353     ["ast"] = 42,
3354     ["midast"] = 42,
3355     ["plus"] = 43,
3356     ["comma"] = 44,
3357     ["period"] = 46,
3358     ["sol"] = 47,
3359     ["colon"] = 58,
3360     ["semi"] = 59,
3361     ["LT"] = 60,
3362     ["lt"] = 60,
3363     ["nvlt"] = {60, 8402},
3364     ["bne"] = {61, 8421},
3365     ["equals"] = 61,
3366     ["GT"] = 62,
3367     ["gt"] = 62,
3368     ["nvgt"] = {62, 8402},
3369     ["quest"] = 63,
3370     ["commat"] = 64,
3371     ["lbrack"] = 91,
3372     ["lsqb"] = 91,
3373     ["bsol"] = 92,
3374     ["rbrack"] = 93,
3375     ["rsqb"] = 93,
3376     ["Hat"] = 94,
3377     ["UnderBar"] = 95,
3378     ["lowbar"] = 95,
3379     ["DiacriticalGrave"] = 96,
3380     ["grave"] = 96,
3381     ["fjlig"] = {102, 106},
3382     ["lbrace"] = 123,
3383     ["lcub"] = 123,
3384     ["VerticalLine"] = 124,
3385     ["verbar"] = 124,

```

```

3386 ["vert"] = 124,
3387 ["rbrace"] = 125,
3388 ["rcub"] = 125,
3389 ["NonBreakingSpace"] = 160,
3390 ["nbsp"] = 160,
3391 ["iexcl"] = 161,
3392 ["cent"] = 162,
3393 ["pound"] = 163,
3394 ["curren"] = 164,
3395 ["yen"] = 165,
3396 ["brvbar"] = 166,
3397 ["sect"] = 167,
3398 ["Dot"] = 168,
3399 ["DoubleDot"] = 168,
3400 ["die"] = 168,
3401 ["uml"] = 168,
3402 ["COPY"] = 169,
3403 ["copy"] = 169,
3404 ["ordf"] = 170,
3405 ["laquo"] = 171,
3406 ["not"] = 172,
3407 ["shy"] = 173,
3408 ["REG"] = 174,
3409 ["circledR"] = 174,
3410 ["reg"] = 174,
3411 ["macr"] = 175,
3412 ["strns"] = 175,
3413 ["deg"] = 176,
3414 ["PlusMinus"] = 177,
3415 ["plusmn"] = 177,
3416 ["pm"] = 177,
3417 ["sup2"] = 178,
3418 ["sup3"] = 179,
3419 ["DiacriticalAcute"] = 180,
3420 ["acute"] = 180,
3421 ["micro"] = 181,
3422 ["para"] = 182,
3423 ["CenterDot"] = 183,
3424 ["centerdot"] = 183,
3425 ["middot"] = 183,
3426 ["Cedilla"] = 184,
3427 ["cedil"] = 184,
3428 ["sup1"] = 185,
3429 ["ordm"] = 186,
3430 ["raquo"] = 187,
3431 ["frac14"] = 188,
3432 ["frac12"] = 189,

```

3433 ["half"] = 189,  
 3434 ["frac34"] = 190,  
 3435 ["iquest"] = 191,  
 3436 ["Agrave"] = 192,  
 3437 ["Aacute"] = 193,  
 3438 ["Acirc"] = 194,  
 3439 ["Atilde"] = 195,  
 3440 ["Auml"] = 196,  
 3441 ["Aring"] = 197,  
 3442 ["angst"] = 197,  
 3443 ["AElig"] = 198,  
 3444 ["Ccedil"] = 199,  
 3445 ["Egrave"] = 200,  
 3446 ["Eacute"] = 201,  
 3447 ["Ecirc"] = 202,  
 3448 ["Euml"] = 203,  
 3449 ["Igrave"] = 204,  
 3450 ["Iacute"] = 205,  
 3451 ["Icirc"] = 206,  
 3452 ["Iuml"] = 207,  
 3453 ["ETH"] = 208,  
 3454 ["Ntilde"] = 209,  
 3455 ["Ograve"] = 210,  
 3456 ["Oacute"] = 211,  
 3457 ["Ocirc"] = 212,  
 3458 ["Otilde"] = 213,  
 3459 ["Ouml"] = 214,  
 3460 ["times"] = 215,  
 3461 ["Oslash"] = 216,  
 3462 ["Ugrave"] = 217,  
 3463 ["Uacute"] = 218,  
 3464 ["Ucirc"] = 219,  
 3465 ["Uuml"] = 220,  
 3466 ["Yacute"] = 221,  
 3467 ["THORN"] = 222,  
 3468 ["szlig"] = 223,  
 3469 ["agrave"] = 224,  
 3470 ["aacute"] = 225,  
 3471 ["acirc"] = 226,  
 3472 ["atilde"] = 227,  
 3473 ["auml"] = 228,  
 3474 ["aring"] = 229,  
 3475 ["aelig"] = 230,  
 3476 ["ccedil"] = 231,  
 3477 ["egrave"] = 232,  
 3478 ["eacute"] = 233,  
 3479 ["ecirc"] = 234,

```

3480 ["euml"] = 235,
3481 ["igrave"] = 236,
3482 ["iacute"] = 237,
3483 ["icirc"] = 238,
3484 ["iuml"] = 239,
3485 ["eth"] = 240,
3486 ["ntilde"] = 241,
3487 ["ograve"] = 242,
3488 ["oacute"] = 243,
3489 ["ocirc"] = 244,
3490 ["otilde"] = 245,
3491 ["ouml"] = 246,
3492 ["div"] = 247,
3493 ["divide"] = 247,
3494 ["oslash"] = 248,
3495 ["ugrave"] = 249,
3496 ["uacute"] = 250,
3497 ["ucirc"] = 251,
3498 ["uuml"] = 252,
3499 ["yacute"] = 253,
3500 ["thorn"] = 254,
3501 ["yuml"] = 255,
3502 ["Amacr"] = 256,
3503 ["amacr"] = 257,
3504 ["Abreve"] = 258,
3505 ["abreve"] = 259,
3506 ["Aogon"] = 260,
3507 ["aogon"] = 261,
3508 ["Cacute"] = 262,
3509 ["cacute"] = 263,
3510 ["Ccirc"] = 264,
3511 ["ccirc"] = 265,
3512 ["Cdot"] = 266,
3513 ["cdot"] = 267,
3514 ["Ccaron"] = 268,
3515 ["ccaron"] = 269,
3516 ["Dcaron"] = 270,
3517 ["dcaron"] = 271,
3518 ["Dstrok"] = 272,
3519 ["dstrok"] = 273,
3520 ["Emacr"] = 274,
3521 ["emacr"] = 275,
3522 ["Edot"] = 278,
3523 ["edot"] = 279,
3524 ["Eogon"] = 280,
3525 ["eogon"] = 281,
3526 ["Ecaron"] = 282,

```

```

3527 ["ecaron"] = 283,
3528 ["Gcirc"] = 284,
3529 ["gcirc"] = 285,
3530 ["Gbreve"] = 286,
3531 ["gbreve"] = 287,
3532 ["Gdot"] = 288,
3533 ["gdot"] = 289,
3534 ["Gcedil"] = 290,
3535 ["Hcirc"] = 292,
3536 ["hcirc"] = 293,
3537 ["Hstrook"] = 294,
3538 ["hstrook"] = 295,
3539 ["Itilde"] = 296,
3540 ["itilde"] = 297,
3541 ["Imacr"] = 298,
3542 ["imacr"] = 299,
3543 ["Iogon"] = 302,
3544 ["iogon"] = 303,
3545 ["Idot"] = 304,
3546 ["imath"] = 305,
3547 ["inodot"] = 305,
3548 ["IJlig"] = 306,
3549 ["ijlig"] = 307,
3550 ["Jcirc"] = 308,
3551 ["jcirc"] = 309,
3552 ["Kcedil"] = 310,
3553 ["kcedil"] = 311,
3554 ["kgreen"] = 312,
3555 ["Lacute"] = 313,
3556 ["lacute"] = 314,
3557 ["Lcedil"] = 315,
3558 ["lcedil"] = 316,
3559 ["Lcaron"] = 317,
3560 ["lcaron"] = 318,
3561 ["Lmidot"] = 319,
3562 ["lmidot"] = 320,
3563 ["Lstrook"] = 321,
3564 ["lstrook"] = 322,
3565 ["Nacute"] = 323,
3566 ["nacute"] = 324,
3567 ["Ncedil"] = 325,
3568 ["ncedil"] = 326,
3569 ["Ncaron"] = 327,
3570 ["ncaron"] = 328,
3571 ["napos"] = 329,
3572 ["ENG"] = 330,
3573 ["eng"] = 331,

```



```

3574 ["Omacr"] = 332,
3575 ["omacr"] = 333,
3576 ["Odblac"] = 336,
3577 ["odblac"] = 337,
3578 ["OElig"] = 338,
3579 ["oelig"] = 339,
3580 ["Racute"] = 340,
3581 ["racute"] = 341,
3582 ["Rcedil"] = 342,
3583 ["rcedil"] = 343,
3584 ["Rcaron"] = 344,
3585 ["rcaron"] = 345,
3586 ["Sacute"] = 346,
3587 ["sacute"] = 347,
3588 ["Scirc"] = 348,
3589 ["scirc"] = 349,
3590 ["Scedil"] = 350,
3591 ["scedil"] = 351,
3592 ["Scaron"] = 352,
3593 ["scaron"] = 353,
3594 ["Tcedil"] = 354,
3595 ["tcedil"] = 355,
3596 ["Tcaron"] = 356,
3597 ["tcaron"] = 357,
3598 ["Tstrok"] = 358,
3599 ["tstrok"] = 359,
3600 ["Utilde"] = 360,
3601 ["utilde"] = 361,
3602 ["Umacr"] = 362,
3603 ["umacr"] = 363,
3604 ["Ubreve"] = 364,
3605 ["ubreve"] = 365,
3606 ["Uring"] = 366,
3607 ["uring"] = 367,
3608 ["Udblac"] = 368,
3609 ["udblac"] = 369,
3610 ["Uogon"] = 370,
3611 ["uogon"] = 371,
3612 ["Wcirc"] = 372,
3613 ["wcirc"] = 373,
3614 ["Ycirc"] = 374,
3615 ["ycirc"] = 375,
3616 ["Yuml"] = 376,
3617 ["Zacute"] = 377,
3618 ["zacute"] = 378,
3619 ["Zdot"] = 379,
3620 ["zdot"] = 380,

```

```

3621 ["Zcaron"] = 381,
3622 ["zcaron"] = 382,
3623 ["fnof"] = 402,
3624 ["imped"] = 437,
3625 ["gacute"] = 501,
3626 ["jmath"] = 567,
3627 ["circ"] = 710,
3628 ["Hacek"] = 711,
3629 ["caron"] = 711,
3630 ["Breve"] = 728,
3631 ["breve"] = 728,
3632 ["DiacriticalDot"] = 729,
3633 ["dot"] = 729,
3634 ["ring"] = 730,
3635 ["ogon"] = 731,
3636 ["DiacriticalTilde"] = 732,
3637 ["tilde"] = 732,
3638 ["DiacriticalDoubleAcute"] = 733,
3639 ["dblac"] = 733,
3640 ["DownBreve"] = 785,
3641 ["Alpha"] = 913,
3642 ["Beta"] = 914,
3643 ["Gamma"] = 915,
3644 ["Delta"] = 916,
3645 ["Epsilon"] = 917,
3646 ["Zeta"] = 918,
3647 ["Eta"] = 919,
3648 ["Theta"] = 920,
3649 ["Iota"] = 921,
3650 ["Kappa"] = 922,
3651 ["Lambda"] = 923,
3652 ["Mu"] = 924,
3653 ["Nu"] = 925,
3654 ["Xi"] = 926,
3655 ["Omicron"] = 927,
3656 ["Pi"] = 928,
3657 ["Rho"] = 929,
3658 ["Sigma"] = 931,
3659 ["Tau"] = 932,
3660 ["Upsilon"] = 933,
3661 ["Phi"] = 934,
3662 ["Chi"] = 935,
3663 ["Psi"] = 936,
3664 ["Omega"] = 937,
3665 ["ohm"] = 937,
3666 ["alpha"] = 945,
3667 ["beta"] = 946,

```

```

3668 ["gamma"] = 947,
3669 ["delta"] = 948,
3670 ["epsi"] = 949,
3671 ["epsilon"] = 949,
3672 ["zeta"] = 950,
3673 ["eta"] = 951,
3674 ["theta"] = 952,
3675 ["iota"] = 953,
3676 ["kappa"] = 954,
3677 ["lambda"] = 955,
3678 ["mu"] = 956,
3679 ["nu"] = 957,
3680 ["xi"] = 958,
3681 ["omicron"] = 959,
3682 ["pi"] = 960,
3683 ["rho"] = 961,
3684 ["sigmaf"] = 962,
3685 ["sigmav"] = 962,
3686 ["varsigma"] = 962,
3687 ["sigma"] = 963,
3688 ["tau"] = 964,
3689 ["upsi"] = 965,
3690 ["upsilon"] = 965,
3691 ["phi"] = 966,
3692 ["chi"] = 967,
3693 ["psi"] = 968,
3694 ["omega"] = 969,
3695 ["thetasym"] = 977,
3696 ["thetav"] = 977,
3697 ["vartheta"] = 977,
3698 ["Upsi"] = 978,
3699 ["upsih"] = 978,
3700 ["phiv"] = 981,
3701 ["straightphi"] = 981,
3702 ["varphi"] = 981,
3703 ["piv"] = 982,
3704 ["varpi"] = 982,
3705 ["Gammad"] = 988,
3706 ["digamma"] = 989,
3707 ["gammad"] = 989,
3708 ["kappav"] = 1008,
3709 ["varkappa"] = 1008,
3710 ["rhov"] = 1009,
3711 ["varrho"] = 1009,
3712 ["epsiv"] = 1013,
3713 ["straightepsilon"] = 1013,
3714 ["varepsilon"] = 1013,

```

```

3715 ["backepsilon"] = 1014,
3716 ["bepsi"] = 1014,
3717 ["IOcy"] = 1025,
3718 ["DJcy"] = 1026,
3719 ["GJcy"] = 1027,
3720 ["Jukcy"] = 1028,
3721 ["DScy"] = 1029,
3722 ["Iukcy"] = 1030,
3723 ["YIcy"] = 1031,
3724 ["Jsercy"] = 1032,
3725 ["LJcy"] = 1033,
3726 ["NJcy"] = 1034,
3727 ["TSHcy"] = 1035,
3728 ["KJcy"] = 1036,
3729 ["Ubrcy"] = 1038,
3730 ["DZcy"] = 1039,
3731 ["Acy"] = 1040,
3732 ["Bcy"] = 1041,
3733 ["Vcy"] = 1042,
3734 ["Gcy"] = 1043,
3735 ["Dcy"] = 1044,
3736 ["IEcy"] = 1045,
3737 ["ZHcy"] = 1046,
3738 ["Zcy"] = 1047,
3739 ["Icy"] = 1048,
3740 ["Jcy"] = 1049,
3741 ["Kcy"] = 1050,
3742 ["Lcy"] = 1051,
3743 ["Mcy"] = 1052,
3744 ["Ncy"] = 1053,
3745 ["Ocy"] = 1054,
3746 ["Pcy"] = 1055,
3747 ["Rcy"] = 1056,
3748 ["Scy"] = 1057,
3749 ["Tcy"] = 1058,
3750 ["Ucy"] = 1059,
3751 ["Fcy"] = 1060,
3752 ["KHcy"] = 1061,
3753 ["TScy"] = 1062,
3754 ["CHcy"] = 1063,
3755 ["SHcy"] = 1064,
3756 ["SHCHcy"] = 1065,
3757 ["HARDcy"] = 1066,
3758 ["Ycy"] = 1067,
3759 ["SOFTcy"] = 1068,
3760 ["Ecy"] = 1069,
3761 ["YUcy"] = 1070,

```

```

3762 ["YAcy"] = 1071,
3763 ["acy"] = 1072,
3764 ["bcy"] = 1073,
3765 ["vcy"] = 1074,
3766 ["gcy"] = 1075,
3767 ["dcy"] = 1076,
3768 ["iecy"] = 1077,
3769 ["zhcy"] = 1078,
3770 ["zcy"] = 1079,
3771 ["icy"] = 1080,
3772 ["jcy"] = 1081,
3773 ["kcy"] = 1082,
3774 ["lcy"] = 1083,
3775 ["mcy"] = 1084,
3776 ["ncy"] = 1085,
3777 ["ocy"] = 1086,
3778 ["pcy"] = 1087,
3779 ["rcy"] = 1088,
3780 ["scy"] = 1089,
3781 ["tcy"] = 1090,
3782 ["ucy"] = 1091,
3783 ["fcy"] = 1092,
3784 ["khcy"] = 1093,
3785 ["tscy"] = 1094,
3786 ["chcy"] = 1095,
3787 ["shcy"] = 1096,
3788 ["shchcy"] = 1097,
3789 ["hardcy"] = 1098,
3790 ["ycy"] = 1099,
3791 ["softcy"] = 1100,
3792 ["ecy"] = 1101,
3793 ["yucy"] = 1102,
3794 ["yacy"] = 1103,
3795 ["iocy"] = 1105,
3796 ["djcy"] = 1106,
3797 ["gjcy"] = 1107,
3798 ["jukcy"] = 1108,
3799 ["dscy"] = 1109,
3800 ["iukcy"] = 1110,
3801 ["yicy"] = 1111,
3802 ["jsercy"] = 1112,
3803 ["ljcy"] = 1113,
3804 ["njcy"] = 1114,
3805 ["tshcy"] = 1115,
3806 ["kjcy"] = 1116,
3807 ["ubrcy"] = 1118,
3808 ["dzcy"] = 1119,

```

```

3809 ["ensp"] = 8194,
3810 ["emsp"] = 8195,
3811 ["emsp13"] = 8196,
3812 ["emsp14"] = 8197,
3813 ["numsp"] = 8199,
3814 ["puncsp"] = 8200,
3815 ["ThinSpace"] = 8201,
3816 ["thinsp"] = 8201,
3817 ["VeryThinSpace"] = 8202,
3818 ["hairsp"] = 8202,
3819 ["NegativeMediumSpace"] = 8203,
3820 ["NegativeThickSpace"] = 8203,
3821 ["NegativeThinSpace"] = 8203,
3822 ["NegativeVeryThinSpace"] = 8203,
3823 ["ZeroWidthSpace"] = 8203,
3824 ["zwnj"] = 8204,
3825 ["zwj"] = 8205,
3826 ["lrm"] = 8206,
3827 ["rlm"] = 8207,
3828 ["dash"] = 8208,
3829 ["hyphen"] = 8208,
3830 ["ndash"] = 8211,
3831 ["mdash"] = 8212,
3832 ["horbar"] = 8213,
3833 ["Verbar"] = 8214,
3834 ["Vert"] = 8214,
3835 ["OpenCurlyQuote"] = 8216,
3836 ["lsquo"] = 8216,
3837 ["CloseCurlyQuote"] = 8217,
3838 ["rsquo"] = 8217,
3839 ["rsquor"] = 8217,
3840 ["lsquor"] = 8218,
3841 ["sbquo"] = 8218,
3842 ["OpenCurlyDoubleQuote"] = 8220,
3843 ["ldquo"] = 8220,
3844 ["CloseCurlyDoubleQuote"] = 8221,
3845 ["rdquo"] = 8221,
3846 ["rdquor"] = 8221,
3847 ["bdquo"] = 8222,
3848 ["ldquor"] = 8222,
3849 ["dagger"] = 8224,
3850 ["Dagger"] = 8225,
3851 ["ddagger"] = 8225,
3852 ["bull"] = 8226,
3853 ["bullet"] = 8226,
3854 ["nldr"] = 8229,
3855 ["hellip"] = 8230,

```

```

3856 ["mldr"] = 8230,
3857 ["permil"] = 8240,
3858 ["pertenk"] = 8241,
3859 ["prime"] = 8242,
3860 ["Prime"] = 8243,
3861 ["tprime"] = 8244,
3862 ["backprime"] = 8245,
3863 ["bprime"] = 8245,
3864 ["lsaquo"] = 8249,
3865 ["rsaquo"] = 8250,
3866 ["OverBar"] = 8254,
3867 ["oline"] = 8254,
3868 ["caret"] = 8257,
3869 ["hybull"] = 8259,
3870 ["frasl"] = 8260,
3871 ["bsemi"] = 8271,
3872 ["qprime"] = 8279,
3873 ["MediumSpace"] = 8287,
3874 ["ThickSpace"] = {8287, 8202},
3875 ["NoBreak"] = 8288,
3876 ["ApplyFunction"] = 8289,
3877 ["af"] = 8289,
3878 ["InvisibleTimes"] = 8290,
3879 ["it"] = 8290,
3880 ["InvisibleComma"] = 8291,
3881 ["ic"] = 8291,
3882 ["euro"] = 8364,
3883 ["TripleDot"] = 8411,
3884 ["tdot"] = 8411,
3885 ["DotDot"] = 8412,
3886 ["Copf"] = 8450,
3887 ["complexes"] = 8450,
3888 ["incare"] = 8453,
3889 ["gscr"] = 8458,
3890 ["HilbertSpace"] = 8459,
3891 ["Hscr"] = 8459,
3892 ["hamilt"] = 8459,
3893 ["Hfr"] = 8460,
3894 ["Poincareplane"] = 8460,
3895 ["Hopf"] = 8461,
3896 ["quaternions"] = 8461,
3897 ["planckh"] = 8462,
3898 ["hbar"] = 8463,
3899 ["hslash"] = 8463,
3900 ["planck"] = 8463,
3901 ["plankv"] = 8463,
3902 ["Iscr"] = 8464,

```

```

3903 ["imagline"] = 8464,
3904 ["Ifr"] = 8465,
3905 ["Im"] = 8465,
3906 ["image"] = 8465,
3907 ["imagpart"] = 8465,
3908 ["Laplacetrif"] = 8466,
3909 ["Lscr"] = 8466,
3910 ["lagran"] = 8466,
3911 ["ell"] = 8467,
3912 ["Nopf"] = 8469,
3913 ["naturals"] = 8469,
3914 ["numero"] = 8470,
3915 ["copysr"] = 8471,
3916 ["weierp"] = 8472,
3917 ["wp"] = 8472,
3918 ["Popf"] = 8473,
3919 ["primes"] = 8473,
3920 ["Qopf"] = 8474,
3921 ["rationals"] = 8474,
3922 ["Rscr"] = 8475,
3923 ["realine"] = 8475,
3924 ["Re"] = 8476,
3925 ["Rfr"] = 8476,
3926 ["real"] = 8476,
3927 ["realpart"] = 8476,
3928 ["Ropf"] = 8477,
3929 ["reals"] = 8477,
3930 ["rx"] = 8478,
3931 ["TRADE"] = 8482,
3932 ["trade"] = 8482,
3933 ["Zopf"] = 8484,
3934 ["integers"] = 8484,
3935 ["mho"] = 8487,
3936 ["Zfr"] = 8488,
3937 ["zeetrif"] = 8488,
3938 ["iiota"] = 8489,
3939 ["Bernoullis"] = 8492,
3940 ["Bscr"] = 8492,
3941 ["bernou"] = 8492,
3942 ["Cayleys"] = 8493,
3943 ["Cfr"] = 8493,
3944 ["escr"] = 8495,
3945 ["Escr"] = 8496,
3946 ["expectation"] = 8496,
3947 ["Fouriertrif"] = 8497,
3948 ["Fscr"] = 8497,
3949 ["Mellintrif"] = 8499,

```



```

3950 ["Mscr"] = 8499,
3951 ["phmmat"] = 8499,
3952 ["order"] = 8500,
3953 ["orderof"] = 8500,
3954 ["oscr"] = 8500,
3955 ["alefsym"] = 8501,
3956 ["aleph"] = 8501,
3957 ["beth"] = 8502,
3958 ["gimel"] = 8503,
3959 ["daleth"] = 8504,
3960 ["CapitalDifferentialD"] = 8517,
3961 ["DD"] = 8517,
3962 ["DifferentialD"] = 8518,
3963 ["dd"] = 8518,
3964 ["ExponentialE"] = 8519,
3965 ["ee"] = 8519,
3966 ["exponentiale"] = 8519,
3967 ["ImaginaryI"] = 8520,
3968 ["ii"] = 8520,
3969 ["frac13"] = 8531,
3970 ["frac23"] = 8532,
3971 ["frac15"] = 8533,
3972 ["frac25"] = 8534,
3973 ["frac35"] = 8535,
3974 ["frac45"] = 8536,
3975 ["frac16"] = 8537,
3976 ["frac56"] = 8538,
3977 ["frac18"] = 8539,
3978 ["frac38"] = 8540,
3979 ["frac58"] = 8541,
3980 ["frac78"] = 8542,
3981 ["LeftArrow"] = 8592,
3982 ["ShortLeftArrow"] = 8592,
3983 ["larr"] = 8592,
3984 ["leftarrow"] = 8592,
3985 ["slarr"] = 8592,
3986 ["ShortUpArrow"] = 8593,
3987 ["UpArrow"] = 8593,
3988 ["uarr"] = 8593,
3989 ["uparrow"] = 8593,
3990 ["RightArrow"] = 8594,
3991 ["ShortRightArrow"] = 8594,
3992 ["rarr"] = 8594,
3993 ["rightarrow"] = 8594,
3994 ["srarr"] = 8594,
3995 ["DownArrow"] = 8595,
3996 ["ShortDownArrow"] = 8595,

```

```

3997 ["darr"] = 8595,
3998 ["downarrow"] = 8595,
3999 ["LeftRightArrow"] = 8596,
4000 ["harr"] = 8596,
4001 ["leftrightharpoon"] = 8596,
4002 ["UpDownArrow"] = 8597,
4003 ["updownarrow"] = 8597,
4004 ["varr"] = 8597,
4005 ["UpperLeftArrow"] = 8598,
4006 ["nwarr"] = 8598,
4007 ["nwarpoon"] = 8598,
4008 ["UpperRightArrow"] = 8599,
4009 ["nearr"] = 8599,
4010 ["nearrow"] = 8599,
4011 ["LowerRightArrow"] = 8600,
4012 ["searr"] = 8600,
4013 ["searrow"] = 8600,
4014 ["LowerLeftArrow"] = 8601,
4015 ["swarr"] = 8601,
4016 ["swarrow"] = 8601,
4017 ["nlarr"] = 8602,
4018 ["nleftarrow"] = 8602,
4019 ["nrarr"] = 8603,
4020 ["nrightarrow"] = 8603,
4021 ["nrarrw"] = {8605, 824},
4022 ["rarrw"] = 8605,
4023 ["rightsquigarrow"] = 8605,
4024 ["Larr"] = 8606,
4025 ["twoheadleftarrow"] = 8606,
4026 ["Uarr"] = 8607,
4027 ["Rarr"] = 8608,
4028 ["twoheadrightarrow"] = 8608,
4029 ["Darr"] = 8609,
4030 ["larrtl"] = 8610,
4031 ["leftarrowtail"] = 8610,
4032 ["rarrtl"] = 8611,
4033 ["rightarrowtail"] = 8611,
4034 ["LeftTeeArrow"] = 8612,
4035 ["mapstoleft"] = 8612,
4036 ["UpTeeArrow"] = 8613,
4037 ["mapstoup"] = 8613,
4038 ["RightTeeArrow"] = 8614,
4039 ["map"] = 8614,
4040 ["mapsto"] = 8614,
4041 ["DownTeeArrow"] = 8615,
4042 ["mapstodown"] = 8615,
4043 ["hookleftarrow"] = 8617,

```

```

4044 ["larrhk"] = 8617,
4045 ["hookrightarrow"] = 8618,
4046 ["rarrhk"] = 8618,
4047 ["larrlp"] = 8619,
4048 ["looparrowleft"] = 8619,
4049 ["looparrowright"] = 8620,
4050 ["rarrlp"] = 8620,
4051 ["harrw"] = 8621,
4052 ["leftrightsquigarrow"] = 8621,
4053 ["nharr"] = 8622,
4054 ["nletrightarrow"] = 8622,
4055 ["Lsh"] = 8624,
4056 ["lsh"] = 8624,
4057 ["Rsh"] = 8625,
4058 ["rsh"] = 8625,
4059 ["ldsh"] = 8626,
4060 ["rdsh"] = 8627,
4061 ["crarr"] = 8629,
4062 ["cularr"] = 8630,
4063 ["curvearrowleft"] = 8630,
4064 ["curarr"] = 8631,
4065 ["curvearrowright"] = 8631,
4066 ["circlearrowleft"] = 8634,
4067 ["olarr"] = 8634,
4068 ["circlearrowright"] = 8635,
4069 ["orarr"] = 8635,
4070 ["LeftVector"] = 8636,
4071 ["leftharpoonup"] = 8636,
4072 ["lharu"] = 8636,
4073 ["DownLeftVector"] = 8637,
4074 ["leftharpoondown"] = 8637,
4075 ["lhard"] = 8637,
4076 ["RightUpVector"] = 8638,
4077 ["uharr"] = 8638,
4078 ["upharpoonright"] = 8638,
4079 ["LeftUpVector"] = 8639,
4080 ["uharl"] = 8639,
4081 ["upharpoonleft"] = 8639,
4082 ["RightVector"] = 8640,
4083 ["rharu"] = 8640,
4084 ["rightharpoonup"] = 8640,
4085 ["DownRightVector"] = 8641,
4086 ["rhard"] = 8641,
4087 ["rightharpoondown"] = 8641,
4088 ["RightDownVector"] = 8642,
4089 ["dharr"] = 8642,
4090 ["downharpoonright"] = 8642,

```

```

4091 ["LeftDownVector"] = 8643,
4092 ["dharl"] = 8643,
4093 ["downharpoonleft"] = 8643,
4094 ["RightArrowLeftArrow"] = 8644,
4095 ["rightleftarrows"] = 8644,
4096 ["rlarr"] = 8644,
4097 ["UpArrowDownArrow"] = 8645,
4098 ["udarr"] = 8645,
4099 ["LeftArrowRightArrow"] = 8646,
4100 ["leftrightarrows"] = 8646,
4101 ["lrarr"] = 8646,
4102 ["leftleftarrows"] = 8647,
4103 ["llarr"] = 8647,
4104 ["upuparrows"] = 8648,
4105 ["uuarr"] = 8648,
4106 ["rightrightarrows"] = 8649,
4107 ["rrarr"] = 8649,
4108 ["ddarr"] = 8650,
4109 ["downdownarrows"] = 8650,
4110 ["ReverseEquilibrium"] = 8651,
4111 ["leftrightharpoons"] = 8651,
4112 ["lrhar"] = 8651,
4113 ["Equilibrium"] = 8652,
4114 ["rightleftharpoons"] = 8652,
4115 ["rlhar"] = 8652,
4116 ["nLeftarrow"] = 8653,
4117 ["nlArr"] = 8653,
4118 ["nLeftrightarrow"] = 8654,
4119 ["nhArr"] = 8654,
4120 ["nRightarrow"] = 8655,
4121 ["nrArr"] = 8655,
4122 ["DoubleLeftArrow"] = 8656,
4123 ["Leftarrow"] = 8656,
4124 ["lArr"] = 8656,
4125 ["DoubleUpArrow"] = 8657,
4126 ["Uparrow"] = 8657,
4127 ["uArr"] = 8657,
4128 ["DoubleRightArrow"] = 8658,
4129 ["Implies"] = 8658,
4130 ["Rightarrow"] = 8658,
4131 ["rArr"] = 8658,
4132 ["DoubleDownArrow"] = 8659,
4133 ["Downarrow"] = 8659,
4134 ["dArr"] = 8659,
4135 ["DoubleLeftRightArrow"] = 8660,
4136 ["Leftrightarrow"] = 8660,
4137 ["hArr"] = 8660,

```

```

4138 ["iff"] = 8660,
4139 ["DoubleUpDownArrow"] = 8661,
4140 ["Updownarrow"] = 8661,
4141 ["vArr"] = 8661,
4142 ["nwArr"] = 8662,
4143 ["neArr"] = 8663,
4144 ["seArr"] = 8664,
4145 ["swArr"] = 8665,
4146 ["Lleftarrow"] = 8666,
4147 ["lAarr"] = 8666,
4148 ["Rrightarrow"] = 8667,
4149 ["rAarr"] = 8667,
4150 ["zigrarr"] = 8669,
4151 ["LeftArrowBar"] = 8676,
4152 ["larrb"] = 8676,
4153 ["RightArrowBar"] = 8677,
4154 ["rarrb"] = 8677,
4155 ["DownArrowUpArrow"] = 8693,
4156 ["duarr"] = 8693,
4157 ["loarr"] = 8701,
4158 ["roarr"] = 8702,
4159 ["hoarr"] = 8703,
4160 ["ForAll"] = 8704,
4161 ["forall"] = 8704,
4162 ["comp"] = 8705,
4163 ["complement"] = 8705,
4164 ["PartialD"] = 8706,
4165 ["npart"] = {8706, 824},
4166 ["part"] = 8706,
4167 ["Exists"] = 8707,
4168 ["exist"] = 8707,
4169 ["NotExists"] = 8708,
4170 ["nexist"] = 8708,
4171 ["nexists"] = 8708,
4172 ["empty"] = 8709,
4173 ["emptyset"] = 8709,
4174 ["emptyv"] = 8709,
4175 ["varnothing"] = 8709,
4176 ["Del"] = 8711,
4177 ["nabla"] = 8711,
4178 ["Element"] = 8712,
4179 ["in"] = 8712,
4180 ["isin"] = 8712,
4181 ["isinv"] = 8712,
4182 ["NotElement"] = 8713,
4183 ["notin"] = 8713,
4184 ["notinva"] = 8713,

```

```

4185 ["ReverseElement"] = 8715,
4186 ["SuchThat"] = 8715,
4187 ["ni"] = 8715,
4188 ["niv"] = 8715,
4189 ["NotReverseElement"] = 8716,
4190 ["notni"] = 8716,
4191 ["notniva"] = 8716,
4192 ["Product"] = 8719,
4193 ["prod"] = 8719,
4194 ["Coproduct"] = 8720,
4195 ["coprod"] = 8720,
4196 ["Sum"] = 8721,
4197 ["sum"] = 8721,
4198 ["minus"] = 8722,
4199 ["MinusPlus"] = 8723,
4200 ["mnplus"] = 8723,
4201 ["mp"] = 8723,
4202 ["dotplus"] = 8724,
4203 ["plusdo"] = 8724,
4204 ["Backslash"] = 8726,
4205 ["setminus"] = 8726,
4206 ["setmn"] = 8726,
4207 ["smallsetminus"] = 8726,
4208 ["ssetmn"] = 8726,
4209 ["lowast"] = 8727,
4210 ["SmallCircle"] = 8728,
4211 ["compfn"] = 8728,
4212 ["Sqrt"] = 8730,
4213 ["radic"] = 8730,
4214 ["Proportional"] = 8733,
4215 ["prop"] = 8733,
4216 ["propto"] = 8733,
4217 ["varpropto"] = 8733,
4218 ["vprop"] = 8733,
4219 ["infin"] = 8734,
4220 ["angrt"] = 8735,
4221 ["ang"] = 8736,
4222 ["angle"] = 8736,
4223 ["nang"] = {8736, 8402},
4224 ["angmsd"] = 8737,
4225 ["measuredangle"] = 8737,
4226 ["angsph"] = 8738,
4227 ["VerticalBar"] = 8739,
4228 ["mid"] = 8739,
4229 ["shortmid"] = 8739,
4230 ["smid"] = 8739,
4231 ["NotVerticalBar"] = 8740,

```

```

4232 ["nmid"] = 8740,
4233 ["nshortmid"] = 8740,
4234 ["nsmid"] = 8740,
4235 ["DoubleVerticalBar"] = 8741,
4236 ["par"] = 8741,
4237 ["parallel"] = 8741,
4238 ["shortparallel"] = 8741,
4239 ["spar"] = 8741,
4240 ["NotDoubleVerticalBar"] = 8742,
4241 ["npar"] = 8742,
4242 ["nparallel"] = 8742,
4243 ["nshortparallel"] = 8742,
4244 ["nspar"] = 8742,
4245 ["and"] = 8743,
4246 ["wedge"] = 8743,
4247 ["or"] = 8744,
4248 ["vee"] = 8744,
4249 ["cap"] = 8745,
4250 ["caps"] = {8745, 65024},
4251 ["cup"] = 8746,
4252 ["cups"] = {8746, 65024},
4253 ["Integral"] = 8747,
4254 ["int"] = 8747,
4255 ["Int"] = 8748,
4256 ["iiint"] = 8749,
4257 ["tint"] = 8749,
4258 ["ContourIntegral"] = 8750,
4259 ["conint"] = 8750,
4260 ["oint"] = 8750,
4261 ["Conint"] = 8751,
4262 ["DoubleContourIntegral"] = 8751,
4263 ["Cconint"] = 8752,
4264 ["cwint"] = 8753,
4265 ["ClockwiseContourIntegral"] = 8754,
4266 ["cwconint"] = 8754,
4267 ["CounterClockwiseContourIntegral"] = 8755,
4268 ["awconint"] = 8755,
4269 ["Therefore"] = 8756,
4270 ["there4"] = 8756,
4271 ["therefore"] = 8756,
4272 ["Because"] = 8757,
4273 ["because"] = 8757,
4274 ["because"] = 8757,
4275 ["ratio"] = 8758,
4276 ["Colon"] = 8759,
4277 ["Proportion"] = 8759,
4278 ["dotminus"] = 8760,

```

```

4279 ["minusd"] = 8760,
4280 ["mDDot"] = 8762,
4281 ["homtht"] = 8763,
4282 ["Tilde"] = 8764,
4283 ["nvsim"] = {8764, 8402},
4284 ["sim"] = 8764,
4285 ["thicksim"] = 8764,
4286 ["thksim"] = 8764,
4287 ["backsim"] = 8765,
4288 ["bsim"] = 8765,
4289 ["race"] = {8765, 817},
4290 ["ac"] = 8766,
4291 ["acE"] = {8766, 819},
4292 ["mstpos"] = 8766,
4293 ["acd"] = 8767,
4294 ["VerticalTilde"] = 8768,
4295 ["wr"] = 8768,
4296 ["wreath"] = 8768,
4297 ["NotTilde"] = 8769,
4298 ["nsim"] = 8769,
4299 ["EqualTilde"] = 8770,
4300 ["NotEqualTilde"] = {8770, 824},
4301 ["eqsim"] = 8770,
4302 ["esim"] = 8770,
4303 ["nesim"] = {8770, 824},
4304 ["TildeEqual"] = 8771,
4305 ["sime"] = 8771,
4306 ["simeq"] = 8771,
4307 ["NotTildeEqual"] = 8772,
4308 ["nsime"] = 8772,
4309 ["nsimeq"] = 8772,
4310 ["TildeFullEqual"] = 8773,
4311 ["cong"] = 8773,
4312 ["simne"] = 8774,
4313 ["NotTildeFullEqual"] = 8775,
4314 ["ncong"] = 8775,
4315 ["TildeTilde"] = 8776,
4316 ["ap"] = 8776,
4317 ["approx"] = 8776,
4318 ["asymp"] = 8776,
4319 ["thickapprox"] = 8776,
4320 ["thkap"] = 8776,
4321 ["NotTildeTilde"] = 8777,
4322 ["nap"] = 8777,
4323 ["napprox"] = 8777,
4324 ["ape"] = 8778,
4325 ["approxeq"] = 8778,

```



```

4326 ["apid"] = 8779,
4327 ["napid"] = {8779, 824},
4328 ["backcong"] = 8780,
4329 ["bcong"] = 8780,
4330 ["CupCap"] = 8781,
4331 ["asympeq"] = 8781,
4332 ["nvap"] = {8781, 8402},
4333 ["Bumpeq"] = 8782,
4334 ["HumpDownHump"] = 8782,
4335 ["NotHumpDownHump"] = {8782, 824},
4336 ["bump"] = 8782,
4337 ["nbump"] = {8782, 824},
4338 ["HumpEqual"] = 8783,
4339 ["NotHumpEqual"] = {8783, 824},
4340 ["bumpe"] = 8783,
4341 ["bumpeq"] = 8783,
4342 ["nbumpe"] = {8783, 824},
4343 ["DotEqual"] = 8784,
4344 ["doteq"] = 8784,
4345 ["esdot"] = 8784,
4346 ["nedot"] = {8784, 824},
4347 ["doteqdot"] = 8785,
4348 ["eDot"] = 8785,
4349 ["efDot"] = 8786,
4350 ["fallingdotseq"] = 8786,
4351 ["erDot"] = 8787,
4352 ["risingdotseq"] = 8787,
4353 ["Assign"] = 8788,
4354 ["colone"] = 8788,
4355 ["coloneq"] = 8788,
4356 ["ecolon"] = 8789,
4357 ["eqcolon"] = 8789,
4358 ["ecir"] = 8790,
4359 ["eqcirc"] = 8790,
4360 ["circeq"] = 8791,
4361 ["cire"] = 8791,
4362 ["wedgeq"] = 8793,
4363 ["veeeq"] = 8794,
4364 ["triangleq"] = 8796,
4365 ["trie"] = 8796,
4366 ["equest"] = 8799,
4367 ["questeq"] = 8799,
4368 ["NotEqual"] = 8800,
4369 ["ne"] = 8800,
4370 ["Congruent"] = 8801,
4371 ["bnequiv"] = {8801, 8421},
4372 ["equiv"] = 8801,

```

```

4373 ["NotCongruent"] = 8802,
4374 ["nequiv"] = 8802,
4375 ["le"] = 8804,
4376 ["leq"] = 8804,
4377 ["nvle"] = {8804, 8402},
4378 ["GreaterEqual"] = 8805,
4379 ["ge"] = 8805,
4380 ["geq"] = 8805,
4381 ["nvge"] = {8805, 8402},
4382 ["LessFullEqual"] = 8806,
4383 ["lE"] = 8806,
4384 ["leqq"] = 8806,
4385 ["nlE"] = {8806, 824},
4386 ["nleqq"] = {8806, 824},
4387 ["GreaterFullEqual"] = 8807,
4388 ["NotGreaterFullEqual"] = {8807, 824},
4389 ["gE"] = 8807,
4390 ["geqq"] = 8807,
4391 ["ngE"] = {8807, 824},
4392 ["ngeqq"] = {8807, 824},
4393 ["lnE"] = 8808,
4394 ["lneqq"] = 8808,
4395 ["lvertneqq"] = {8808, 65024},
4396 ["lvnE"] = {8808, 65024},
4397 ["gnE"] = 8809,
4398 ["gneqq"] = 8809,
4399 ["gvertneqq"] = {8809, 65024},
4400 ["gvnE"] = {8809, 65024},
4401 ["Lt"] = 8810,
4402 ["NestedLessLess"] = 8810,
4403 ["NotLessLess"] = {8810, 824},
4404 ["ll"] = 8810,
4405 ["nLt"] = {8810, 8402},
4406 ["nLtv"] = {8810, 824},
4407 ["Gt"] = 8811,
4408 ["NestedGreaterGreater"] = 8811,
4409 ["NotGreaterGreater"] = {8811, 824},
4410 ["gg"] = 8811,
4411 ["nGt"] = {8811, 8402},
4412 ["nGtv"] = {8811, 824},
4413 ["between"] = 8812,
4414 ["twixt"] = 8812,
4415 ["NotCupCap"] = 8813,
4416 ["NotLess"] = 8814,
4417 ["nless"] = 8814,
4418 ["nlt"] = 8814,
4419 ["NotGreater"] = 8815,

```

```

4420 ["ngt"] = 8815,
4421 ["ngtr"] = 8815,
4422 ["NotLessEqual"] = 8816,
4423 ["nle"] = 8816,
4424 ["nleq"] = 8816,
4425 ["NotGreaterEqual"] = 8817,
4426 ["nge"] = 8817,
4427 ["ngeq"] = 8817,
4428 ["LessTilde"] = 8818,
4429 ["lesssim"] = 8818,
4430 ["lsim"] = 8818,
4431 ["GreaterTilde"] = 8819,
4432 ["gsim"] = 8819,
4433 ["gtrsim"] = 8819,
4434 ["NotLessTilde"] = 8820,
4435 ["nlsim"] = 8820,
4436 ["NotGreaterTilde"] = 8821,
4437 ["ngsim"] = 8821,
4438 ["LessGreater"] = 8822,
4439 ["lessgtr"] = 8822,
4440 ["lg"] = 8822,
4441 ["GreaterLess"] = 8823,
4442 ["gl"] = 8823,
4443 ["gtrless"] = 8823,
4444 ["NotLessGreater"] = 8824,
4445 ["ntlg"] = 8824,
4446 ["NotGreaterLess"] = 8825,
4447 ["ntgl"] = 8825,
4448 ["Precedes"] = 8826,
4449 ["pr"] = 8826,
4450 ["prec"] = 8826,
4451 ["Succeeds"] = 8827,
4452 ["sc"] = 8827,
4453 ["succ"] = 8827,
4454 ["PrecedesSlantEqual"] = 8828,
4455 ["prcue"] = 8828,
4456 ["preccurlyeq"] = 8828,
4457 ["SucceedsSlantEqual"] = 8829,
4458 ["sccue"] = 8829,
4459 ["succcurlyeq"] = 8829,
4460 ["PrecedesTilde"] = 8830,
4461 ["precsim"] = 8830,
4462 ["prsim"] = 8830,
4463 ["NotSucceedsTilde"] = {8831, 824},
4464 ["SucceedsTilde"] = 8831,
4465 ["scsim"] = 8831,
4466 ["succsim"] = 8831,

```

```

4467 ["NotPrecedes"] = 8832,
4468 ["npr"] = 8832,
4469 ["nprec"] = 8832,
4470 ["NotSucceeds"] = 8833,
4471 ["nsc"] = 8833,
4472 ["nsucc"] = 8833,
4473 ["NotSubset"] = {8834, 8402},
4474 ["nsubset"] = {8834, 8402},
4475 ["sub"] = 8834,
4476 ["subset"] = 8834,
4477 ["vnsup"] = {8834, 8402},
4478 ["NotSuperset"] = {8835, 8402},
4479 ["Superset"] = 8835,
4480 ["nsupset"] = {8835, 8402},
4481 ["sup"] = 8835,
4482 ["supset"] = 8835,
4483 ["vnsup"] = {8835, 8402},
4484 ["nsub"] = 8836,
4485 ["nsup"] = 8837,
4486 ["SubsetEqual"] = 8838,
4487 ["sube"] = 8838,
4488 ["subseteq"] = 8838,
4489 ["SupersetEqual"] = 8839,
4490 ["supe"] = 8839,
4491 ["supseteq"] = 8839,
4492 ["NotSubsetEqual"] = 8840,
4493 ["nsube"] = 8840,
4494 ["nsubseteq"] = 8840,
4495 ["NotSupersetEqual"] = 8841,
4496 ["nsupe"] = 8841,
4497 ["nsupseteq"] = 8841,
4498 ["subne"] = 8842,
4499 ["subsetneq"] = 8842,
4500 ["varsubsetneq"] = {8842, 65024},
4501 ["vsubne"] = {8842, 65024},
4502 ["supne"] = 8843,
4503 ["supsetneq"] = 8843,
4504 ["varsupsetneq"] = {8843, 65024},
4505 ["vsupne"] = {8843, 65024},
4506 ["cupdot"] = 8845,
4507 ["UnionPlus"] = 8846,
4508 ["uplus"] = 8846,
4509 ["NotSquareSubset"] = {8847, 824},
4510 ["SquareSubset"] = 8847,
4511 ["sqsub"] = 8847,
4512 ["sqsubset"] = 8847,
4513 ["NotSquareSuperset"] = {8848, 824},

```

```

4514 ["SquareSuperset"] = 8848,
4515 ["sqsup"] = 8848,
4516 ["sqsupset"] = 8848,
4517 ["SquareSubsetEqual"] = 8849,
4518 ["sqsube"] = 8849,
4519 ["sqsubseteq"] = 8849,
4520 ["SquareSupersetEqual"] = 8850,
4521 ["sqsupe"] = 8850,
4522 ["sqsupseteq"] = 8850,
4523 ["SquareIntersection"] = 8851,
4524 ["sqcap"] = 8851,
4525 ["sqcaps"] = {8851, 65024},
4526 ["SquareUnion"] = 8852,
4527 ["sqcup"] = 8852,
4528 ["sqcups"] = {8852, 65024},
4529 ["CirclePlus"] = 8853,
4530 ["oplus"] = 8853,
4531 ["CircleMinus"] = 8854,
4532 ["ominus"] = 8854,
4533 ["CircleTimes"] = 8855,
4534 ["otimes"] = 8855,
4535 ["osol"] = 8856,
4536 ["CircleDot"] = 8857,
4537 ["odot"] = 8857,
4538 ["circledcirc"] = 8858,
4539 ["ocir"] = 8858,
4540 ["circledast"] = 8859,
4541 ["oast"] = 8859,
4542 ["circleddash"] = 8861,
4543 ["odash"] = 8861,
4544 ["boxplus"] = 8862,
4545 ["plusb"] = 8862,
4546 ["boxminus"] = 8863,
4547 ["minusb"] = 8863,
4548 ["boxtimes"] = 8864,
4549 ["timesb"] = 8864,
4550 ["dotsquare"] = 8865,
4551 ["sdotb"] = 8865,
4552 ["RightTee"] = 8866,
4553 ["vdash"] = 8866,
4554 ["LeftTee"] = 8867,
4555 ["dashv"] = 8867,
4556 ["DownTee"] = 8868,
4557 ["top"] = 8868,
4558 ["UpTee"] = 8869,
4559 ["bot"] = 8869,
4560 ["bottom"] = 8869,

```

```

4561 ["perp"] = 8869,
4562 ["models"] = 8871,
4563 ["DoubleRightTee"] = 8872,
4564 ["vDash"] = 8872,
4565 ["Vdash"] = 8873,
4566 ["Vvdash"] = 8874,
4567 ["VDash"] = 8875,
4568 ["nvdash"] = 8876,
4569 ["nvDash"] = 8877,
4570 ["nVdash"] = 8878,
4571 ["nVDash"] = 8879,
4572 ["prurel"] = 8880,
4573 ["LeftTriangle"] = 8882,
4574 ["vartriangleleft"] = 8882,
4575 ["vltri"] = 8882,
4576 ["RightTriangle"] = 8883,
4577 ["vartriangleright"] = 8883,
4578 ["vrtri"] = 8883,
4579 ["LeftTriangleEqual"] = 8884,
4580 ["ltrie"] = 8884,
4581 ["nvltrie"] = {8884, 8402},
4582 ["trianglelefteq"] = 8884,
4583 ["RightTriangleEqual"] = 8885,
4584 ["nvrtrie"] = {8885, 8402},
4585 ["rtrie"] = 8885,
4586 ["trianglerighteq"] = 8885,
4587 ["origof"] = 8886,
4588 ["imof"] = 8887,
4589 ["multimap"] = 8888,
4590 ["mumap"] = 8888,
4591 ["hercon"] = 8889,
4592 ["intcal"] = 8890,
4593 ["intercal"] = 8890,
4594 ["veebar"] = 8891,
4595 ["barvee"] = 8893,
4596 ["angrtvb"] = 8894,
4597 ["lrltri"] = 8895,
4598 ["Wedge"] = 8896,
4599 ["bigwedge"] = 8896,
4600 ["xwedge"] = 8896,
4601 ["Vee"] = 8897,
4602 ["bigvee"] = 8897,
4603 ["xvee"] = 8897,
4604 ["Intersection"] = 8898,
4605 ["bigcap"] = 8898,
4606 ["xcap"] = 8898,
4607 ["Union"] = 8899,

```

```

4608 ["bigcup"] = 8899,
4609 ["xcup"] = 8899,
4610 ["Diamond"] = 8900,
4611 ["diam"] = 8900,
4612 ["diamond"] = 8900,
4613 ["sdot"] = 8901,
4614 ["Star"] = 8902,
4615 ["sstarf"] = 8902,
4616 ["divideontimes"] = 8903,
4617 ["divonx"] = 8903,
4618 ["bowtie"] = 8904,
4619 ["ltimes"] = 8905,
4620 ["rtimes"] = 8906,
4621 ["leftthreetimes"] = 8907,
4622 ["lthree"] = 8907,
4623 ["rightthreetimes"] = 8908,
4624 ["rthree"] = 8908,
4625 ["backsimeq"] = 8909,
4626 ["bsime"] = 8909,
4627 ["curlyvee"] = 8910,
4628 ["cuvee"] = 8910,
4629 ["curlywedge"] = 8911,
4630 ["cuwed"] = 8911,
4631 ["Sub"] = 8912,
4632 ["Subset"] = 8912,
4633 ["Sup"] = 8913,
4634 ["Supset"] = 8913,
4635 ["Cap"] = 8914,
4636 ["Cup"] = 8915,
4637 ["fork"] = 8916,
4638 ["pitchfork"] = 8916,
4639 ["epar"] = 8917,
4640 ["lessdot"] = 8918,
4641 ["ltdot"] = 8918,
4642 ["gtdot"] = 8919,
4643 ["gtrdot"] = 8919,
4644 ["Ll"] = 8920,
4645 ["nLl"] = {8920, 824},
4646 ["Gg"] = 8921,
4647 ["ggg"] = 8921,
4648 ["nGg"] = {8921, 824},
4649 ["LessEqualGreater"] = 8922,
4650 ["leg"] = 8922,
4651 ["lesg"] = {8922, 65024},
4652 ["lesseqgtr"] = 8922,
4653 ["GreaterEqualLess"] = 8923,
4654 ["gel"] = 8923,

```

```

4655 ["gesl"] = {8923, 65024},
4656 ["gtreqless"] = 8923,
4657 ["cuepr"] = 8926,
4658 ["curlyeqprec"] = 8926,
4659 ["cuesc"] = 8927,
4660 ["curlyeqsucc"] = 8927,
4661 ["NotPrecedesSlantEqual"] = 8928,
4662 ["nprcue"] = 8928,
4663 ["NotSucceedsSlantEqual"] = 8929,
4664 ["nsccue"] = 8929,
4665 ["NotSquareSubsetEqual"] = 8930,
4666 ["nsqsube"] = 8930,
4667 ["NotSquareSupersetEqual"] = 8931,
4668 ["nsqsupe"] = 8931,
4669 ["lnsim"] = 8934,
4670 ["gnsim"] = 8935,
4671 ["precnsim"] = 8936,
4672 ["prnsim"] = 8936,
4673 ["scnsim"] = 8937,
4674 ["succnsim"] = 8937,
4675 ["NotLeftTriangle"] = 8938,
4676 ["nltri"] = 8938,
4677 ["ntriangleleft"] = 8938,
4678 ["NotRightTriangle"] = 8939,
4679 ["nrtri"] = 8939,
4680 ["ntriangleright"] = 8939,
4681 ["NotLeftTriangleEqual"] = 8940,
4682 ["nltrie"] = 8940,
4683 ["ntrianglelefteq"] = 8940,
4684 ["NotRightTriangleEqual"] = 8941,
4685 ["nrtrie"] = 8941,
4686 ["ntrianglerighteq"] = 8941,
4687 ["vellip"] = 8942,
4688 ["ctdot"] = 8943,
4689 ["utdot"] = 8944,
4690 ["dtdot"] = 8945,
4691 ["disin"] = 8946,
4692 ["isinsv"] = 8947,
4693 ["isins"] = 8948,
4694 ["isindot"] = 8949,
4695 ["notindot"] = {8949, 824},
4696 ["notinvc"] = 8950,
4697 ["notinvb"] = 8951,
4698 ["isinE"] = 8953,
4699 ["notinE"] = {8953, 824},
4700 ["nisd"] = 8954,
4701 ["xnisd"] = 8955,

```



```

4702 ["nis"] = 8956,
4703 ["notnivc"] = 8957,
4704 ["notnivb"] = 8958,
4705 ["barwed"] = 8965,
4706 ["barwedge"] = 8965,
4707 ["Barwed"] = 8966,
4708 ["doublebarwedge"] = 8966,
4709 ["LeftCeiling"] = 8968,
4710 ["lceil"] = 8968,
4711 ["RightCeiling"] = 8969,
4712 ["rceil"] = 8969,
4713 ["LeftFloor"] = 8970,
4714 ["lfloor"] = 8970,
4715 ["RightFloor"] = 8971,
4716 ["rfloor"] = 8971,
4717 ["drcrop"] = 8972,
4718 ["dlcrop"] = 8973,
4719 ["urcrop"] = 8974,
4720 ["ulcrop"] = 8975,
4721 ["bnot"] = 8976,
4722 ["proflines"] = 8978,
4723 ["profsurf"] = 8979,
4724 ["telrec"] = 8981,
4725 ["target"] = 8982,
4726 ["ulcorn"] = 8988,
4727 ["ulcorner"] = 8988,
4728 ["urcorn"] = 8989,
4729 ["urcorner"] = 8989,
4730 ["dlcorn"] = 8990,
4731 ["llcorner"] = 8990,
4732 ["drcorn"] = 8991,
4733 ["lrcorn"] = 8991,
4734 ["frown"] = 8994,
4735 ["sfrown"] = 8994,
4736 ["smile"] = 8995,
4737 ["ssmile"] = 8995,
4738 ["cylcty"] = 9005,
4739 ["profalar"] = 9006,
4740 ["topbot"] = 9014,
4741 ["ovbar"] = 9021,
4742 ["solbar"] = 9023,
4743 ["angzarr"] = 9084,
4744 ["lmoust"] = 9136,
4745 ["lmoustache"] = 9136,
4746 ["rmoust"] = 9137,
4747 ["rmoustache"] = 9137,
4748 ["OverBracket"] = 9140,

```

```

4749 ["tbrk"] = 9140,
4750 ["UnderBracket"] = 9141,
4751 ["bbrk"] = 9141,
4752 ["bbrktbrk"] = 9142,
4753 ["OverParenthesis"] = 9180,
4754 ["UnderParenthesis"] = 9181,
4755 ["OverBrace"] = 9182,
4756 ["UnderBrace"] = 9183,
4757 ["trpezium"] = 9186,
4758 ["elinters"] = 9191,
4759 ["blank"] = 9251,
4760 ["circledS"] = 9416,
4761 ["oS"] = 9416,
4762 ["HorizontalLine"] = 9472,
4763 ["boxh"] = 9472,
4764 ["boxv"] = 9474,
4765 ["boxdr"] = 9484,
4766 ["boxdl"] = 9488,
4767 ["boxur"] = 9492,
4768 ["boxul"] = 9496,
4769 ["boxvr"] = 9500,
4770 ["boxvl"] = 9508,
4771 ["boxhd"] = 9516,
4772 ["boxhu"] = 9524,
4773 ["boxvh"] = 9532,
4774 ["boxH"] = 9552,
4775 ["boxV"] = 9553,
4776 ["boxdR"] = 9554,
4777 ["boxDr"] = 9555,
4778 ["boxDR"] = 9556,
4779 ["boxdL"] = 9557,
4780 ["boxDl"] = 9558,
4781 ["boxDL"] = 9559,
4782 ["boxuR"] = 9560,
4783 ["boxUr"] = 9561,
4784 ["boxUR"] = 9562,
4785 ["boxuL"] = 9563,
4786 ["boxUl"] = 9564,
4787 ["boxUL"] = 9565,
4788 ["boxvR"] = 9566,
4789 ["boxVr"] = 9567,
4790 ["boxVR"] = 9568,
4791 ["boxvL"] = 9569,
4792 ["boxVl"] = 9570,
4793 ["boxVL"] = 9571,
4794 ["boxHd"] = 9572,
4795 ["boxhD"] = 9573,

```

```

4796 ["boxHD"] = 9574,
4797 ["boxHu"] = 9575,
4798 ["boxhU"] = 9576,
4799 ["boxHU"] = 9577,
4800 ["boxvH"] = 9578,
4801 ["boxVh"] = 9579,
4802 ["boxVH"] = 9580,
4803 ["uhblk"] = 9600,
4804 ["lhblk"] = 9604,
4805 ["block"] = 9608,
4806 ["blk14"] = 9617,
4807 ["blk12"] = 9618,
4808 ["blk34"] = 9619,
4809 ["Square"] = 9633,
4810 ["squ"] = 9633,
4811 ["square"] = 9633,
4812 ["FilledVerySmallSquare"] = 9642,
4813 ["blacksquare"] = 9642,
4814 ["squarf"] = 9642,
4815 ["squf"] = 9642,
4816 ["EmptyVerySmallSquare"] = 9643,
4817 ["rect"] = 9645,
4818 ["marker"] = 9646,
4819 ["fltns"] = 9649,
4820 ["bigtriangleup"] = 9651,
4821 ["xutri"] = 9651,
4822 ["blacktriangle"] = 9652,
4823 ["utrif"] = 9652,
4824 ["triangle"] = 9653,
4825 ["utri"] = 9653,
4826 ["blacktriangleright"] = 9656,
4827 ["rtrif"] = 9656,
4828 ["rtri"] = 9657,
4829 ["triangleright"] = 9657,
4830 ["bigtriangledown"] = 9661,
4831 ["xdtri"] = 9661,
4832 ["blacktriangledown"] = 9662,
4833 ["dtrif"] = 9662,
4834 ["dtri"] = 9663,
4835 ["triangledown"] = 9663,
4836 ["blacktriangleleft"] = 9666,
4837 ["ltrif"] = 9666,
4838 ["ltri"] = 9667,
4839 ["triangleleft"] = 9667,
4840 ["loz"] = 9674,
4841 ["lozenge"] = 9674,
4842 ["cir"] = 9675,

```

```

4843 ["tridot"] = 9708,
4844 ["bigcirc"] = 9711,
4845 ["xcirc"] = 9711,
4846 ["ultri"] = 9720,
4847 ["urtri"] = 9721,
4848 ["lltri"] = 9722,
4849 ["EmptySmallSquare"] = 9723,
4850 ["FilledSmallSquare"] = 9724,
4851 ["bigstar"] = 9733,
4852 ["starf"] = 9733,
4853 ["star"] = 9734,
4854 ["phone"] = 9742,
4855 ["female"] = 9792,
4856 ["male"] = 9794,
4857 ["spades"] = 9824,
4858 ["spadesuit"] = 9824,
4859 ["clubs"] = 9827,
4860 ["clubsuit"] = 9827,
4861 ["hearts"] = 9829,
4862 ["heartsuit"] = 9829,
4863 ["diamondsuit"] = 9830,
4864 ["diams"] = 9830,
4865 ["sung"] = 9834,
4866 ["flat"] = 9837,
4867 ["natur"] = 9838,
4868 ["natural"] = 9838,
4869 ["sharp"] = 9839,
4870 ["check"] = 10003,
4871 ["checkmark"] = 10003,
4872 ["cross"] = 10007,
4873 ["malt"] = 10016,
4874 ["maltese"] = 10016,
4875 ["sext"] = 10038,
4876 ["VerticalSeparator"] = 10072,
4877 ["lbbbrk"] = 10098,
4878 ["rbbrk"] = 10099,
4879 ["bsolhsub"] = 10184,
4880 ["suphsol"] = 10185,
4881 ["LeftDoubleBracket"] = 10214,
4882 ["lobrk"] = 10214,
4883 ["RightDoubleBracket"] = 10215,
4884 ["robrk"] = 10215,
4885 ["LeftAngleBracket"] = 10216,
4886 ["lang"] = 10216,
4887 ["langle"] = 10216,
4888 ["RightAngleBracket"] = 10217,
4889 ["rang"] = 10217,

```

```

4890 ["rangle"] = 10217,
4891 ["Lang"] = 10218,
4892 ["Rang"] = 10219,
4893 ["loang"] = 10220,
4894 ["roang"] = 10221,
4895 ["LongLeftArrow"] = 10229,
4896 ["longleftarrow"] = 10229,
4897 ["xlarr"] = 10229,
4898 ["LongRightArrow"] = 10230,
4899 ["longrightarrow"] = 10230,
4900 ["xrarr"] = 10230,
4901 ["LongLeftRightArrow"] = 10231,
4902 ["longleftrightarrow"] = 10231,
4903 ["xharr"] = 10231,
4904 ["DoubleLongLeftArrow"] = 10232,
4905 ["Longleftarrow"] = 10232,
4906 ["xlArr"] = 10232,
4907 ["DoubleLongRightArrow"] = 10233,
4908 ["Longrightarrow"] = 10233,
4909 ["xrArr"] = 10233,
4910 ["DoubleLongLeftRightArrow"] = 10234,
4911 ["Longleftrightarrow"] = 10234,
4912 ["xhArr"] = 10234,
4913 ["longmapsto"] = 10236,
4914 ["xmap"] = 10236,
4915 ["dzigrarr"] = 10239,
4916 ["nvlArr"] = 10498,
4917 ["nvrArr"] = 10499,
4918 ["nvHarr"] = 10500,
4919 ["Map"] = 10501,
4920 ["lbarr"] = 10508,
4921 ["bkarow"] = 10509,
4922 ["rbarr"] = 10509,
4923 ["lBarr"] = 10510,
4924 ["dbkarow"] = 10511,
4925 ["rBarr"] = 10511,
4926 ["RBarr"] = 10512,
4927 ["drbkarow"] = 10512,
4928 ["DDottrahd"] = 10513,
4929 ["UpArrowBar"] = 10514,
4930 ["DownArrowBar"] = 10515,
4931 ["Rarrtl"] = 10518,
4932 ["latail"] = 10521,
4933 ["ratail"] = 10522,
4934 ["lAtail"] = 10523,
4935 ["rAtail"] = 10524,
4936 ["larrfs"] = 10525,

```

```

4937 ["rarrfs"] = 10526,
4938 ["larrbfs"] = 10527,
4939 ["rarrbfs"] = 10528,
4940 ["nwarhk"] = 10531,
4941 ["nearhk"] = 10532,
4942 ["hksearow"] = 10533,
4943 ["searhk"] = 10533,
4944 ["hkswarow"] = 10534,
4945 ["swarhk"] = 10534,
4946 ["nwnear"] = 10535,
4947 ["nesear"] = 10536,
4948 ["toea"] = 10536,
4949 ["seswar"] = 10537,
4950 ["tosa"] = 10537,
4951 ["swnwar"] = 10538,
4952 ["nrarrc"] = {10547, 824},
4953 ["rarrc"] = 10547,
4954 ["cudarrr"] = 10549,
4955 ["ldca"] = 10550,
4956 ["rdca"] = 10551,
4957 ["cudarrrl"] = 10552,
4958 ["larrpl"] = 10553,
4959 ["curarrm"] = 10556,
4960 ["cularrp"] = 10557,
4961 ["rarrpl"] = 10565,
4962 ["harrcir"] = 10568,
4963 ["Uarrocir"] = 10569,
4964 ["lurdshar"] = 10570,
4965 ["ldrushar"] = 10571,
4966 ["LeftRightVector"] = 10574,
4967 ["RightUpDownVector"] = 10575,
4968 ["DownLeftRightVector"] = 10576,
4969 ["LeftUpDownVector"] = 10577,
4970 ["LeftVectorBar"] = 10578,
4971 ["RightVectorBar"] = 10579,
4972 ["RightUpVectorBar"] = 10580,
4973 ["RightDownVectorBar"] = 10581,
4974 ["DownLeftVectorBar"] = 10582,
4975 ["DownRightVectorBar"] = 10583,
4976 ["LeftUpVectorBar"] = 10584,
4977 ["LeftDownVectorBar"] = 10585,
4978 ["LeftTeeVector"] = 10586,
4979 ["RightTeeVector"] = 10587,
4980 ["RightUpTeeVector"] = 10588,
4981 ["RightDownTeeVector"] = 10589,
4982 ["DownLeftTeeVector"] = 10590,
4983 ["DownRightTeeVector"] = 10591,

```

```

4984 ["LeftUpTeeVector"] = 10592,
4985 ["LeftDownTeeVector"] = 10593,
4986 ["lHar"] = 10594,
4987 ["uHar"] = 10595,
4988 ["rHar"] = 10596,
4989 ["dHar"] = 10597,
4990 ["luruhar"] = 10598,
4991 ["ldrdhar"] = 10599,
4992 ["ruluhar"] = 10600,
4993 ["rdldhar"] = 10601,
4994 ["lharul"] = 10602,
4995 ["llhard"] = 10603,
4996 ["rharul"] = 10604,
4997 ["lrhard"] = 10605,
4998 ["UpEquilibrium"] = 10606,
4999 ["udhar"] = 10606,
5000 ["ReverseUpEquilibrium"] = 10607,
5001 ["duhar"] = 10607,
5002 ["RoundImplies"] = 10608,
5003 ["erarr"] = 10609,
5004 ["simrarr"] = 10610,
5005 ["larrsim"] = 10611,
5006 ["rarrsim"] = 10612,
5007 ["rarrap"] = 10613,
5008 ["ltlarr"] = 10614,
5009 ["gtrarr"] = 10616,
5010 ["subrarr"] = 10617,
5011 ["suplarr"] = 10619,
5012 ["lfisht"] = 10620,
5013 ["rfisht"] = 10621,
5014 ["ufisht"] = 10622,
5015 ["dfisht"] = 10623,
5016 ["lopar"] = 10629,
5017 ["ropar"] = 10630,
5018 ["lbrke"] = 10635,
5019 ["rbrke"] = 10636,
5020 ["lbrkslu"] = 10637,
5021 ["rbrksld"] = 10638,
5022 ["lbrksld"] = 10639,
5023 ["rbrkslu"] = 10640,
5024 ["langd"] = 10641,
5025 ["rangd"] = 10642,
5026 ["lparlt"] = 10643,
5027 ["rpargt"] = 10644,
5028 ["gtlPar"] = 10645,
5029 ["ltrPar"] = 10646,
5030 ["vzigzag"] = 10650,

```

```

5031 ["vangrt"] = 10652,
5032 ["angrtvbd"] = 10653,
5033 ["ange"] = 10660,
5034 ["range"] = 10661,
5035 ["dwangle"] = 10662,
5036 ["uwangle"] = 10663,
5037 ["angmsdaa"] = 10664,
5038 ["angmsdab"] = 10665,
5039 ["angmsdac"] = 10666,
5040 ["angmsdad"] = 10667,
5041 ["angmsdae"] = 10668,
5042 ["angmsdaf"] = 10669,
5043 ["angmsdag"] = 10670,
5044 ["angmsdah"] = 10671,
5045 ["bemptyv"] = 10672,
5046 ["demptyv"] = 10673,
5047 ["cemptyv"] = 10674,
5048 ["raemptyv"] = 10675,
5049 ["laemptyv"] = 10676,
5050 ["ohbar"] = 10677,
5051 ["omid"] = 10678,
5052 ["opar"] = 10679,
5053 ["operp"] = 10681,
5054 ["olcross"] = 10683,
5055 ["odsold"] = 10684,
5056 ["olcir"] = 10686,
5057 ["ofcir"] = 10687,
5058 ["olt"] = 10688,
5059 ["ogt"] = 10689,
5060 ["cirscir"] = 10690,
5061 ["cirE"] = 10691,
5062 ["solb"] = 10692,
5063 ["bsolb"] = 10693,
5064 ["boxbox"] = 10697,
5065 ["trish"] = 10701,
5066 ["rtriltri"] = 10702,
5067 ["LeftTriangleBar"] = 10703,
5068 ["NotLeftTriangleBar"] = {10703, 824},
5069 ["NotRightTriangleBar"] = {10704, 824},
5070 ["RightTriangleBar"] = 10704,
5071 ["iinfin"] = 10716,
5072 ["infintie"] = 10717,
5073 ["nvinfin"] = 10718,
5074 ["eparsl"] = 10723,
5075 ["smeparsl"] = 10724,
5076 ["eqvparsl"] = 10725,
5077 ["blacklozenge"] = 10731,

```



```

5078 ["lozf"] = 10731,
5079 ["RuleDelayed"] = 10740,
5080 ["dsol"] = 10742,
5081 ["bigodot"] = 10752,
5082 ["xodot"] = 10752,
5083 ["bigoplus"] = 10753,
5084 ["xoplus"] = 10753,
5085 ["bigotimes"] = 10754,
5086 ["xotime"] = 10754,
5087 ["biguplus"] = 10756,
5088 ["xuplus"] = 10756,
5089 ["bigsqcup"] = 10758,
5090 ["xsqcup"] = 10758,
5091 ["iiiint"] = 10764,
5092 ["qint"] = 10764,
5093 ["fpartint"] = 10765,
5094 ["cirfnint"] = 10768,
5095 ["awint"] = 10769,
5096 ["rppolint"] = 10770,
5097 ["scpolint"] = 10771,
5098 ["npolint"] = 10772,
5099 ["pointint"] = 10773,
5100 ["quatint"] = 10774,
5101 ["intlarhk"] = 10775,
5102 ["pluscir"] = 10786,
5103 ["plusacir"] = 10787,
5104 ["simplus"] = 10788,
5105 ["plusdu"] = 10789,
5106 ["plussim"] = 10790,
5107 ["plustwo"] = 10791,
5108 ["mcomma"] = 10793,
5109 ["minusdu"] = 10794,
5110 ["loplus"] = 10797,
5111 ["roplus"] = 10798,
5112 ["Cross"] = 10799,
5113 ["timesd"] = 10800,
5114 ["timesbar"] = 10801,
5115 ["smashp"] = 10803,
5116 ["lotimes"] = 10804,
5117 ["rotimes"] = 10805,
5118 ["otimesas"] = 10806,
5119 ["Otimes"] = 10807,
5120 ["odiv"] = 10808,
5121 ["triplus"] = 10809,
5122 ["triminus"] = 10810,
5123 ["tritime"] = 10811,
5124 ["intprod"] = 10812,

```

```

5125 ["iprod"] = 10812,
5126 ["amalg"] = 10815,
5127 ["capdot"] = 10816,
5128 ["ncup"] = 10818,
5129 ["ncap"] = 10819,
5130 ["capand"] = 10820,
5131 ["cupor"] = 10821,
5132 ["cupcap"] = 10822,
5133 ["capcup"] = 10823,
5134 ["cupbrcap"] = 10824,
5135 ["capbrcup"] = 10825,
5136 ["cupcup"] = 10826,
5137 ["capcap"] = 10827,
5138 ["ccups"] = 10828,
5139 ["ccaps"] = 10829,
5140 ["ccupssm"] = 10832,
5141 ["And"] = 10835,
5142 ["Or"] = 10836,
5143 ["andand"] = 10837,
5144 ["oror"] = 10838,
5145 ["orslope"] = 10839,
5146 ["andslope"] = 10840,
5147 ["andv"] = 10842,
5148 ["orv"] = 10843,
5149 ["andd"] = 10844,
5150 ["ord"] = 10845,
5151 ["wedbar"] = 10847,
5152 ["sdote"] = 10854,
5153 ["simdot"] = 10858,
5154 ["congdots"] = 10861,
5155 ["ncongdots"] = {10861, 824},
5156 ["easter"] = 10862,
5157 ["apacir"] = 10863,
5158 ["apE"] = 10864,
5159 ["napE"] = {10864, 824},
5160 ["eplus"] = 10865,
5161 ["pluse"] = 10866,
5162 ["Esim"] = 10867,
5163 ["Colone"] = 10868,
5164 ["Equal"] = 10869,
5165 ["ddotseq"] = 10871,
5166 ["eDDot"] = 10871,
5167 ["equivDD"] = 10872,
5168 ["ltcir"] = 10873,
5169 ["gtcir"] = 10874,
5170 ["ltquest"] = 10875,
5171 ["gtquest"] = 10876,

```

```

5172 ["LessSlantEqual"] = 10877,
5173 ["NotLessSlantEqual"] = {10877, 824},
5174 ["leqslant"] = 10877,
5175 ["les"] = 10877,
5176 ["nleqslant"] = {10877, 824},
5177 ["nles"] = {10877, 824},
5178 ["GreaterSlantEqual"] = 10878,
5179 ["NotGreaterSlantEqual"] = {10878, 824},
5180 ["geqslant"] = 10878,
5181 ["ges"] = 10878,
5182 ["ngeqslant"] = {10878, 824},
5183 ["nges"] = {10878, 824},
5184 ["lesdot"] = 10879,
5185 ["gesdot"] = 10880,
5186 ["lesdoto"] = 10881,
5187 ["gesdoto"] = 10882,
5188 ["lesdotor"] = 10883,
5189 ["gesdoto1"] = 10884,
5190 ["lap"] = 10885,
5191 ["lessapprox"] = 10885,
5192 ["gap"] = 10886,
5193 ["gtrapprox"] = 10886,
5194 ["lne"] = 10887,
5195 ["lneq"] = 10887,
5196 ["gne"] = 10888,
5197 ["gneq"] = 10888,
5198 ["lnap"] = 10889,
5199 ["lnapprox"] = 10889,
5200 ["gnap"] = 10890,
5201 ["gnapprox"] = 10890,
5202 ["lEg"] = 10891,
5203 ["lesseqqgtr"] = 10891,
5204 ["gEl"] = 10892,
5205 ["gtreqqless"] = 10892,
5206 ["lsime"] = 10893,
5207 ["gsime"] = 10894,
5208 ["lsimg"] = 10895,
5209 ["gsiml"] = 10896,
5210 ["lgE"] = 10897,
5211 ["glE"] = 10898,
5212 ["lesges"] = 10899,
5213 ["gesles"] = 10900,
5214 ["els"] = 10901,
5215 ["eqslantless"] = 10901,
5216 ["egs"] = 10902,
5217 ["eqslantgtr"] = 10902,
5218 ["elsdot"] = 10903,

```

```

5219 ["egsdot"] = 10904,
5220 ["el"] = 10905,
5221 ["eg"] = 10906,
5222 ["siml"] = 10909,
5223 ["simg"] = 10910,
5224 ["simlE"] = 10911,
5225 ["simgE"] = 10912,
5226 ["LessLess"] = 10913,
5227 ["NotNestedLessLess"] = {10913, 824},
5228 ["GreaterGreater"] = 10914,
5229 ["NotNestedGreaterGreater"] = {10914, 824},
5230 ["glj"] = 10916,
5231 ["gla"] = 10917,
5232 ["ltcc"] = 10918,
5233 ["gtcc"] = 10919,
5234 ["lescc"] = 10920,
5235 ["gescc"] = 10921,
5236 ["smt"] = 10922,
5237 ["lat"] = 10923,
5238 ["smtE"] = 10924,
5239 ["smtes"] = {10924, 65024},
5240 ["late"] = 10925,
5241 ["lates"] = {10925, 65024},
5242 ["bumpE"] = 10926,
5243 ["NotPrecedesEqual"] = {10927, 824},
5244 ["PrecedesEqual"] = 10927,
5245 ["npre"] = {10927, 824},
5246 ["npreceq"] = {10927, 824},
5247 ["pre"] = 10927,
5248 ["preceq"] = 10927,
5249 ["NotSucceedsEqual"] = {10928, 824},
5250 ["SucceedsEqual"] = 10928,
5251 ["nsce"] = {10928, 824},
5252 ["nsucceq"] = {10928, 824},
5253 ["sce"] = 10928,
5254 ["succeq"] = 10928,
5255 ["prE"] = 10931,
5256 ["scE"] = 10932,
5257 ["precneqq"] = 10933,
5258 ["prnE"] = 10933,
5259 ["scnE"] = 10934,
5260 ["succneqq"] = 10934,
5261 ["prap"] = 10935,
5262 ["precapprox"] = 10935,
5263 ["scap"] = 10936,
5264 ["succapprox"] = 10936,
5265 ["precnapprox"] = 10937,

```

```

5266 ["prnap"] = 10937,
5267 ["scnap"] = 10938,
5268 ["succnapprox"] = 10938,
5269 ["Pr"] = 10939,
5270 ["Sc"] = 10940,
5271 ["subdot"] = 10941,
5272 ["supdot"] = 10942,
5273 ["subplus"] = 10943,
5274 ["supplus"] = 10944,
5275 ["submult"] = 10945,
5276 ["supmult"] = 10946,
5277 ["subedot"] = 10947,
5278 ["supedot"] = 10948,
5279 ["nsubE"] = {10949, 824},
5280 ["nsubseteqq"] = {10949, 824},
5281 ["subE"] = 10949,
5282 ["subseteqq"] = 10949,
5283 ["nsupE"] = {10950, 824},
5284 ["nsupseteqq"] = {10950, 824},
5285 ["supE"] = 10950,
5286 ["supseteqq"] = 10950,
5287 ["subsim"] = 10951,
5288 ["supsim"] = 10952,
5289 ["subnE"] = 10955,
5290 ["subsetneqq"] = 10955,
5291 ["varsubsetneqq"] = {10955, 65024},
5292 ["vsubnE"] = {10955, 65024},
5293 ["supnE"] = 10956,
5294 ["supsetneqq"] = 10956,
5295 ["varsupsetneqq"] = {10956, 65024},
5296 ["vsupnE"] = {10956, 65024},
5297 ["csub"] = 10959,
5298 ["csup"] = 10960,
5299 ["csube"] = 10961,
5300 ["csupe"] = 10962,
5301 ["subsup"] = 10963,
5302 ["supsub"] = 10964,
5303 ["subsub"] = 10965,
5304 ["supsup"] = 10966,
5305 ["suphsub"] = 10967,
5306 ["supdsub"] = 10968,
5307 ["forkv"] = 10969,
5308 ["topfork"] = 10970,
5309 ["mlcp"] = 10971,
5310 ["Dashv"] = 10980,
5311 ["DoubleLeftTee"] = 10980,
5312 ["Vdashl"] = 10982,

```

```

5313 ["Barv"] = 10983,
5314 ["vBar"] = 10984,
5315 ["vBarv"] = 10985,
5316 ["Vbar"] = 10987,
5317 ["Not"] = 10988,
5318 ["bNot"] = 10989,
5319 ["rnmid"] = 10990,
5320 ["cirmid"] = 10991,
5321 ["midcir"] = 10992,
5322 ["topcir"] = 10993,
5323 ["nhpar"] = 10994,
5324 ["parsim"] = 10995,
5325 ["nparsl"] = {11005, 8421},
5326 ["parsl"] = 11005,
5327 ["fflig"] = 64256,
5328 ["filig"] = 64257,
5329 ["fllig"] = 64258,
5330 ["ffilig"] = 64259,
5331 ["ffllig"] = 64260,
5332 ["Ascr"] = 119964,
5333 ["Cscr"] = 119966,
5334 ["Dscr"] = 119967,
5335 ["Gscr"] = 119970,
5336 ["Jscr"] = 119973,
5337 ["Kscr"] = 119974,
5338 ["Nscr"] = 119977,
5339 ["Oscr"] = 119978,
5340 ["Pscr"] = 119979,
5341 ["Qscr"] = 119980,
5342 ["Sscr"] = 119982,
5343 ["Tscr"] = 119983,
5344 ["Uscr"] = 119984,
5345 ["Vscr"] = 119985,
5346 ["Wscr"] = 119986,
5347 ["Xscr"] = 119987,
5348 ["Yscr"] = 119988,
5349 ["Zscr"] = 119989,
5350 ["ascr"] = 119990,
5351 ["bscr"] = 119991,
5352 ["cscr"] = 119992,
5353 ["dscr"] = 119993,
5354 ["fscr"] = 119995,
5355 ["hscr"] = 119997,
5356 ["iscr"] = 119998,
5357 ["jscr"] = 119999,
5358 ["kscr"] = 120000,
5359 ["lscr"] = 120001,

```

```

5360 ["mscr"] = 120002,
5361 ["nscr"] = 120003,
5362 ["pscr"] = 120005,
5363 ["qscr"] = 120006,
5364 ["rscr"] = 120007,
5365 ["sscr"] = 120008,
5366 ["tscr"] = 120009,
5367 ["uscr"] = 120010,
5368 ["vscr"] = 120011,
5369 ["wscr"] = 120012,
5370 ["xscr"] = 120013,
5371 ["yscr"] = 120014,
5372 ["zscr"] = 120015,
5373 ["Afr"] = 120068,
5374 ["Bfr"] = 120069,
5375 ["Dfr"] = 120071,
5376 ["Efr"] = 120072,
5377 ["Ffr"] = 120073,
5378 ["Gfr"] = 120074,
5379 ["Jfr"] = 120077,
5380 ["Kfr"] = 120078,
5381 ["Lfr"] = 120079,
5382 ["Mfr"] = 120080,
5383 ["Nfr"] = 120081,
5384 ["Ofr"] = 120082,
5385 ["Pfr"] = 120083,
5386 ["Qfr"] = 120084,
5387 ["Sfr"] = 120086,
5388 ["Tfr"] = 120087,
5389 ["Ufr"] = 120088,
5390 ["Vfr"] = 120089,
5391 ["Wfr"] = 120090,
5392 ["Xfr"] = 120091,
5393 ["Yfr"] = 120092,
5394 ["afr"] = 120094,
5395 ["bfr"] = 120095,
5396 ["cfr"] = 120096,
5397 ["dfr"] = 120097,
5398 ["efr"] = 120098,
5399 ["ffr"] = 120099,
5400 ["gfr"] = 120100,
5401 ["hfr"] = 120101,
5402 ["ifr"] = 120102,
5403 ["jfr"] = 120103,
5404 ["kfr"] = 120104,
5405 ["lfr"] = 120105,
5406 ["mfr"] = 120106,

```

```

5407 ["nfr"] = 120107,
5408 ["ofr"] = 120108,
5409 ["pfr"] = 120109,
5410 ["qfr"] = 120110,
5411 ["rfr"] = 120111,
5412 ["sfr"] = 120112,
5413 ["tfr"] = 120113,
5414 ["ufr"] = 120114,
5415 ["vfr"] = 120115,
5416 ["wfr"] = 120116,
5417 ["xfr"] = 120117,
5418 ["yfr"] = 120118,
5419 ["zfr"] = 120119,
5420 ["Aopf"] = 120120,
5421 ["Bopf"] = 120121,
5422 ["Dopf"] = 120123,
5423 ["Eopf"] = 120124,
5424 ["Fopf"] = 120125,
5425 ["Gopf"] = 120126,
5426 ["Iopf"] = 120128,
5427 ["Jopf"] = 120129,
5428 ["Kopf"] = 120130,
5429 ["Lopf"] = 120131,
5430 ["Mopf"] = 120132,
5431 ["Oopf"] = 120134,
5432 ["Sopf"] = 120138,
5433 ["Topf"] = 120139,
5434 ["Uopf"] = 120140,
5435 ["Vopf"] = 120141,
5436 ["Wopf"] = 120142,
5437 ["Xopf"] = 120143,
5438 ["Yopf"] = 120144,
5439 ["aopf"] = 120146,
5440 ["bopf"] = 120147,
5441 ["copf"] = 120148,
5442 ["dopf"] = 120149,
5443 ["eopf"] = 120150,
5444 ["fopf"] = 120151,
5445 ["gopf"] = 120152,
5446 ["hopf"] = 120153,
5447 ["iopf"] = 120154,
5448 ["jopf"] = 120155,
5449 ["kopf"] = 120156,
5450 ["lopf"] = 120157,
5451 ["mopf"] = 120158,
5452 ["nopf"] = 120159,
5453 ["oopf"] = 120160,

```



```

5454 ["popf"] = 120161,
5455 ["qopf"] = 120162,
5456 ["ropf"] = 120163,
5457 ["sopf"] = 120164,
5458 ["topf"] = 120165,
5459 ["uopf"] = 120166,
5460 ["vopf"] = 120167,
5461 ["wopf"] = 120168,
5462 ["xopf"] = 120169,
5463 ["yopf"] = 120170,
5464 ["zopf"] = 120171,
5465 }

```

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

```

5466 function entities.dec_entity(s)
5467   local n = tonumber(s)
5468   if n == nil then
5469     return "&#" .. s .. ";" -- fallback for unknown entities
5470   end
5471   return unicode.utf8.char(n)
5472 end

```

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

```

5473 function entities.hex_entity(s)
5474   local n = tonumber("0x"..s)
5475   if n == nil then
5476     return "&#x" .. s .. ";" -- fallback for unknown entities
5477   end
5478   return unicode.utf8.char(n)
5479 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.

```

5480 function entities.hex_entity_with_x_char(x, s)
5481   local n = tonumber("0x"..s)
5482   if n == nil then
5483     return "&#" .. x .. s .. ";" -- fallback for unknown entities
5484   end
5485   return unicode.utf8.char(n)
5486 end

```

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

```

5487 function entities.char_entity(s)
5488   local code_points = character_entities[s]

```

```

5489   if code_points == nil then
5490       return "&" .. s .. ";"
5491   end
5492   if type(code_points) ~= 'table' then
5493       code_points = {code_points}
5494   end
5495   local char_table = {}
5496   for _, code_point in ipairs(code_points) do
5497       table.insert(char_table, unicode.utf8.char(code_point))
5498   end
5499   return table.concat(char_table)
5500 end

```

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

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

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

```

5501 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.

```

5502 function M.writer.new(options)
5503     local self = {}

```

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

```

5504     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.

```

5505     self.flatten_inlines = false

```

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

```
5506 local slice_specifiers = {}
5507 for specifier in options.slice:gmatch("[^%s]+") do
5508     table.insert(slice_specifiers, specifier)
5509 end
5510
5511 if #slice_specifiers == 2 then
5512     self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
5513     local slice_begin_type = self.slice_begin:sub(1, 1)
5514     if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
5515         self.slice_begin = "^" .. self.slice_begin
5516     end
5517     local slice_end_type = self.slice_end:sub(1, 1)
5518     if slice_end_type ~= "^" and slice_end_type ~= "$" then
5519         self.slice_end = "$" .. self.slice_end
5520     end
5521 elseif #slice_specifiers == 1 then
5522     self.slice_begin = "^" .. slice_specifiers[1]
5523     self.slice_end = "$" .. slice_specifiers[1]
5524 end
5525
5526 self.slice_begin_type = self.slice_begin:sub(1, 1)
5527 self.slice_begin_identifier = self.slice_begin:sub(2) or ""
5528 self.slice_end_type = self.slice_end:sub(1, 1)
5529 self.slice_end_identifier = self.slice_end:sub(2) or ""
5530
5531 if self.slice_begin == "^" and self.slice_end ~= "^" then
5532     self.is_writing = true
5533 else
5534     self.is_writing = false
5535 end
```

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

```
5536 self.suffix = ".tex"
```

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

```
5537 self.space = " "
```

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

```
5538 self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
5539 function self.plain(s)
5540     return s
5541 end
```

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

```
5542 function self.paragraph(s)
5543   if not self.is_writing then return "" end
5544   return s
5545 end
```

Define `writer->pack` as a function that will take the filename `name` of the output file prepared by the reader and transform it to the output format.

```
5546 function self.pack(name)
5547   return [[input{}]] .. name .. [{}\relax]]
5548 end
```

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

```
5549 function self.interblocksep()
5550   if not self.is_writing then return "" end
5551   return "\\markdownRendererInterblockSeparator\n{}"
5552 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.

```
5553 function self.paragraphsep()
5554   if not self.is_writing then return "" end
5555   return "\\markdownRendererParagraphSeparator\n{}"
5556 end
```

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

```
5557 self.soft_line_break = function()
5558   if self.flatten_inlines then return "\n" end
5559   return "\\markdownRendererSoftLineBreak\n{}"
5560 end
```

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

```
5561 self.hard_line_break = function()
5562   if self.flatten_inlines then return "\n" end
5563   return "\\markdownRendererHardLineBreak\n{}"
5564 end
```

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

```
5565 self.ellipsis = "\\markdownRendererEllipsis{}"
```

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

```
5566 function self.thematic_break()
5567   if not self.is_writing then return "" end
5568   return "\\markdownRendererThematicBreak{}"
5569 end
```

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

```

5570 self.escaped_uri_chars = {
5571     ["{"] = "\\markdownRendererLeftBrace{}",
5572     ["}"] = "\\markdownRendererRightBrace{}",
5573     ["\\"] = "\\markdownRendererBackslash{}",
5574 }
5575 self.escaped_minimal_strings = {
5576     ["^"] = "\\markdownRendererCircumflex\\markdownRendererCircumflex ",
5577     ["☒"] = "\\markdownRendererTickedBox{}",
5578     ["☐"] = "\\markdownRendererHalfTickedBox{}",
5579     ["□"] = "\\markdownRendererUntickedBox{}",
5580     [entities.hex_entity('FFFD')] = "\\markdownRendererReplacementCharacter{}",
5581 }

```

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

```

5582 self.escaped_strings = util.table_copy(self.escaped_minimal_strings)
5583 self.escaped_strings[entities.hex_entity('00A0')] = self.nbsp

```

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

```

5584 self.escaped_chars = {
5585     ["{"] = "\\markdownRendererLeftBrace{}",
5586     ["}"] = "\\markdownRendererRightBrace{}",
5587     ["%"] = "\\markdownRendererPercentSign{}",
5588     ["\\"] = "\\markdownRendererBackslash{}",
5589     ["#"] = "\\markdownRendererHash{}",
5590     ["$"] = "\\markdownRendererDollarSign{}",
5591     ["&"] = "\\markdownRendererAmpersand{}",
5592     ["_"] = "\\markdownRendererUnderscore{}",
5593     ["^"] = "\\markdownRendererCircumflex{}",
5594     ["~"] = "\\markdownRendererTilde{}",
5595     ["|"] = "\\markdownRendererPipe{}",
5596     [entities.hex_entity('0000')] = "\\markdownRendererReplacementCharacter{}",
5597 }

```

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

```

5598 local function create_escaper(char_escapes, string_escapes)
5599     local escape = util.escaper(char_escapes, string_escapes)
5600     return function(s)
5601         if self.flatten_inlines then return s end
5602         return escape(s)

```

```

5603     end
5604 end
5605 local escape_typographic_text = create_escaper(
5606     self.escaped_chars, self.escaped_strings)
5607 local escape_programmatic_text = create_escaper(
5608     self.escaped_uri_chars, self.escaped_minimal_strings)
5609 local escape_minimal = create_escaper(
5610     {}, 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 URL.
- `writer->infostring` transforms a fence code infostring.

```

5611 self.escape = escape_typographic_text
5612 self.math = escape_minimal
5613 if options.hybrid then
5614     self.identifier = escape_minimal
5615     self.string = escape_minimal
5616     self.uri = escape_minimal
5617     self.infostring = escape_minimal
5618 else
5619     self.identifier = escape_programmatic_text
5620     self.string = escape_typographic_text
5621     self.uri = escape_programmatic_text
5622     self.infostring = escape_programmatic_text
5623 end

```

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

```

5624 function self.code(s, attributes)
5625     if self.flatten_inlines then return s end
5626     local buf = {}
5627     if attributes ~= nil then
5628         table.insert(buf,
5629             "\\markdownRendererCodeSpanAttributeContextBegin\n")
5630         table.insert(buf, self.attributes(attributes))
5631     end
5632     table.insert(buf,
5633         {"\\markdownRendererCodeSpan{", self.escape(s), "}")})
5634     if attributes ~= nil then

```

```

5635         table.insert(buf,
5636                       "\\markdownRendererCodeSpanAttributeContextEnd{")
5637     end
5638     return buf
5639 end

```

Define `writer->link` as a function that will transform an input hyperlink to the output format, where `lab` corresponds to the label, `src` to URI, `tit` to the title of the link, and `attributes` to optional attributes.

```

5640 function self.link(lab, src, tit, attributes)
5641     if self.flatten_inlines then return lab end
5642     local buf = {}
5643     if attributes ~= nil then
5644         table.insert(buf,
5645                       "\\markdownRendererLinkAttributeContextBegin\n")
5646         table.insert(buf, self.attributes(attributes))
5647     end
5648     table.insert(buf, {"\\markdownRendererLink{",lab,"",
5649                       "{",self.escape(src),"",
5650                       "{",self.uri(src),"",
5651                       "{",self.string(tit or ""),""})
5652     if attributes ~= nil then
5653         table.insert(buf,
5654                       "\\markdownRendererLinkAttributeContextEnd{")
5655     end
5656     return buf
5657 end

```

Define `writer->image` as a function that will transform an input image to the output format, where `lab` corresponds to the label, `src` to the URL, `tit` to the title of the image, and `attributes` to optional attributes.

```

5658 function self.image(lab, src, tit, attributes)
5659     if self.flatten_inlines then return lab end
5660     local buf = {}
5661     if attributes ~= nil then
5662         table.insert(buf,
5663                       "\\markdownRendererImageAttributeContextBegin\n")
5664         table.insert(buf, self.attributes(attributes))
5665     end
5666     table.insert(buf, {"\\markdownRendererImage{",lab,"",
5667                       "{",self.string(src),"",
5668                       "{",self.uri(src),"",
5669                       "{",self.string(tit or ""),""})
5670     if attributes ~= nil then
5671         table.insert(buf,
5672                       "\\markdownRendererImageAttributeContextEnd{")
5673     end

```

```

5674     return buf
5675 end

```

Define `writer->bulletlist` as a function that will transform an input bulleted list to the output format, where `items` is an array of the list items and `tight` specifies, whether the list is tight or not.

```

5676 function self.bulletlist(items,tight)
5677   if not self.is_writing then return "" end
5678   local buffer = {}
5679   for _,item in ipairs(items) do
5680     if item ~= "" then
5681       buffer[#buffer + 1] = self.bulletitem(item)
5682     end
5683   end
5684   local contents = util.intersperse(buffer,"\n")
5685   if tight and options.tightLists then
5686     return {"\\markdownRendererUBeginTight\n",contents,
5687       "\n\\markdownRendererUEndTight "}
5688   else
5689     return {"\\markdownRendererUBegin\n",contents,
5690       "\n\\markdownRendererUEnd "}
5691   end
5692 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.

```

5693 function self.bulletitem(s)
5694   return {"\\markdownRendererUItem ",s,
5695     "\\markdownRendererUItemEnd "}
5696 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.

```

5697 function self.orderedlist(items,tight,startnum)
5698   if not self.is_writing then return "" end
5699   local buffer = {}
5700   local num = startnum
5701   for _,item in ipairs(items) do
5702     if item ~= "" then
5703       buffer[#buffer + 1] = self.ordereditem(item,num)
5704     end
5705     if num ~= nil and item ~= "" then
5706       num = num + 1
5707     end
5708   end
5709   local contents = util.intersperse(buffer,"\n")

```



```

5710     if tight and options.tightLists then
5711         return {"\\markdownRenderer01BeginTight\\n",contents,
5712             "\\n\\markdownRenderer01EndTight "}
5713     else
5714         return {"\\markdownRenderer01Begin\\n",contents,
5715             "\\n\\markdownRenderer01End "}
5716     end
5717 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.

```

5718 function self.ordereditem(s,num)
5719     if num ~= nil then
5720         return {"\\markdownRenderer01ItemWithNumber{" ,num,"}",s,
5721             "\\markdownRenderer01ItemEnd "}
5722     else
5723         return {"\\markdownRenderer01Item ",s,
5724             "\\markdownRenderer01ItemEnd "}
5725     end
5726 end

```

Define `writer->inline_html_comment` as a function that will transform the contents of an inline HTML comment, to the output format, where `contents` are the contents of the HTML comment.

```

5727 function self.inline_html_comment(contents)
5728     if self.flatten_inlines then return contents end
5729     return {"\\markdownRendererInlineHtmlComment{" ,contents,""}
5730 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.

```

5731 function self.inline_html_tag(contents)
5732     if self.flatten_inlines then return contents end
5733     return {"\\markdownRendererInlineHtmlTag{" ,self.string(contents),""}
5734 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.

```

5735 function self.block_html_element(s)
5736     if not self.is_writing then return "" end
5737     local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
5738     return {"\\markdownRendererInputBlockHtmlElement{" ,name,""}
5739 end

```

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

```
5740 function self.emphasis(s)
5741   if self.flatten_inlines then return s end
5742   return {"\\markdownRendererEmphasis{" ,s,""} }
5743 end
```

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

```
5744 function self.checkbox(f)
5745   if f == 1.0 then
5746     return "☒ "
5747   elseif f == 0.0 then
5748     return "☐ "
5749   else
5750     return "◻ "
5751   end
5752 end
```

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

```
5753 function self.strong(s)
5754   if self.flatten_inlines then return s end
5755   return {"\\markdownRendererStrongEmphasis{" ,s,""} }
5756 end
```

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

```
5757 function self.blockquote(s)
5758   if not self.is_writing then return "" end
5759   return {"\\markdownRendererBlockQuoteBegin\\n",s,
5760         "\\n\\markdownRendererBlockQuoteEnd "}
5761 end
```

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

```
5762 function self.verbatim(s)
5763   if not self.is_writing then return "" end
5764   s = s:gsub("\\n$", "")
5765   local name = util.cache_verbatim(options.cacheDir, s)
5766   return {"\\markdownRendererInputVerbatim{" ,name,""} }
5767 end
```

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

```
5768 function self.document(d)
5769   local buf = {"\\markdownRendererDocumentBegin\\n", d}
5770
```

```

5771     -- pop all attributes
5772     table.insert(buf, self.pop_attributes())
5773
5774     table.insert(buf, "\\markdownRendererDocumentEnd")
5775
5776     return buf
5777 end

```

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

```

5778 local seen_identifiers = {}
5779 local key_value_regex = "([^\s= ]+)%s*=%s*(.*)"
5780 local function normalize_attributes(attributes, auto_identifiers)
5781     -- normalize attributes
5782     local normalized_attributes = {}
5783     local has_explicit_identifiers = false
5784     local key, value
5785     for _, attribute in ipairs(attributes or {}) do
5786         if attribute:sub(1, 1) == "#" then
5787             table.insert(normalized_attributes, attribute)
5788             has_explicit_identifiers = true
5789             seen_identifiers[attribute:sub(2)] = true
5790         elseif attribute:sub(1, 1) == "." then
5791             table.insert(normalized_attributes, attribute)
5792         else
5793             key, value = attribute:match(key_value_regex)
5794             if key:lower() == "id" then
5795                 table.insert(normalized_attributes, "#" .. value)
5796             elseif key:lower() == "class" then
5797                 local classes = {}
5798                 for class in value:gmatch("%S+") do
5799                     table.insert(classes, class)
5800                 end
5801                 table.sort(classes)
5802                 for _, class in ipairs(classes) do
5803                     table.insert(normalized_attributes, "." .. class)
5804                 end
5805             else
5806                 table.insert(normalized_attributes, attribute)
5807             end
5808         end
5809     end
5810
5811     -- if no explicit identifiers exist, add auto identifiers
5812     if not has_explicit_identifiers and auto_identifiers ~= nil then
5813         local seen_auto_identifiers = {}
5814         for _, auto_identifier in ipairs(auto_identifiers) do

```

```

5815         if seen_auto_identifiers[auto_identifier] == nil then
5816             seen_auto_identifiers[auto_identifier] = true
5817         if seen_identifiers[auto_identifier] == nil then
5818             seen_identifiers[auto_identifier] = true
5819             table.insert(normalized_attributes,
5820                 "#" .. auto_identifier)
5821         else
5822             local auto_identifier_number = 1
5823             while true do
5824                 local numbered_auto_identifier = auto_identifier .. "-"
5825                                     .. auto_identifier_number
5826                 if seen_identifiers[numbered_auto_identifier] == nil then
5827                     seen_identifiers[numbered_auto_identifier] = true
5828                     table.insert(normalized_attributes,
5829                         "#" .. numbered_auto_identifier)
5830                     break
5831                 end
5832                 auto_identifier_number = auto_identifier_number + 1
5833             end
5834         end
5835     end
5836 end
5837 end
5838
5839 -- sort and deduplicate normalized attributes
5840 table.sort(normalized_attributes)
5841 local seen_normalized_attributes = {}
5842 local deduplicated_normalized_attributes = {}
5843 for _, attribute in ipairs(normalized_attributes) do
5844     if seen_normalized_attributes[attribute] == nil then
5845         seen_normalized_attributes[attribute] = true
5846         table.insert(deduplicated_normalized_attributes, attribute)
5847     end
5848 end
5849
5850 return deduplicated_normalized_attributes
5851 end
5852
5853 function self.attributes(attributes, should_normalize_attributes)
5854     local normalized_attributes
5855     if should_normalize_attributes == false then
5856         normalized_attributes = attributes
5857     else
5858         normalized_attributes = normalize_attributes(attributes)
5859     end
5860
5861     local buf = {}

```

```

5862     local key, value
5863     for _, attribute in ipairs(normalized_attributes) do
5864         if attribute:sub(1, 1) == "#" then
5865             table.insert(buf, {"\\markdownRendererAttributeIdentifier{" ,
5866                               attribute:sub(2), "}"})
5867         elseif attribute:sub(1, 1) == "." then
5868             table.insert(buf, {"\\markdownRendererAttributeName{" ,
5869                               attribute:sub(2), "}"})
5870         else
5871             key, value = attribute:match(key_value_regex)
5872             table.insert(buf, {"\\markdownRendererAttributeKeyValue{" ,
5873                               key, "{" , value, "}"})
5874         end
5875     end
5876
5877     return buf
5878 end

```

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

```

5879     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`.

```

5880     self.attribute_type_levels = {}
5881     setmetatable(self.attribute_type_levels,
5882                 { __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`.

```

5883     local function apply_attributes()
5884         local buf = {}
5885         for i = 1, #self.active_attributes do
5886             local start_output = self.active_attributes[i][3]
5887             if start_output ~= nil then
5888                 table.insert(buf, start_output)
5889             end
5890         end
5891         return buf
5892     end
5893
5894     local function tear_down_attributes()
5895         local buf = {}
5896         for i = #self.active_attributes, 1, -1 do
5897             local end_output = self.active_attributes[i][4]
5898             if end_output ~= nil then
5899                 table.insert(buf, end_output)

```

```

5900     end
5901     end
5902     return buf
5903 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.

```

5904 function self.push_attributes(attribute_type, attributes,
5905                               start_output, end_output)
5906     local attribute_type_level = self.attribute_type_levels[attribute_type]
5907     self.attribute_type_levels[attribute_type] = attribute_type_level + 1
5908
5909     -- index attributes in a hash table for easy lookup
5910     attributes = attributes or {}
5911     for i = 1, #attributes do
5912         attributes[attributes[i]] = true
5913     end
5914
5915     local buf = {}
5916     -- handle slicing
5917     if attributes["#" .. self.slice_end_identifier] ~= nil and
5918        self.slice_end_type == "^" then
5919         if self.is_writing then
5920             table.insert(buf, tear_down_attributes())
5921         end
5922         self.is_writing = false
5923     end
5924     if attributes["#" .. self.slice_begin_identifier] ~= nil and
5925        self.slice_begin_type == "^" then
5926         table.insert(buf, apply_attributes())
5927         self.is_writing = true
5928     end
5929     if self.is_writing and start_output ~= nil then
5930         table.insert(buf, start_output)
5931     end
5932     table.insert(self.active_attributes,
5933                 {attribute_type, attributes,
5934                  start_output, end_output})
5935     return buf
5936 end
5937

```

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`).

```

5938 function self.pop_attributes(attribute_type)
5939     local buf = {}
5940     -- pop attributes until we find attributes of correct type
5941     -- or until no attributes remain
5942     local current_attribute_type = false
5943     while current_attribute_type ~= attribute_type and
5944         #self.active_attributes > 0 do
5945         local attributes, _, end_output
5946         current_attribute_type, attributes, _, end_output = table.unpack(
5947             self.active_attributes[#self.active_attributes])
5948         local attribute_type_level = self.attribute_type_levels[current_attribute_type]
5949         self.attribute_type_levels[current_attribute_type] = attribute_type_level - 1
5950         if self.is_writing and end_output ~= nil then
5951             table.insert(buf, end_output)
5952         end
5953         table.remove(self.active_attributes, #self.active_attributes)
5954         -- handle slicing
5955         if attributes["#" .. self.slice_end_identifier] ~= nil
5956             and self.slice_end_type == "$" then
5957             if self.is_writing then
5958                 table.insert(buf, tear_down_attributes())
5959             end
5960             self.is_writing = false
5961         end
5962         if attributes["#" .. self.slice_begin_identifier] ~= nil and
5963             self.slice_begin_type == "$" then
5964             self.is_writing = true
5965             table.insert(buf, apply_attributes())
5966         end
5967     end
5968     return buf
5969 end

```

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

```

5970 local function create_auto_identifier(s)
5971     local buffer = {}
5972     local prev_space = false
5973     local letter_found = false
5974
5975     for _, code in utf8.codes(uni_algos.normalize.NFC(s)) do
5976         local char = utf8.char(code)
5977
5978         -- Remove everything up to the first letter.

```

```

5979     if not letter_found then
5980         local is_letter = unicode.utf8.match(char, "%a")
5981         if is_letter then
5982             letter_found = true
5983         else
5984             goto continue
5985         end
5986     end
5987
5988     -- Remove all non-alphanumeric characters, except underscores, hyphens, and periods.
5989     if not unicode.utf8.match(char, "[%w_-%.%s]") then
5990         goto continue
5991     end
5992
5993     -- Replace all spaces and newlines with hyphens.
5994     if unicode.utf8.match(char, "[%s\\n]") then
5995         char = "-"
5996         if prev_space then
5997             goto continue
5998         else
5999             prev_space = true
6000         end
6001     else
6002         -- Convert all alphabetic characters to lowercase.
6003         char = unicode.utf8.lower(char)
6004         prev_space = false
6005     end
6006
6007     table.insert(buffer, char)
6008
6009     ::continue::
6010 end
6011
6012 if prev_space then
6013     table.remove(buffer)
6014 end
6015
6016 local identifier = #buffer == 0 and "section" or table.concat(buffer, "")
6017 return identifier
6018 end

```

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

```

6019 local function create_gfm_auto_identifier(s)
6020     local buffer = {}
6021     local prev_space = false
6022     local letter_found = false

```



```

6023
6024 for _, code in utf8.codes(uni_algos.normalize.NFC(s)) do
6025     local char = utf8.char(code)
6026
6027     -- Remove everything up to the first non-space.
6028     if not letter_found then
6029         local is_letter = unicode.utf8.match(char, "%S")
6030         if is_letter then
6031             letter_found = true
6032         else
6033             goto continue
6034         end
6035     end
6036
6037     -- Remove all non-alphanumeric characters, except underscores and hyphens.
6038     if not unicode.utf8.match(char, "[%w_-%s]") then
6039         prev_space = false
6040         goto continue
6041     end
6042
6043     -- Replace all spaces and newlines with hyphens.
6044     if unicode.utf8.match(char, "[%s\n]") then
6045         char = "-"
6046         if prev_space then
6047             goto continue
6048         else
6049             prev_space = true
6050         end
6051     else
6052         -- Convert all alphabetic characters to lowercase.
6053         char = unicode.utf8.lower(char)
6054         prev_space = false
6055     end
6056
6057     table.insert(buffer, char)
6058
6059     ::continue::
6060 end
6061
6062 if prev_space then
6063     table.remove(buffer)
6064 end
6065
6066 local identifier = #buffer == 0 and "section" or table.concat(buffer, "")
6067 return identifier
6068 end

```

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

```

6069 function self.heading(s, level, attributes)
6070     local buf = {}
6071     local flat_text, inlines = table.unpack(s)
6072
6073     -- push empty attributes for implied sections
6074     while self.attribute_type_levels["heading"] < level - 1 do
6075         table.insert(buf,
6076                     self.push_attributes("heading",
6077                                         nil,
6078                                         "\\markdownRendererSectionBegin\\n",
6079                                         "\\n\\markdownRendererSectionEnd "))
6080     end
6081
6082     -- pop attributes for sections that have ended
6083     while self.attribute_type_levels["heading"] >= level do
6084         table.insert(buf, self.pop_attributes("heading"))
6085     end
6086
6087     -- construct attributes for the new section
6088     local auto_identifiers = {}
6089     if self.options.autoIdentifiers then
6090         table.insert(auto_identifiers, create_auto_identifier(flat_text))
6091     end
6092     if self.options.gfmAutoIdentifiers then
6093         table.insert(auto_identifiers, create_gfm_auto_identifier(flat_text))
6094     end
6095     local normalized_attributes = normalize_attributes(attributes, auto_identifiers)
6096
6097     -- push attributes for the new section
6098     local start_output = {}
6099     local end_output = {}
6100     table.insert(start_output, "\\markdownRendererSectionBegin\\n")
6101     table.insert(end_output, "\\n\\markdownRendererSectionEnd ")
6102
6103     table.insert(buf, self.push_attributes("heading",
6104                                         normalized_attributes,
6105                                         start_output,
6106                                         end_output))
6107     assert(self.attribute_type_levels["heading"] == level)
6108
6109     -- render the heading and its attributes
6110     if self.is_writing and #normalized_attributes > 0 then
6111         table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin\\n")
6112         table.insert(buf, self.attributes(normalized_attributes, false))
6113     end

```

```

6114
6115     local cmd
6116     level = level + options.shiftHeadings
6117     if level <= 1 then
6118         cmd = "\\markdownRendererHeadingOne"
6119     elseif level == 2 then
6120         cmd = "\\markdownRendererHeadingTwo"
6121     elseif level == 3 then
6122         cmd = "\\markdownRendererHeadingThree"
6123     elseif level == 4 then
6124         cmd = "\\markdownRendererHeadingFour"
6125     elseif level == 5 then
6126         cmd = "\\markdownRendererHeadingFive"
6127     elseif level >= 6 then
6128         cmd = "\\markdownRendererHeadingSix"
6129     else
6130         cmd = ""
6131     end
6132     if self.is_writing then
6133         table.insert(buf, {cmd, "{", inlines, "}"})
6134     end
6135
6136     if self.is_writing and #normalized_attributes > 0 then
6137         table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd ")
6138     end
6139
6140     return buf
6141 end

```

Define `writer->get_state` as a function that returns the current state of the writer, where the state of a writer are its mutable member variables.

```

6142 function self.get_state()
6143     return {
6144         is_writing=self.is_writing,
6145         flatten_inlines=self.flatten_inlines,
6146         active_attributes={table.unpack(self.active_attributes)},
6147     }
6148 end

```

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

```

6149 function self.set_state(s)
6150     local previous_state = self.get_state()
6151     for key, value in pairs(s) do
6152         self[key] = value
6153     end
6154     return previous_state
6155 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`.

```

6156 function self.defer_call(f)
6157     local previous_state = self.get_state()
6158     return function(...)
6159         local state = self.set_state(previous_state)
6160         local return_value = f(...)
6161         self.set_state(state)
6162         return return_value
6163     end
6164 end
6165
6166 return self
6167 end

```

### 3.1.4 Parsers

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

```

6168 local parsers = {}

```

#### 3.1.4.1 Basic Parsers

```

6169 parsers.percent = P("%")
6170 parsers.at = P("@")
6171 parsers.comma = P(",")
6172 parsers.asterisk = P("*")
6173 parsers.dash = P("-")
6174 parsers.plus = P("+")
6175 parsers.underscore = P("_")
6176 parsers.period = P(".")
6177 parsers.hash = P("#")
6178 parsers.dollar = P("$")
6179 parsers.ampersand = P("&")
6180 parsers.backtick = P("`")
6181 parsers.less = P("<")
6182 parsers.more = P(">")
6183 parsers.space = P(" ")
6184 parsers.squote = P("'")
6185 parsers.dquote = P('"')
6186 parsers.lparent = P("(")
6187 parsers.rparent = P(")")
6188 parsers.lbracket = P("[")
6189 parsers.rbracket = P("]")
6190 parsers.lbrace = P("{")

```

```

6191 parsers.rbrace           = P("}")
6192 parsers.circumflex        = P("^")
6193 parsers.slash              = P("/")
6194 parsers.equal              = P("=")
6195 parsers.colon              = P(":")
6196 parsers.semicolon         = P(";")
6197 parsers.exclamation        = P("!")
6198 parsers.pipe               = P("|")
6199 parsers.tilde              = P("~")
6200 parsers.backslash          = P("\\")
6201 parsers.tab                = P("\t")
6202 parsers.newline            = P("\n")
6203
6204 parsers.digit              = R("09")
6205 parsers.hexdigit           = R("09","af","AF")
6206 parsers.letter             = R("AZ","az")
6207 parsers.alphanumeric       = R("AZ","az","09")
6208 parsers.keyword            = parsers.letter
6209                             * (parsers.alphanumeric + parsers.dash)^0
6210 parsers.internal_punctuation = S(":;,.?")
6211
6212 parsers.doubleasterisks    = P("**")
6213 parsers.doubleunderscores  = P("__")
6214 parsers.doubletildes       = P("~")
6215 parsers.fourspace         = P("    ")
6216
6217 parsers.any                = P(1)
6218 parsers.succeed            = P(true)
6219 parsers.fail               = P(false)
6220
6221 parsers.escapable          = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")
6222 parsers.anyescaped         = parsers.backslash / " " * parsers.escapable
6223                             + parsers.any
6224
6225 parsers.spacechar          = S("\t ")
6226 parsers.spacing            = S(" \n\r\t")
6227 parsers.nonspacechar       = parsers.any - parsers.spacing
6228 parsers.optionalspace      = parsers.spacechar^0
6229
6230 parsers.normalchar         = parsers.any - (V("SpecialChar")
6231                                         + parsers.spacing)
6232 parsers.eof                 = -parsers.any
6233 parsers.nonindentSPACE     = parsers.space^-3 * - parsers.spacechar
6234 parsers.indent             = parsers.space^-3 * parsers.tab
6235                             + parsers.fourspace / " "
6236 parsers.linechar           = P(1 - parsers.newline)
6237

```

```

6238 parsers.blankline           = parsers.optionalspace
6239                             * parsers.newline / "\n"
6240 parsers.blanklines           = parsers.blankline^0
6241 parsers.skipblanklines       = (parsers.optionalspace * parsers.newline)^0
6242 parsers.indentedline         = parsers.indent /""
6243                             * C(parsers.linechar^1 * parsers.newline^-
1)
6244 parsers.optionallyindentedline = parsers.indent^-1 /""
6245                             * C(parsers.linechar^1 * parsers.newline^-
1)
6246 parsers.sp                   = parsers.spacing^0
6247 parsers.spnl                 = parsers.optionalspace
6248                             * (parsers.newline * parsers.optionalspace)^-
1
6249 parsers.line                  = parsers.linechar^0 * parsers.newline
6250 parsers.nonemptyline         = parsers.line - parsers.blankline

```

### 3.1.4.2 Parsers Used for Indentation

```

6251
6252 parsers.leader              = parsers.space^-3
6253

```

Check if a trail exists and is non-empty in the indent table `indent_table`.

```

6254 local function has_trail(indent_table)
6255   return indent_table ~= nil and
6256     indent_table.trail ~= nil and
6257     next(indent_table.trail) ~= nil
6258 end
6259

```

Check if indent table `indent_table` has any indents.

```

6260 local function has_indents(indent_table)
6261   return indent_table ~= nil and
6262     indent_table.indents ~= nil and
6263     next(indent_table.indents) ~= nil
6264 end
6265

```

Add a trail `trail_info` to the indent table `indent_table`.

```

6266 local function add_trail(indent_table, trail_info)
6267   indent_table.trail = trail_info
6268   return indent_table
6269 end
6270

```

Remove a trail `trail_info` from the indent table `indent_table`.

```

6271 local function remove_trail(indent_table)
6272   indent_table.trail = nil

```

```

6273     return indent_table
6274 end
6275

```

Update the indent table `indent_table` by adding or removing a new indent `add`.

```

6276 local function update_indent_table(indent_table, new_indent, add)
6277     indent_table = remove_trail(indent_table)
6278
6279     if not has_indents(indent_table) then
6280         indent_table.indents = {}
6281     end
6282
6283
6284     if add then
6285         indent_table.indents[#indent_table.indents + 1] = new_indent
6286     else
6287         if indent_table.indents[#indent_table.indents].name == new_indent.name then
6288             indent_table.indents[#indent_table.indents] = nil
6289         end
6290     end
6291
6292     return indent_table
6293 end
6294

```

Remove an indent by its name `name`.

```

6295 local function remove_indent(name)
6296     local function remove_indent_level(s, i, indent_table) -- luacheck: ignore s i
6297         indent_table = update_indent_table(indent_table, {name=name}, false)
6298         return true, indent_table
6299     end
6300
6301     return Cg(Cmt(Cb("indent_info"), remove_indent_level), "indent_info")
6302 end
6303

```

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`.

```

6304 local function process_starter_spacing(indent, spacing, minimum, left_strip_length)
6305     left_strip_length = left_strip_length or 0
6306
6307     local count = 0
6308     local tab_value = 4 - (indent) % 4
6309
6310     local code_started, minimum_found = false, false

```

```

6311 local code_start, minimum_remainder = "", ""
6312
6313 local left_total_stripped = 0
6314 local full_remainder = ""
6315
6316 if spacing ~= nil then
6317     for i = 1, #spacing do
6318         local character = spacing:sub(i, i)
6319
6320         if character == "\t" then
6321             count = count + tab_value
6322             tab_value = 4
6323         elseif character == " " then
6324             count = count + 1
6325             tab_value = 4 - (1 - tab_value) % 4
6326         end
6327
6328         if (left_strip_length ~= 0) then
6329             local possible_to_strip = math.min(count, left_strip_length)
6330             count = count - possible_to_strip
6331             left_strip_length = left_strip_length - possible_to_strip
6332             left_total_stripped = left_total_stripped + possible_to_strip
6333         else
6334             full_remainder = full_remainder .. character
6335         end
6336
6337         if (minimum_found) then
6338             minimum_remainder = minimum_remainder .. character
6339         elseif (count >= minimum) then
6340             minimum_found = true
6341             minimum_remainder = minimum_remainder .. string.rep(" ", count - minimum)
6342         end
6343
6344         if (code_started) then
6345             code_start = code_start .. character
6346         elseif (count >= minimum + 4) then
6347             code_started = true
6348             code_start = code_start .. string.rep(" ", count - (minimum + 4))
6349         end
6350     end
6351 end
6352
6353 local remainder
6354 if (code_started) then
6355     remainder = code_start
6356 else
6357     remainder = string.rep(" ", count - minimum)

```



```

6358 end
6359
6360 local is_minimum = count >= minimum
6361 return {
6362     is_code = code_started,
6363     remainder = remainder,
6364     left_total_stripped = left_total_stripped,
6365     is_minimum = is_minimum,
6366     minimum_remainder = minimum_remainder,
6367     total_length = count,
6368     full_remainder = full_remainder
6369 }
6370 end
6371

```

Count the total width of all indents in the indent table `indent_table`.

```

6372 local function count_indent_tab_level(indent_table)
6373     local count = 0
6374     if not has_indents(indent_table) then
6375         return count
6376     end
6377
6378     for i=1, #indent_table.indents do
6379         count = count + indent_table.indents[i].length
6380     end
6381     return count
6382 end
6383

```

Count the total width of a delimiter `delimiter`.

```

6384 local function total_delimiter_length(delimiter)
6385     local count = 0
6386     if type(delimiter) == "string" then return #delimiter end
6387     for _, value in pairs(delimiter) do
6388         count = count + total_delimiter_length(value)
6389     end
6390     return count
6391 end
6392

```

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`.

```

6393 local function process_starter_indent(_, _, indent_table, starter, is_blank, indent_t
6394     local last_trail = starter[1]
6395     local delimiter = starter[2]
6396     local raw_new_trail = starter[3]
6397
6398     if indent_type == "bq" and not breakable then

```

```

6399     indent_table.ignore_blockquote_blank = true
6400 end
6401
6402 if has_trail(indent_table) then
6403     local trail = indent_table.trail
6404     if trail.is_code then
6405         return false
6406     end
6407     last_trail = trail.remainder
6408 else
6409     local sp = process_starter_spacing(0, last_trail, 0, 0)
6410
6411     if sp.is_code then
6412         return false
6413     end
6414     last_trail = sp.remainder
6415 end
6416
6417 local preceding_indentation = count_indent_tab_level(indent_table) % 4
6418 local last_trail_length = #last_trail
6419 local delimiter_length = total_delimiter_length(delimiter)
6420
6421 local total_indent_level = preceding_indentation + last_trail_length + delimiter_le
6422
6423 local sp = {}
6424 if not is_blank then
6425     sp = process_starter_spacing(total_indent_level, raw_new_trail, 0, 1)
6426 end
6427
6428 local del_trail_length = sp.left_total_stripped
6429 if is_blank then
6430     del_trail_length = 1
6431 elseif not sp.is_code then
6432     del_trail_length = del_trail_length + #sp.remainder
6433 end
6434
6435 local indent_length = last_trail_length + delimiter_length + del_trail_length
6436 local new_indent_info = {name=indent_type, length=indent_length}
6437
6438 indent_table = update_indent_table(indent_table, new_indent_info, true)
6439 indent_table = add_trail(indent_table, {is_code=sp.is_code, remainder=sp.remainder,
6440                                     full_remainder=sp.full_remainder})
6441
6442 return true, indent_table
6443 end
6444

```

Return the pattern corresponding with the indent name `name`.

```
6445 local function decode_pattern(name)
6446   local delimiter = parsers.succeed
6447   if name == "bq" then
6448     delimiter = parsers.more
6449   end
6450
6451   return C(parsers.optionalspace) * C(delimiter) * C(parsers.optionalspace) * Cp()
6452 end
6453
```

Find the first blank-only indent of the indent table `indent_table` followed by blank-only indents.

```
6454 local function left_blank_starter(indent_table)
6455   local blank_starter_index
6456
6457   if not has_indents(indent_table) then
6458     return
6459   end
6460
6461   for i = #indent_table.indents,1,-1 do
6462     local value = indent_table.indents[i]
6463     if value.name == "li" then
6464       blank_starter_index = i
6465     else
6466       break
6467     end
6468   end
6469
6470   return blank_starter_index
6471 end
6472
```

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.

```
6473 local function traverse_indent(s, i, indent_table, is_optional, is_blank)
6474   local new_index = i
6475
6476   local preceding_indentation = 0
6477   local current_trail = {}
6478
6479   local blank_starter = left_blank_starter(indent_table)
6480
6481   for index = 1, #indent_table.indents do
```

```

6482     local value = indent_table.indents[index]
6483     local pattern = decode_pattern(value.name)
6484
6485     -- match decoded pattern
6486     local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
6487     if new_indent_info == nil then
6488         local blankline_end = lpeg.match(Ct(parsers.blankline * Cg(Cp(), "pos")), s, new_index)
6489         if is_optional or not indent_table.ignore_blockquote_blank or not blankline_end then
6490             return is_optional, new_index, current_trail
6491         end
6492
6493         return traverse_indent(s, tonumber(blankline_end.pos), indent_table, is_optional)
6494     end
6495
6496     local raw_last_trail = new_indent_info[1]
6497     local delimiter = new_indent_info[2]
6498     local raw_new_trail = new_indent_info[3]
6499     local next_index = new_indent_info[4]
6500
6501     local space_only = delimiter == ""
6502
6503     -- check previous trail
6504     if not space_only and next(current_trail) == nil then
6505         local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
6506         current_trail = {is_code=sp.is_code, remainder=sp.remainder, total_length=sp.total_length,
6507             full_remainder=sp.full_remainder}
6508     end
6509
6510     if next(current_trail) ~= nil then
6511         if not space_only and current_trail.is_code then
6512             return is_optional, new_index, current_trail
6513         end
6514         if current_trail.internal_remainder ~= nil then
6515             raw_last_trail = current_trail.internal_remainder
6516         end
6517     end
6518
6519     local raw_last_trail_length = 0
6520     local delimiter_length = 0
6521
6522     if not space_only then
6523         delimiter_length = #delimiter
6524         raw_last_trail_length = #raw_last_trail
6525     end
6526
6527     local total_indent_level = preceding_indentation + raw_last_trail_length + delimiter_length
6528

```

```

6529     local spacing_to_process
6530     local minimum = 0
6531     local left_strip_length = 0
6532
6533     if not space_only then
6534         spacing_to_process = raw_new_trail
6535         left_strip_length = 1
6536     else
6537         spacing_to_process = raw_last_trail
6538         minimum = value.length
6539     end
6540
6541     local sp = process_starter_spacing(total_indent_level, spacing_to_process, minimum)
6542
6543     if space_only and not sp.is_minimum then
6544         return is_optional or (is_blank and blank_starter <= index), new_index, current_trail
6545     end
6546
6547     local indent_length = raw_last_trail_length + delimiter_length + sp.left_total_strip_length
6548
6549     -- update info for the next pattern
6550     if not space_only then
6551         preceding_indentation = preceding_indentation + indent_length
6552     else
6553         preceding_indentation = preceding_indentation + value.length
6554     end
6555
6556     current_trail = {is_code=sp.is_code, remainder=sp.remainder, internal_remainder=sp.internal_remainder,
6557                     total_length=sp.total_length, full_remainder=sp.full_remainder}
6558     new_index = next_index
6559 end
6560
6561 return true, new_index, current_trail
6562 end
6563

```

Check if a code trail is expected.

```

6564 local function check_trail(expect_code, is_code)
6565     return (expect_code and is_code) or (not expect_code and not is_code)
6566 end
6567

```

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`.

```

6568 local function check_trail_joined(s, i, indent_table, spacing, expect_code, omit_remainder)
6569     local is_code
6570     local remainder

```

```

6571
6572 if has_trail(indent_table) then
6573     local trail = indent_table.trail
6574     is_code = trail.is_code
6575     if is_code then
6576         remainder = trail.remainder
6577     else
6578         remainder = trail.full_remainder
6579     end
6580 else
6581     local sp = process_starter_spacing(0, spacing, 0, 0)
6582     is_code = sp.is_code
6583     if is_code then
6584         remainder = sp.remainder
6585     else
6586         remainder = sp.full_remainder
6587     end
6588 end
6589
6590 local result = check_trail(expect_code, is_code)
6591 if omit_remainder then
6592     return result
6593 end
6594 return result, remainder
6595 end
6596

```

Check if the current trail of the `indent_table` is of length between `min` and `max`.

```

6597 local function check_trail_length(s, i, indent_table, spacing, min, max) -- luacheck:
6598     local trail
6599
6600     if has_trail(indent_table) then
6601         trail = indent_table.trail
6602     else
6603         trail = process_starter_spacing(0, spacing, 0, 0)
6604     end
6605
6606     local total_length = trail.total_length
6607     if total_length == nil then
6608         return false
6609     end
6610
6611     return min <= total_length and total_length <= max
6612 end
6613

```

Check the indentation of the continuation line, optionally with the mode `is_optional` selected. Check blank line exclusively with `is_blank`.

```

6614 local function check_continuation_indentation(s, i, indent_table, is_optional, is_bla
6615     if not has_indents(indent_table) then
6616         return true
6617     end
6618
6619     local passes, new_index, current_trail = traverse_indent(s, i, indent_table, is_opt
6620
6621     if passes then
6622         indent_table = add_trail(indent_table, current_trail)
6623         return new_index, indent_table
6624     end
6625     return false
6626 end
6627

```

Get name of the last indent from the `indent_table`.

```

6628 local function get_last_indent_name(indent_table)
6629     if has_indents(indent_table) then
6630         return indent_table.indents[#indent_table.indents].name
6631     end
6632 end
6633

```

Remove the remainder altogether if the last indent from the `indent_table` is blank-only.

```

6634 local function remove_remainder_if_blank(indent_table, remainder)
6635     if get_last_indent_name(indent_table) == "li" then
6636         return ""
6637     end
6638     return remainder
6639 end
6640

```

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.

```

6641 local function check_trail_type(s, i, trail, spacing, trail_type) -- luacheck: ignore
6642     if trail == nil then
6643         trail = process_starter_spacing(0, spacing, 0, 0)
6644     end
6645
6646     if trail_type == "non-code" then
6647         return check_trail(false, trail.is_code)
6648     end
6649     if trail_type == "code" then
6650         return check_trail(true, trail.is_code)
6651     end
6652     if trail_type == "full-code" then
6653         if (trail.is_code) then

```

```

6654     return i, trail.remainder
6655 end
6656 return i, ""
6657 end
6658 if trail_type == "full-any" then
6659     return i, trail.internal_remainder
6660 end
6661 end
6662

```

Stores or restores an `is_freezing` trail from indent table `indent_table`.

```

6663 local function trail_freezing(s, i, indent_table, is_freezing) -- luacheck: ignore s
6664     if is_freezing then
6665         if indent_table.is_trail_frozen then
6666             indent_table.trail = indent_table.frozen_trail
6667         else
6668             indent_table.frozen_trail = indent_table.trail
6669             indent_table.is_trail_frozen = true
6670         end
6671     else
6672         indent_table.frozen_trail = nil
6673         indent_table.is_trail_frozen = false
6674     end
6675     return true, indent_table
6676 end
6677

```

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`.

```

6678 local function check_continuation_indentation_and_trail(s, i, indent_table, is_optional,
6679                                                         reset_rem, omit_remainder)
6680     if not has_indents(indent_table) then
6681         local spacing, new_index = lpeg.match(C(parsers.spacechar^0) * Cp(), s, i)
6682         local result, remainder = check_trail_type(s, i, indent_table.trail, spacing, trail_type)
6683         if remainder == nil then
6684             if result then
6685                 return new_index
6686             end
6687             return false
6688         end
6689         if result then
6690             return new_index, remainder
6691         end
6692         return false
6693     end
6694
6695     local passes, new_index, current_trail = traverse_indent(s, i, indent_table, is_optional,

```



```

6696
6697   if passes then
6698       local spacing
6699       if current_trail == nil then
6700           local newer_spacing, newer_index = lpeg.match(C(parsers.spacechar^0) * Cp(), s,
6701               current_trail = process_starter_spacing(0, newer_spacing, 0, 0)
6702               new_index = newer_index
6703               spacing = newer_spacing
6704       else
6705           spacing = current_trail.remainder
6706       end
6707       local result, remainder = check_trail_type(s, new_index, current_trail, spacing,
6708       if remainder == nil or omit_remainder then
6709           if result then
6710               return new_index
6711           end
6712           return false
6713       end
6714
6715       if is_blank and reset_rem then
6716           remainder = remove_remainder_if_blank(indent_table, remainder)
6717       end
6718       if result then
6719           return new_index, remainder
6720       end
6721       return false
6722   end
6723   return false
6724 end
6725

```

The following patterns check whitespace indentation at the start of a block.

```

6726 parsers.check_trail = Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(false), che
6727
6728 parsers.check_trail_no_rem = Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(fals
6729
6730 parsers.check_code_trail = Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(true)
6731
6732 parsers.check_trail_length_range = function(min, max)
6733     return Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(min) * Cc(max), check_tr
6734 end
6735
6736 parsers.check_trail_length = function(n)
6737     return parsers.check_trail_length_range(n, n)
6738 end
6739

```

The following patterns handle trail backup, to prevent a failing pattern to modify it before passing it to the next.

```
6740 parsers.freeze_trail = Cg(Cmt(Cb("indent_info") * Cc(true), trail_freezing), "indent_
6741
6742 parsers.unfreeze_trail = Cg(Cmt(Cb("indent_info") * Cc(false), trail_freezing), "inde
6743
```

The following patterns check indentation in continuation lines as defined by the container start.

```
6744 parsers.check_minimal_indent = Cmt(Cb("indent_info") * Cc(false), check_continuation_
6745
6746 parsers.check_optional_indent = Cmt(Cb("indent_info") * Cc(true), check_continuation_
6747
6748 parsers.check_minimal_blank_indent = Cmt(Cb("indent_info") * Cc(false) * Cc(true), ch
6749
```

The following patterns check indentation in continuation lines as defined by the container start. Additionally the subsequent trail is also directly checked.

```
6750
6751 parsers.check_minimal_indent_and_trail = Cmt( Cb("indent_info")
6752                                     * Cc(false) * Cc(false) * Cc("non-
        code") * Cc(true),
6753                                     check_continuation_indentation_and_trail)
6754
6755 parsers.check_minimal_indent_and_code_trail = Cmt( Cb("indent_info")
6756                                     * Cc(false) * Cc(false) * Cc("code")
6757                                     check_continuation_indentation_and_t
6758
6759 parsers.check_minimal_blank_indent_and_full_code_trail = Cmt( Cb("indent_info")
6760                                     * Cc(false) * Cc(true) *
        code") * Cc(true),
6761                                     check_continuation_indentation_and_trail)
6762
6763 parsers.check_minimal_indent_and_any_trail = Cmt( Cb("indent_info")
6764                                     * Cc(false) * Cc(false) * Cc("full-
        any") * Cc(true) * Cc(false),
6765                                     check_continuation_indentation_and_trail)
6766
6767 parsers.check_minimal_blank_indent_and_any_trail = Cmt( Cb("indent_info")
6768                                     * Cc(false) * Cc(true) * Cc("fu
        any") * Cc(true) * Cc(false),
6769                                     check_continuation_indentation_and_trail)
6770
6771 parsers.check_minimal_blank_indent_and_any_trail_no_rem = Cmt( Cb("indent_info")
6772                                     * Cc(false) * Cc(true) * Cc("fu
        any") * Cc(true) * Cc(true),
6773                                     check_continuation_indentation_and_trail)
```

```

6774
6775 parsers.check_optional_indent_and_any_trail = Cmt( Cb("indent_info")
6776                                     * Cc(true) * Cc(false) * Cc("full-
any") * Cc(true) * Cc(false),
6777                                     check_continuation_indentation_and_tr
6778
6779 parsers.check_optional_blank_indent_and_any_trail = Cmt( Cb("indent_info")
6780                                     * Cc(true) * Cc(true) * Cc("ful
any") * Cc(true) * Cc(false),
6781                                     check_continuation_indentation
6782

```

The following patterns specify behaviour around newlines.

```

6783
6784 parsers.spnlc_noexc = parsers.optionalspace
6785                                     * (parsers.newline * parsers.check_minimal_indent_and_any_trail)^
1
6786
6787 parsers.spnlc = parsers.optionalspace
6788                                     * (V("EndlineNoSub"))^-1
6789
6790 parsers.spnlc_sep = parsers.optionalspace * V("EndlineNoSub")
6791                                     + parsers.spacechar^1
6792
6793 parsers.only_blank = parsers.spacechar^0 * (parsers.newline + parsers.eof)
6794
6795 % \end{macrocode}
6796 % \begin{figure}
6797 % \hspace*{-0.1\textwidth}
6798 % \begin{minipage}{1.2\textwidth}
6799 % \centering
6800 % \begin{tikzpicture}[shorten >=1pt, line width=0.1mm, >={Stealth[length=2mm]}, node
6801 % \node[state, initial by diamond, accepting] (noop) {initial};
6802 % \node[state] (odd_backslash) [above right=of noop] {odd backslash};
6803 % \node[state] (even_backslash) [below right=of odd_backslash] {even backslash};
6804 % \node[state] (comment) [below=of noop] {comment};
6805 % \node[state] (leading_spaces) [below=of even_backslash, align=center] {leading tabs
6806 % \node[state] (blank_line) [below right=of comment] {blank line};
6807 % \path[->]
6808 % (noop) edge [in=150, out=180, loop] node [align=center, yshift=-0.75cm] {match [$^
6809 % edge [bend right=10] node [below right=-0.2cm] {match \textbackslash} (odd_b
6810 % edge [bend left=30] node [left, align=center] {match \%\\capture \textbacksl
6811 % (comment) edge [in=305, out=325, loop] node [xshift=-1.2cm] {match [$^\wedge\$\drsh$
6812 % edge [bend left=10] node {match $\drsh$} (leading_spaces)
6813 % (leading_spaces) edge [loop below] node {match [\textvisiblespace$\rightarrow$
6814 % edge [bend right=90] node [right] {match \textbackslash} (odd_back
6815 % edge [bend left=10] node {match \%} (comment)
6816 % edge [bend right=10] node {$\epsilon$} (blank_line)

```

```

6817 %             edge [bend left=10] node [align=center, right=0.3cm] {match [$~\we
6818 % (blank_line) edge [loop below] node {match [\textvisiblespace$\rightleftarrows$]} (
6819 %             edge [bend left=90] node [align=center, below=1.2cm] {match $\drsh$\
6820 % (odd_backslash) edge [bend right=10] node [align=center, xshift=-0.3cm, yshift=0.2cm]
6821 %             edge [bend right=10] node [align=center, above left=-
0.3cm, xshift=0.1cm] {match [$~\wedge$\textbackslash$\for \%, capture \textbackslash
6822 % (even_backslash) edge [bend left=10] node {$\epsilon$} (noop);
6823 % \end{tikzpicture}
6824 % \caption{A pushdown automaton that recognizes \TeX{} comments}
6825 % \label{fig:commented_line}
6826 % \end{minipage}
6827 % \end{figure}
6828 % \begin{markdown}
6829 %
6830 % The \luamdef{parsers.commented_line}~^1~ parser recognizes the regular
6831 % language of \TeX{} comments, see an equivalent finite automaton in Figure
6832 % <#fig:commented_line>.
6833 %
6834 % \end{markdown}
6835 % \begin{macrocode}
6836 parsers.comment_line_letter = parsers.linechar
6837                               + parsers.newline
6838                               - parsers.backslash
6839                               - parsers.percent
6840 parsers.comment_line         = Cg(Cc(""), "backslashes")
6841                               * ((#(parsers.comment_line_letter
6842                                   - parsers.newline)
6843                                   * Cb("backslashes")
6844                                   * Cs(parsers.comment_line_letter
6845                                       - parsers.newline)^1 -- initial
6846                                   * Cg(Cc(""), "backslashes"))
6847                               + #(parsers.backslash * parsers.backslash)
6848                               * Cg((parsers.backslash -- even backslash
6849                                   * parsers.backslash)^1, "backslashes")
6850                               + (parsers.backslash
6851                                   * (#parsers.percent
6852                                       * Cb("backslashes")
6853                                       / function(backslashes)
6854                                           return string.rep("\\", #backslashes / 2)
6855                                       end
6856                                   * C(parsers.percent)
6857                                   + #parsers.comment_line_letter
6858                                   * Cb("backslashes")
6859                                   * Cc("\\")
6860                                   * C(parsers.comment_line_letter))
6861                                   * Cg(Cc(""), "backslashes"))^0
6862                               * (#parsers.percent

```

```

6863 * Cb("backslashes")
6864 / function(backslashes)
6865   return string.rep("\\", #backslashes / 2)
6866 end
6867 * ((parsers.percent -- comment
6868   * parsers.line
6869   * #parsers.blankline) -- blank line
6870 / "\n"
6871 + parsers.percent -- comment
6872 * parsers.line
6873 * parsers.optionalspace) -- leading tabs and space
6874 + #(parsers.newline)
6875 * Cb("backslashes")
6876 * C(parsers.newline))
6877
6878 parsers.chunk = parsers.line * (parsers.optionallyindentedline
6879   - parsers.blankline)^0
6880
6881 parsers.attribute_key_char = parsers.alphanumeric + S("-_:.")
6882 parsers.attribute_raw_char = parsers.alphanumeric + S("-_")
6883 parsers.attribute_key = (parsers.attribute_key_char
6884   - parsers.dash - parsers.digit)
6885 * parsers.attribute_key_char^0
6886 parsers.attribute_value = ( (parsers.dquote / "\"")
6887   * (parsers.anyescaped - parsers.dquote)^0
6888   * (parsers.dquote / "\""))
6889 + ( (parsers.squote / "\"")
6890   * (parsers.anyescaped - parsers.squote)^0
6891   * (parsers.squote / "\""))
6892 + ( parsers.anyescaped - parsers.dquote - parsers.rbrace
6893   - parsers.space)^0
6894 parsers.attribute_identifier = parsers.attribute_key_char^1
6895 parsers.attribute_classname = parsers.letter
6896 * parsers.attribute_key_char^0
6897 parsers.attribute_raw = parsers.attribute_raw_char^1
6898
6899 parsers.attribute = (parsers.dash * Cc(".unnumbered"))
6900   + C( parsers.hash
6901     * parsers.attribute_identifier)
6902   + C( parsers.period
6903     * parsers.attribute_classname)
6904   + Cs( parsers.attribute_key
6905     * parsers.optionalspace * parsers.equal * parsers.optionalspace
6906     * parsers.attribute_value)
6907 parsers.attributes = parsers.lbrace
6908   * parsers.optionalspace
6909   * parsers.attribute

```

```

6910             * (parsers.spacechar^1
6911               * parsers.attribute)^0
6912             * parsers.optionalspace
6913             * parsers.rbrace
6914
6915
6916 parsers.raw_attribute = parsers.lbrace
6917                       * parsers.optionalspace
6918                       * parsers.equal
6919                       * C(parsers.attribute_raw)
6920                       * parsers.optionalspace
6921                       * parsers.rbrace
6922
6923 -- block followed by 0 or more optionally
6924 -- indented blocks with first line indented.
6925 parsers.indented_blocks = function(bl)
6926   return Cs( bl
6927             * (parsers.blankline^1 * parsers.indent * -parsers.blankline * bl)^0
6928             * (parsers.blankline^1 + parsers.eof) )
6929 end

```

### 3.1.4.3 Parsers Used for HTML Entities

```

6930 local function repeat_between(pattern, min, max)
6931   return -pattern^(max + 1) * pattern^min
6932 end
6933
6934 parsers.hexentity = parsers.ampersand * parsers.hash * C(S("Xx"))
6935                   * C(repeat_between(parsers.hexdigit, 1, 6)) * parsers.semicolon
6936 parsers.decentity = parsers.ampersand * parsers.hash
6937                   * C(repeat_between(parsers.digit, 1, 7)) * parsers.semicolon
6938 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
6939                   * parsers.semicolon
6940
6941 parsers.html_entities = parsers.hexentity / entities.hex_entity_with_x_char
6942                       + parsers.decentity / entities.dec_entity
6943                       + parsers.tagentity / entities.char_entity

```

### 3.1.4.4 Parsers Used for Markdown Lists

```

6944 parsers.bullet = function(bullet_char, interrupting)
6945   local allowed_end
6946   if interrupting then
6947     allowed_end = C(parsers.spacechar^1) * #parsers.linechar
6948   else
6949     allowed_end = C(parsers.spacechar^1) + #(parsers.newline + parsers.eof)
6950   end
6951   return parsers.check_trail

```

```

6952         * Ct(C(bullet_char) * Cc(""))
6953         * allowed_end
6954     end
6955
6956     local function tickbox(interior)
6957         return parsers.optionalspace * parsers.lbracket
6958             * interior * parsers.rbracket * parsers.spacechar^1
6959     end
6960
6961     parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
6962     parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
6963     parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
6964

```

### 3.1.4.5 Parsers Used for Markdown Code Spans

```

6965     parsers.openticks = Cg(parsers.backtick^1, "ticks")
6966
6967     local function captures_equal_length(_,i,a,b)
6968         return #a == #b and i
6969     end
6970
6971     parsers.closeticks = Cmt(C(parsers.backtick^1)
6972         * Cb("ticks"), captures_equal_length)
6973
6974     parsers.intickschar = (parsers.any - S("\n\r`"))
6975         + V("NoSoftLineBreakEndline")
6976         + (parsers.backtick^1 - parsers.closeticks)
6977
6978     local function process_inticks(s)
6979         s = s:gsub("\n", " ")
6980         s = s:gsub("^ (.*) $", "%1")
6981         return s
6982     end
6983
6984     parsers.inticks = parsers.openticks
6985         * C(parsers.space^0)
6986         * parsers.closeticks
6987         + parsers.openticks
6988         * Cs(Cs(parsers.intickschar^0) / process_inticks)
6989         * parsers.closeticks
6990

```

### 3.1.4.6 Parsers Used for HTML

```

6991     -- case-insensitive match (we assume s is lowercase). must be single byte encoding
6992     parsers.keyword_exact = function(s)
6993         local parser = P(0)

```

```

6994   for i=1,#s do
6995       local c = s:sub(i,i)
6996       local m = c .. upper(c)
6997       parser = parser * S(m)
6998   end
6999   return parser
7000 end
7001
7002 parsers.special_block_keyword =
7003     parsers.keyword_exact("pre") +
7004     parsers.keyword_exact("script") +
7005     parsers.keyword_exact("style") +
7006     parsers.keyword_exact("textarea")
7007
7008 parsers.block_keyword =
7009     parsers.keyword_exact("address") +
7010     parsers.keyword_exact("article") +
7011     parsers.keyword_exact("aside") +
7012     parsers.keyword_exact("base") +
7013     parsers.keyword_exact("basefont") +
7014     parsers.keyword_exact("blockquote") +
7015     parsers.keyword_exact("body") +
7016     parsers.keyword_exact("caption") +
7017     parsers.keyword_exact("center") +
7018     parsers.keyword_exact("col") +
7019     parsers.keyword_exact("colgroup") +
7020     parsers.keyword_exact("dd") +
7021     parsers.keyword_exact("details") +
7022     parsers.keyword_exact("dialog") +
7023     parsers.keyword_exact("dir") +
7024     parsers.keyword_exact("div") +
7025     parsers.keyword_exact("dl") +
7026     parsers.keyword_exact("dt") +
7027     parsers.keyword_exact("fieldset") +
7028     parsers.keyword_exact("figcaption") +
7029     parsers.keyword_exact("figure") +
7030     parsers.keyword_exact("footer") +
7031     parsers.keyword_exact("form") +
7032     parsers.keyword_exact("frame") +
7033     parsers.keyword_exact("frameset") +
7034     parsers.keyword_exact("h1") +
7035     parsers.keyword_exact("h2") +
7036     parsers.keyword_exact("h3") +
7037     parsers.keyword_exact("h4") +
7038     parsers.keyword_exact("h5") +
7039     parsers.keyword_exact("h6") +
7040     parsers.keyword_exact("head") +

```



```

7041     parsers.keyword_exact("header") +
7042     parsers.keyword_exact("hr") +
7043     parsers.keyword_exact("html") +
7044     parsers.keyword_exact("iframe") +
7045     parsers.keyword_exact("legend") +
7046     parsers.keyword_exact("li") +
7047     parsers.keyword_exact("link") +
7048     parsers.keyword_exact("main") +
7049     parsers.keyword_exact("menu") +
7050     parsers.keyword_exact("menuitem") +
7051     parsers.keyword_exact("nav") +
7052     parsers.keyword_exact("noframes") +
7053     parsers.keyword_exact("ol") +
7054     parsers.keyword_exact("optgroup") +
7055     parsers.keyword_exact("option") +
7056     parsers.keyword_exact("p") +
7057     parsers.keyword_exact("param") +
7058     parsers.keyword_exact("section") +
7059     parsers.keyword_exact("source") +
7060     parsers.keyword_exact("summary") +
7061     parsers.keyword_exact("table") +
7062     parsers.keyword_exact("tbody") +
7063     parsers.keyword_exact("td") +
7064     parsers.keyword_exact("tfoot") +
7065     parsers.keyword_exact("th") +
7066     parsers.keyword_exact("thead") +
7067     parsers.keyword_exact("title") +
7068     parsers.keyword_exact("tr") +
7069     parsers.keyword_exact("track") +
7070     parsers.keyword_exact("ul")
7071
7072 -- end conditions
7073 parsers.html_blankline_end_condition = parsers.linechar^0
7074                                     * ( parsers.newline
7075                                     * (parsers.check_minimal_blank_indent_and_any
7076                                     * #parsers.blankline
7077                                     + parsers.check_minimal_indent_and_any_trai
7078                                     * parsers.linechar^1)^0
7079                                     * (parsers.newline^-1 / "")
7080
7081 local function remove_trailing_blank_lines(s)
7082     return s:gsub("[\n\r]+%s*$", "")
7083 end
7084
7085 parsers.html_until_end = function(end_marker)
7086     return Cs(Cs((parsers.newline
7087                 * (parsers.check_minimal_blank_indent_and_any_trail

```

```

7088         * #parsers.blankline
7089         + parsers.check_minimal_indent_and_any_trail)
7090     + parsers.linechar - end_marker)^0
7091     * parsers.linechar^0 * parsers.newline~1)
7092     / remove_trailing_blank_lines)
7093 end
7094
7095 -- attributes
7096 parsers.html_attribute_spacing = parsers.optionalspace
7097                               * V("NoSoftLineBreakEndline")
7098                               * parsers.optionalspace
7099                               + parsers.spacechar^1
7100
7101 parsers.html_attribute_name = (parsers.letter + parsers.colon + parsers.underscore)
7102                               * (parsers.alphanumeric + parsers.colon + parsers.underscore
7103                               + parsers.period + parsers.dash)^0
7104
7105 parsers.html_attribute_value = parsers.squote
7106                               * (parsers.linechar - parsers.squote)^0
7107                               * parsers.squote
7108                               + parsers.dquote
7109                               * (parsers.linechar - parsers.dquote)^0
7110                               * parsers.dquote
7111                               + ( parsers.any - parsers.spacechar - parsers.newline
7112                               - parsers.dquote - parsers.squote - parsers.backtick
7113                               - parsers.equal - parsers.less - parsers.more)^1
7114
7115 parsers.html_inline_attribute_value = parsers.squote
7116                                     * (V("NoSoftLineBreakEndline")
7117                                     + parsers.any
7118                                     - parsers.blankline^2
7119                                     - parsers.squote)^0
7120                                     * parsers.squote
7121                                     + parsers.dquote
7122                                     * (V("NoSoftLineBreakEndline")
7123                                     + parsers.any
7124                                     - parsers.blankline^2
7125                                     - parsers.dquote)^0
7126                                     * parsers.dquote
7127                                     + (parsers.any - parsers.spacechar - parsers.newline
7128                                     - parsers.dquote - parsers.squote - parsers.backtick
7129                                     - parsers.equal - parsers.less - parsers.more)^1
7130
7131 parsers.html_attribute_value_specification = parsers.optionalspace
7132                                             * parsers.equal
7133                                             * parsers.optionalspace
7134                                             * parsers.html_attribute_value

```

```

7135
7136 parsers.html_spnl = parsers.optionalspace
7137         * (V("NoSoftLineBreakEndline") * parsers.optionalspace)^-
7138 1
7139 parsers.html_inline_attribute_value_specification = parsers.html_spnl
7140         * parsers.equal
7141         * parsers.html_spnl
7142         * parsers.html_inline_attribute_val
7143
7144 parsers.html_attribute = parsers.html_attribute_spacing
7145         * parsers.html_attribute_name
7146         * parsers.html_inline_attribute_value_specification^-
7147 1
7148 parsers.html_non_newline_attribute = parsers.spacechar^1
7149         * parsers.html_attribute_name
7150         * parsers.html_attribute_value_specification^-
7151 1
7152 parsers.nested_breaking_blank = parsers.newline
7153         * parsers.check_minimal_blank_indent
7154         * parsers.blankline
7155
7156 parsers.html_comment_start = P("<!--")
7157
7158 parsers.html_comment_end = P("-->")
7159
7160 parsers.html_comment = Cs( parsers.html_comment_start
7161         * parsers.html_until_end(parsers.html_comment_end))
7162
7163 parsers.html_inline_comment = (parsers.html_comment_start / "")
7164         * -P(">") * -P("-->")
7165         * Cs((V("NoSoftLineBreakEndline") + parsers.any - P("--
7166         "
7167         - parsers.nested_breaking_blank - parsers.html_commen
7168         * (parsers.html_comment_end / ""))
7169
7170 parsers.html_cdatasection_start = P("<![CDATA[")
7171
7172 parsers.html_cdatasection_end = P("]]>")
7173
7174 parsers.html_cdatasection = Cs( parsers.html_cdatasection_start
7175         * parsers.html_until_end(parsers.html_cdatasection_end))
7176
7177 parsers.html_inline_cdatasection = parsers.html_cdatasection_start
7178         * Cs(V("NoSoftLineBreakEndline") + parsers.any

```

```

7178             - parsers.nested_breaking_blank - parsers.html_
7179             * parsers.html_cdatasection_end
7180
7181 parsers.html_declaration_start = P("<!") * parsers.letter
7182
7183 parsers.html_declaration_end = P(">")
7184
7185 parsers.html_declaration = Cs( parsers.html_declaration_start
7186                               * parsers.html_until_end(parsers.html_declaration_end))
7187
7188 parsers.html_inline_declaration = parsers.html_declaration_start
7189                               * Cs(V("NoSoftLineBreakEndline") + parsers.any
7190                                   - parsers.nested_breaking_blank - parsers.html_de
7191                                   * parsers.html_declaration_end
7192
7193 parsers.html_instruction_start = P("<?")
7194
7195 parsers.html_instruction_end = P("?>")
7196
7197 parsers.html_instruction = Cs( parsers.html_instruction_start
7198                               * parsers.html_until_end(parsers.html_instruction_end))
7199
7200 parsers.html_inline_instruction = parsers.html_instruction_start
7201                               * Cs(V("NoSoftLineBreakEndline") + parsers.any
7202                                   - parsers.nested_breaking_blank - parsers.html_in
7203                                   * parsers.html_instruction_end
7204
7205 parsers.html_blankline = parsers.newline
7206                       * parsers.optionalspace
7207                       * parsers.newline
7208
7209 parsers.html_tag_start = parsers.less
7210
7211 parsers.html_tag_closing_start = parsers.less
7212                               * parsers.slash
7213
7214 parsers.html_tag_end = parsers.html_spnl
7215                       * parsers.more
7216
7217 parsers.html_empty_tag_end = parsers.html_spnl
7218                               * parsers.slash
7219                               * parsers.more
7220
7221 -- opening tags
7222 parsers.html_any_open_inline_tag = parsers.html_tag_start
7223                               * parsers.keyword
7224                               * parsers.html_attribute^0

```

```

7225             * parsers.html_tag_end
7226
7227 parsers.html_any_open_tag = parsers.html_tag_start
7228             * parsers.keyword
7229             * parsers.html_non_newline_attribute^0
7230             * parsers.html_tag_end
7231
7232 parsers.html_open_tag = parsers.html_tag_start
7233             * parsers.block_keyword
7234             * parsers.html_attribute^0
7235             * parsers.html_tag_end
7236
7237 parsers.html_open_special_tag = parsers.html_tag_start
7238             * parsers.special_block_keyword
7239             * parsers.html_attribute^0
7240             * parsers.html_tag_end
7241
7242 -- incomplete tags
7243 parsers.incomplete_tag_following = parsers.spacechar
7244             + parsers.more
7245             + parsers.slash * parsers.more
7246             + #(parsers.newline + parsers.eof)
7247
7248 parsers.incomplete_special_tag_following = parsers.spacechar
7249             + parsers.more
7250             + #(parsers.newline + parsers.eof)
7251
7252 parsers.html_incomplete_open_tag = parsers.html_tag_start
7253             * parsers.block_keyword
7254             * parsers.incomplete_tag_following
7255
7256 parsers.html_incomplete_open_special_tag = parsers.html_tag_start
7257             * parsers.special_block_keyword
7258             * parsers.incomplete_special_tag_following
7259
7260 parsers.html_incomplete_close_tag = parsers.html_tag_closing_start
7261             * parsers.block_keyword
7262             * parsers.incomplete_tag_following
7263
7264 parsers.html_incomplete_close_special_tag = parsers.html_tag_closing_start
7265             * parsers.special_block_keyword
7266             * parsers.incomplete_tag_following
7267
7268 -- closing tags
7269 parsers.html_close_tag = parsers.html_tag_closing_start
7270             * parsers.block_keyword
7271             * parsers.html_tag_end

```

```

7272
7273 parsers.html_any_close_tag = parsers.html_tag_closing_start
7274                             * parsers.keyword
7275                             * parsers.html_tag_end
7276
7277 parsers.html_close_special_tag = parsers.html_tag_closing_start
7278                             * parsers.special_block_keyword
7279                             * parsers.html_tag_end
7280
7281 -- empty tags
7282 parsers.html_any_empty_inline_tag = parsers.html_tag_start
7283                                 * parsers.keyword
7284                                 * parsers.html_attribute^0
7285                                 * parsers.html_empty_tag_end
7286
7287 parsers.html_any_empty_tag = parsers.html_tag_start
7288                             * parsers.keyword
7289                             * parsers.html_non_newline_attribute^0
7290                             * parsers.optionalspace
7291                             * parsers.slash
7292                             * parsers.more
7293
7294 parsers.html_empty_tag = parsers.html_tag_start
7295                         * parsers.block_keyword
7296                         * parsers.html_attribute^0
7297                         * parsers.html_empty_tag_end
7298
7299 parsers.html_empty_special_tag = parsers.html_tag_start
7300                             * parsers.special_block_keyword
7301                             * parsers.html_attribute^0
7302                             * parsers.html_empty_tag_end
7303
7304 parsers.html_incomplete_blocks = parsers.html_incomplete_open_tag
7305                               + parsers.html_incomplete_open_special_tag
7306                               + parsers.html_incomplete_close_tag
7307
7308 -- parse special html blocks
7309 parsers.html_blankline_ending_special_block_opening = (parsers.html_close_special_tag
7310                                                         + parsers.html_empty_special_tag
7311                                                         * #(parsers.optionalspace
7312                                                         * (parsers.newline + parsers.e
7313
7314 parsers.html_blankline_ending_special_block = parsers.html_blankline_ending_special_block_opening
7315                                             * parsers.html_blankline_end_condition
7316
7317 parsers.html_special_block_opening = parsers.html_incomplete_open_special_tag
7318                                   - parsers.html_empty_special_tag

```

```

7319
7320 parsers.html_closing_special_block = parsers.html_special_block_opening
7321                                     * parsers.html_until_end(parsers.html_close_speci
7322
7323 parsers.html_special_block = parsers.html_blankline_ending_special_block
7324                             + parsers.html_closing_special_block
7325
7326 -- parse html blocks
7327 parsers.html_block_opening = parsers.html_incomplete_open_tag
7328                             + parsers.html_incomplete_close_tag
7329
7330 parsers.html_block = parsers.html_block_opening
7331                     * parsers.html_blankline_end_condition
7332
7333 -- parse any html blocks
7334 parsers.html_any_block_opening = (parsers.html_any_open_tag
7335                                   + parsers.html_any_close_tag
7336                                   + parsers.html_any_empty_tag)
7337                                   * #(parsers.optionalspace * (parsers.newline + parser
7338
7339 parsers.html_any_block = parsers.html_any_block_opening
7340                         * parsers.html_blankline_end_condition
7341
7342 parsers.html_inline_comment_full = parsers.html_comment_start
7343                                  * -P(">") * -P("->")
7344                                  * Cs((V("NoSoftLineBreakEndline") + parsers.any - P
7345                                     ")
7346                                     - parsers.nested_breaking_blank - parsers.html_
7347                                     * parsers.html_comment_end
7348
7349 parsers.html_inline_tags = parsers.html_inline_comment_full
7350                          + parsers.html_any_empty_inline_tag
7351                          + parsers.html_inline_instruction
7352                          + parsers.html_inline_cdatasection
7353                          + parsers.html_inline_declaration
7354                          + parsers.html_any_open_inline_tag
7355                          + parsers.html_any_close_tag

```

### 3.1.4.7 Parsers Used for Markdown Tags and Links

```

7356 parsers.urlchar = parsers.anyescaped
7357                 - parsers.newline
7358                 - parsers.more
7359
7360 parsers.auto_link_scheme_part = parsers.alphanumeric
7361                               + parsers.plus

```

```

7362             + parsers.period
7363             + parsers.dash
7364
7365 parsers.auto_link_scheme = parsers.letter
7366             * parsers.auto_link_scheme_part
7367             * parsers.auto_link_scheme_part^~30
7368
7369 parsers.absolute_uri = parsers.auto_link_scheme * parsers.colon
7370             * (parsers.any - parsers.spacing - parsers.less - parsers.more)
7371
7372 parsers.printable_characters = S("!.#$%&'*/=?^_`{|}~-")
7373
7374 parsers.email_address_local_part_char = parsers.alphanumeric
7375             + parsers.printable_characters
7376
7377 parsers.email_address_local_part = parsers.email_address_local_part_char^1
7378
7379 parsers.email_address_dns_label = parsers.alphanumeric
7380             * (parsers.alphanumeric + parsers.dash)^-
7381             62
7382             * B(parsers.alphanumeric)
7383
7384 parsers.email_address_domain = parsers.email_address_dns_label
7385             * (parsers.period * parsers.email_address_dns_label)^0
7386
7387 parsers.email_address = parsers.email_address_local_part
7388             * parsers.at
7389             * parsers.email_address_domain
7390
7391 parsers.auto_link_url = parsers.less
7392             * C(parsers.absolute_uri)
7393             * parsers.more
7394
7395 parsers.auto_link_email = parsers.less
7396             * C(parsers.email_address)
7397             * parsers.more
7398
7399 parsers.auto_link_relative_reference = parsers.less
7400             * C(parsers.urlchar^1)
7401             * parsers.more
7402
7403 parsers.autolink = parsers.auto_link_url
7404             + parsers.auto_link_email
7405
7406 -- content in balanced brackets, parentheses, or quotes:
7407 parsers.bracketed = P{ parsers.lbracket
7408             * (( parsers.backslash / '"' * parsers.rbracket

```



```

7408         + parsers.any - (parsers.lbracket
7409                             + parsers.rbracket
7410                             + parsers.blankline^2)
7411         ) + V(1))^0
7412     * parsers.rbracket }
7413
7414 parsers.inparens = P{ parsers.lparent
7415     * ((parsers.anyescaped - (parsers.lparent
7416                             + parsers.rparent
7417                             + parsers.blankline^2)
7418         ) + V(1))^0
7419     * parsers.rparent }
7420
7421 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
7422     * ((parsers.anyescaped - (parsers.squote
7423                             + parsers.blankline^2)
7424         ) + V(1))^0
7425     * parsers.squote }
7426
7427 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
7428     * ((parsers.anyescaped - (parsers.dquote
7429                             + parsers.blankline^2)
7430         ) + V(1))^0
7431     * parsers.dquote }
7432
7433 parsers.link_text = parsers.lbracket
7434     * Cs((parsers.alphanumeric^1
7435         + parsers.bracketed
7436         + parsers.inticks
7437         + parsers.autolink
7438         + V("InlineHtml")
7439         + ( parsers.backslash * parsers.backslash)
7440         + ( parsers.backslash * (parsers.lbracket + parsers.rbracket)
7441         + V("NoSoftLineBreakSpace")
7442         + V("NoSoftLineBreakEndline")
7443         + (parsers.any
7444             - (parsers.newline + parsers.lbracket + parsers.rbracket)
7445         * parsers.rbracket
7446
7447 parsers.link_label = parsers.lbracket
7448     * -(parsers.sp * parsers.rbracket)
7449     * #((parsers.any - parsers.rbracket)^-999 * parsers.rbracket)
7450     * Cs((parsers.alphanumeric^1
7451         + parsers.inticks
7452         + parsers.autolink
7453         + V("InlineHtml")
7454         + ( parsers.backslash * parsers.backslash)

```

```

7455         + ( parsers.backslash * (parsers.lbracket + parsers.rbracket)
7456         + V("NoSoftLineBreakSpace")
7457         + V("NoSoftLineBreakEndline")
7458         + (parsers.any
7459           - (parsers.newline + parsers.lbracket + parsers.rbracket
7460             * parsers.rbracket
7461
7462 parsers.inparens_url = P{ parsers.lparent
7463         * ((parsers.anyescaped - (parsers.lparent
7464           + parsers.rparent
7465           + parsers.spacing)
7466           ) + V(1))^0
7467         * parsers.rparent }
7468
7469 -- url for markdown links, allowing nested brackets:
7470 parsers.url = parsers.less * Cs((parsers.anyescaped
7471   - parsers.newline
7472   - parsers.less
7473   - parsers.more)^0)
7474   * parsers.more
7475 + -parsers.less
7476 * Cs((parsers.inparens_url + (parsers.anyescaped
7477   - parsers.spacing
7478   - parsers.lparent
7479   - parsers.rparent))^1)
7480
7481 -- quoted text:
7482 parsers.title_s = parsers.squote
7483   * Cs((parsers.html_entities
7484     + V("NoSoftLineBreakSpace")
7485     + V("NoSoftLineBreakEndline")
7486     + (parsers.anyescaped - parsers.newline - parsers.squote - p
7487   * parsers.squote
7488
7489 parsers.title_d = parsers.dquote
7490   * Cs((parsers.html_entities
7491     + V("NoSoftLineBreakSpace")
7492     + V("NoSoftLineBreakEndline")
7493     + (parsers.anyescaped - parsers.newline - parsers.dquote - p
7494   * parsers.dquote
7495
7496 parsers.title_p = parsers.lparent
7497   * Cs((parsers.html_entities
7498     + V("NoSoftLineBreakSpace")
7499     + V("NoSoftLineBreakEndline")
7500     + (parsers.anyescaped - parsers.newline - parsers.lparent -
7501     - parsers.blankline^2))^0)

```

```

7502             * parsers.rparent
7503
7504 parsers.title      = parsers.title_d + parsers.title_s + parsers.title_p
7505
7506 parsers.optionaltitle
7507             = parsers.spnlc * parsers.title * parsers.spacechar^0
7508             + Cc("")
7509

```

### 3.1.4.8 Helpers for Links and Link Reference Definitions

```

7510 -- parse a reference definition: [foo]: /bar "title"
7511 parsers.define_reference_parser = (parsers.check_trail / "") * parsers.link_label * p
7512             * parsers.spnlc * parsers.url
7513             * ( parsers.spnlc_sep * parsers.title * parsers.only_
7514             + Cc("") * parsers.only_blank)

```

### 3.1.4.9 Inline Elements

```

7515 parsers.Inline      = V("Inline")
7516
7517 -- parse many p between starter and ender
7518 parsers.between = function(p, starter, ender)
7519     local ender2 = B(parsers.nonspacechar) * ender
7520     return (starter * #parsers.nonspacechar * Ct(p * (p - ender2)^0) * ender2)
7521 end
7522

```

### 3.1.4.10 Block Elements

```

7523 parsers.lineof = function(c)
7524     return (parsers.check_trail_no_rem * (P(c) * parsers.optionalspace)^3
7525             * (parsers.newline + parsers.eof))
7526 end
7527
7528 parsers.thematic_break_lines = parsers.lineof(parsers.asterisk)
7529             + parsers.lineof(parsers.dash)
7530             + parsers.lineof(parsers.underscore)

```

### 3.1.4.11 Headings

```

7531 -- parse Atx heading start and return level
7532 parsers.heading_start = #parsers.hash * C(parsers.hash^6)
7533             * -parsers.hash / length
7534
7535 -- parse setext header ending and return level
7536 parsers.heading_level = parsers.nonindentSPACE * parsers.equal^1 * parsers.optionalsp
7537             + parsers.nonindentSPACE * parsers.dash^1 * parsers.optionalspa
7538

```

```

7539 local function strip_atx_end(s)
7540     return s:gsub("%s+##%s*\n$", "")
7541 end
7542
7543 parsers.atx_heading = parsers.check_trail_no_rem
7544                     * Cg(parsers.heading_start, "level")
7545                     * (C( parsers.optionalspace
7546                         * parsers.hash^0
7547                         * parsers.optionalspace
7548                         * parsers.newline)
7549                       + parsers.spacechar^1
7550                       * C(parsers.line))

```

### 3.1.5 Markdown Reader

This section documents the `reader` object, which implements the routines for parsing the markdown input. The object corresponds to the markdown reader object that was located in the `lunamark/reader/markdown.lua` file in the Lunamark Lua module.

The `reader.new` method creates and returns a new TeX reader object associated with the Lua interface options (see Section 2.1.3) `options` and with a writer object `writer`. When `options` are unspecified, it is assumed that an empty table was passed to the method.

The objects produced by the `reader.new` method expose instance methods and variables of their own. As a convention, I will refer to these *member*s as `reader->member`.

```

7551 M.reader = {}
7552 function M.reader.new(writer, options)
7553     local self = {}

```

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

```

7554     self.writer = writer
7555     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.

```

7556     self.parsers = {}
7557     (function(parsers)
7558         setmetatable(self.parsers, {
7559             __index = function (_, key)
7560                 return parsers[key]
7561             end
7562         })
7563     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.

```
7564   local parsers = self.parsers
```

### 3.1.5.1 Top-Level Helper Functions

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

```
7565   function self.normalize_tag(tag)
7566     tag = util.ropetostring(tag)
7567     tag = tag:gsub("[\n\r\t]+", " ")
7568     tag = tag:gsub("^ ", ""):gsub(" $", "")
7569     tag = uni_algos.case.casefold(tag, true, false)
7570     return tag
7571   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.

```
7572   local function iterlines(s, f)
7573     local rope = lpeg.match(Ct((parsers.line / f)^1), s)
7574     return util.ropetostring(rope)
7575   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.

```
7576   if options.preserveTabs then
7577     self.expandtabs = function(s) return s end
7578   else
7579     self.expandtabs = function(s)
7580       if s:find("\t") then
7581         return iterlines(s, util.expand_tabs_in_line)
7582       else
7583         return s
7584       end
7585     end
7586   end
```

### 3.1.5.2 High-Level Parser Functions

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

```
7587   self.parser_functions = {}
```

```

7588     self.create_parser = function(name, grammar, toplevel)
7589         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.

```

7590         if toplevel and options.stripIndent then
7591             local min_prefix_length, min_prefix = nil, ''
7592             str = iterlines(str, function(line)
7593                 if lpeg.match(parsers.nonemptyline, line) == nil then
7594                     return line
7595                 end
7596                 line = util.expand_tabs_in_line(line)
7597                 local prefix = lpeg.match(C(parsers.optionalspace), line)
7598                 local prefix_length = #prefix
7599                 local is_shorter = min_prefix_length == nil
7600                 is_shorter = is_shorter or prefix_length < min_prefix_length
7601                 if is_shorter then
7602                     min_prefix_length, min_prefix = prefix_length, prefix
7603                 end
7604                 return line
7605             end)
7606             str = str:gsub('^' .. min_prefix, '')
7607         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.

```

7608         if toplevel and (options.texComments or options.hybrid) then
7609             str = lpeg.match(Ct(parsers.commented_line^1), str)
7610             str = util.rope_to_string(str)
7611         end
7612         local res = lpeg.match(grammar(), str)
7613         if res == nil then
7614             error(format("%s failed on:\n%s", name, str:sub(1,20)))
7615         else
7616             return res
7617         end
7618     end
7619 end
7620
7621 self.create_parser("parse_blocks",
7622     function()
7623         return parsers.blocks
7624     end, true)
7625

```

```

7626 self.create_parser("parse_blocks_nested",
7627                     function()
7628                         return parsers.blocks_nested
7629                     end, false)
7630
7631 self.create_parser("parse_inlines",
7632                   function()
7633                       return parsers.inlines
7634                   end, false)
7635
7636 self.create_parser("parse_inlines_no_inline_note",
7637                   function()
7638                       return parsers.inlines_no_inline_note
7639                   end, false)
7640
7641 self.create_parser("parse_inlines_no_html",
7642                   function()
7643                       return parsers.inlines_no_html
7644                   end, false)
7645
7646 self.create_parser("parse_inlines_nbsp",
7647                   function()
7648                       return parsers.inlines_nbsp
7649                   end, false)
7650 self.create_parser("parse_inlines_no_link_or_emphasis",
7651                   function()
7652                       return parsers.inlines_no_link_or_emphasis
7653                   end, false)

```

### 3.1.5.3 Parsers Used for Indentation (local)

The following patterns represent basic building blocks of indented content.

```

7654 parsers.minimally_indented_blankline = parsers.check_minimal_indent * (parsers.blankline)
7655
7656 parsers.minimally_indented_block = parsers.check_minimal_indent * V("Block")
7657
7658 parsers.minimally_indented_block_or_paragraph = parsers.check_minimal_indent * V("Block|Paragraph")
7659
7660 parsers.minimally_indented_paragraph = parsers.check_minimal_indent * V("Paragraph")
7661
7662 parsers.minimally_indented_plain = parsers.check_minimal_indent * V("Plain")
7663
7664 parsers.minimally_indented_par_or_plain = parsers.minimally_indented_paragraph
7665                                         + parsers.minimally_indented_plain
7666
7667 parsers.minimally_indented_par_or_plain_no_blank = parsers.minimally_indented_par_or_plain
7668                                                    - parsers.minimally_indented_blankline

```

```

7669
7670 parsers.minimally_indented_ref = parsers.check_minimal_indent * V("Reference")
7671
7672 parsers.minimally_indented_blank = parsers.check_minimal_indent * V("Blank")
7673
7674 parsers.conditionally_indented_blankline = parsers.check_minimal_blank_indent * (pa
7675
7676 parsers.minimally_indented_ref_or_block = parsers.minimally_indented_ref
7677                                     + parsers.minimally_indented_block
7678                                     - parsers.minimally_indented_blankline
7679
7680 parsers.minimally_indented_ref_or_block_or_par = parsers.minimally_indented_ref
7681                                               + parsers.minimally_indented_block
7682                                               - parsers.minimally_indented_blankl
7683

```

The following pattern parses the properly indented content that follows the initial container start.

```

7684
7685 parsers.separator_loop = function(separated_block, paragraph, block_separator, para
7686     return separated_block
7687         + block_separator
7688         * paragraph
7689         * separated_block
7690     + paragraph_separator
7691     * paragraph
7692 end
7693
7694 parsers.create_loop_body_pair = function(separated_block, paragraph, block_separato
7695     return {
7696         block = parsers.separator_loop(separated_block, paragraph, block_separator, blo
7697         par = parsers.separator_loop(separated_block, paragraph, block_separator, para
7698     }
7699 end
7700
7701 parsers.block_sep_group = function(blank)
7702     return blank^0 * parsers.eof
7703         + ( blank^2 / writer.paragraphsep
7704         + blank^0 / writer.interblocksep
7705         )
7706 end
7707
7708 parsers.par_sep_group = function(blank)
7709     return blank^0 * parsers.eof
7710         + blank^0 / writer.paragraphsep
7711 end
7712

```



```

7713 parsers.sep_group_no_output = function(blank)
7714     return blank^0 * parsers.eof
7715         + blank^0
7716 end
7717
7718 parsers.content_blank = parsers.minimally_indented_blankline
7719
7720 parsers.ref_or_block_separated = parsers.sep_group_no_output(parsers.content_blank
7721     * ( parsers.minimally_indented_ref
7722         - parsers.content_blank)
7723     + parsers.block_sep_group(parsers.content_blank)
7724     * ( parsers.minimally_indented_block
7725         - parsers.content_blank)
7726
7727 parsers.loop_body_pair =
7728     parsers.create_loop_body_pair(parsers.ref_or_block_separated,
7729     parsers.minimally_indented_par_or_plain_no_blank,
7730     parsers.block_sep_group(parsers.content_blank),
7731     parsers.par_sep_group(parsers.content_blank))
7732
7733 parsers.content_loop = ( V("Block")
7734     * parsers.loop_body_pair.block^0
7735     + (V("Paragraph") + V("Plain"))
7736     * parsers.ref_or_block_separated
7737     * parsers.loop_body_pair.block^0
7738     + (V("Paragraph") + V("Plain"))
7739     * parsers.loop_body_pair.par^0)
7740     * parsers.content_blank^0
7741
7742 parsers.indented_content = function()
7743     return Ct( (V("Reference") + (parsers.blankline / ""))
7744         * parsers.content_blank^0
7745         * parsers.check_minimal_indent
7746         * parsers.content_loop
7747         + (V("Reference") + (parsers.blankline / ""))
7748         * parsers.content_blank^0
7749         + parsers.content_loop)
7750 end
7751
7752 parsers.add_indent = function(pattern, name, breakable)
7753     return Cg(Cmt( Cb("indent_info")
7754         * Ct(pattern)
7755         * (#parsers.linechar * Cc(false) + Cc(true)) -- check if starter is
7756         * Cc(name)
7757         * Cc(breakable),
7758         process_starter_indent), "indent_info")
7759 end

```

7760

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

```
7761 if options.hashEnumerators then
7762   parsers.dig = parsers.digit + parsers.hash
7763 else
7764   parsers.dig = parsers.digit
7765 end
7766
7767 parsers.enumerator = function(delimiter_type, interrupting)
7768   local delimiter_range
7769   local allowed_end
7770   if interrupting then
7771     delimiter_range = P("1")
7772     allowed_end = C(parsers.spacechar~1) * #parsers.linechar
7773   else
7774     delimiter_range = parsers.dig * parsers.dig~-8
7775     allowed_end = C(parsers.spacechar~1) + #(parsers.newline + parsers.eof)
7776   end
7777
7778   return parsers.check_trail
7779     * Ct(C(delimiter_range) * C(delimiter_type))
7780     * allowed_end
7781 end
7782
7783 parsers.starter = parsers.bullet(parsers.dash)
7784               + parsers.bullet(parsers.asterisk)
7785               + parsers.bullet(parsers.plus)
7786               + parsers.enumerator(parsers.period)
7787               + parsers.enumerator(parsers.rparent)
7788
```

#### 3.1.5.5 Parsers Used for Blockquotes (local)

```
7789 parsers.blockquote_start = parsers.check_trail * C(parsers.more) * C(parsers.space)
7790
7791 parsers.blockquote_body = parsers.add_indent(parsers.blockquote_start, "bq", true)
7792                       * parsers.indented_content()
7793                       * remove_indent("bq")
7794
7795 if not options.breakableBlockquotes then
7796   parsers.blockquote_body = parsers.add_indent(parsers.blockquote_start, "bq", false)
7797                       * parsers.indented_content()
7798                       * remove_indent("bq")
7799 end
```

#### 3.1.5.6 Helpers for Emphasis and Strong Emphasis (local)

Parse the content of a table `content_part` with links, images and emphasis disabled.

```
7800 local function parse_content_part(content_part)
7801   local rope = util.rope_to_string(content_part)
7802   local parsed = self.parser_functions.parse_inlines_no_link_or_emphasis(rope)
7803   parsed.indent_info = nil
7804   return parsed
7805 end
7806
```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
7807 local function collect_emphasis_content(t, opening_index, closing_index)
7808   local content = {}
7809
7810   local content_part = {}
7811   for i = opening_index, closing_index do
7812     local value = t[i]
7813
7814     if value.rendered ~= nil then
7815       content[#content + 1] = parse_content_part(content_part)
7816       content_part = {}
7817       content[#content + 1] = value.rendered
7818       value.rendered = nil
7819     else
7820       if value.type == "delimiter" and value.element == "emphasis" then
7821         if value.is_active then
7822           content_part[#content_part + 1] = string.rep(value.character, value.current)
7823         end
7824       else
7825         content_part[#content_part + 1] = value.content
7826       end
7827       value.content = ''
7828       value.is_active = false
7829     end
7830   end
7831
7832   if next(content_part) ~= nil then
7833     content[#content + 1] = parse_content_part(content_part)
7834   end
7835
7836   return content
7837 end
7838
```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as emphasis.

```

7839 local function fill_emph(t, opening_index, closing_index)
7840     local content = collect_emphasis_content(t, opening_index + 1, closing_index - 1)
7841     t[opening_index + 1].is_active = true
7842     t[opening_index + 1].rendered = writer.emphasis(content)
7843 end
7844

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as strong emphasis.

```

7845 local function fill_strong(t, opening_index, closing_index)
7846     local content = collect_emphasis_content(t, opening_index + 1, closing_index - 1)
7847     t[opening_index + 1].is_active = true
7848     t[opening_index + 1].rendered = writer.strong(content)
7849 end
7850

```

Check whether the opening delimiter `opening_delimiter` and closing delimiter `closing_delimiter` break rule three together.

```

7851 local function breaks_three_rule(opening_delimiter, closing_delimiter)
7852     return (opening_delimiter.is_closing or closing_delimiter.is_opening) and
7853         ((opening_delimiter.original_count + closing_delimiter.original_count) % 3 == 0)
7854         (opening_delimiter.original_count % 3 ~= 0 or closing_delimiter.original_count % 3 ~= 0)
7855 end
7856

```

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`.

```

7857 local function find_emphasis_opener(t, bottom_index, latest_index, character, closing_delimiter)
7858     for i = latest_index, bottom_index, -1 do
7859         local value = t[i]
7860         if value.is_active and
7861             value.is_opening and
7862             value.type == "delimiter" and
7863             value.element == "emphasis" and
7864             (value.character == character) and
7865             (value.current_count > 0) then
7866             if not breaks_three_rule(value, closing_delimiter) then
7867                 return i
7868             end
7869         end
7870     end
7871 end
7872

```

Iterate over the delimiters in the delimiter table `t`, producing emphasis or strong emphasis macros.

```

7873 local function process_emphasis(t, opening_index, closing_index)

```

```

7874     for i = opening_index, closing_index do
7875         local value = t[i]
7876         if value.type == "delimiter" and value.element == "emphasis" then
7877             local delimiter_length = string.len(value.content)
7878             value.character = string.sub(value.content, 1, 1)
7879             value.current_count = delimiter_length
7880             value.original_count = delimiter_length
7881         end
7882     end
7883
7884     local openers_bottom = {
7885         ['*'] = {
7886             [true] = {opening_index, opening_index, opening_index},
7887             [false] = {opening_index, opening_index, opening_index}
7888         },
7889         ['_'] = {
7890             [true] = {opening_index, opening_index, opening_index},
7891             [false] = {opening_index, opening_index, opening_index}
7892         }
7893     }
7894
7895     local current_position = opening_index
7896     local max_position = closing_index
7897
7898     while current_position <= max_position do
7899         local value = t[current_position]
7900
7901         if value.type ~= "delimiter" or
7902            value.element ~= "emphasis" or
7903            not value.is_active or
7904            not value.is_closing or
7905            (value.current_count <= 0) then
7906             current_position = current_position + 1
7907             goto continue
7908         end
7909
7910         local character = value.character
7911         local is_opening = value.is_opening
7912         local closing_length_modulo_three = value.original_count % 3
7913
7914         local current_openers_bottom = openers_bottom[character][is_opening][closing_length_modulo_three]
7915
7916         local opener_position = find_emphasis_opener(t, current_openers_bottom, current_position)
7917
7918         if (opener_position == nil) then
7919             openers_bottom[character][is_opening][closing_length_modulo_three + 1] = current_position
7920             current_position = current_position + 1

```

```

7921         goto continue
7922     end
7923
7924     local opening_delimiter = t[opener_position]
7925
7926     local current_opening_count = opening_delimiter.current_count
7927     local current_closing_count = t[current_position].current_count
7928
7929     if (current_opening_count >= 2) and (current_closing_count >= 2) then
7930         opening_delimiter.current_count = current_opening_count - 2
7931         t[current_position].current_count = current_closing_count - 2
7932         fill_strong(t, opener_position, current_position)
7933     else
7934         opening_delimiter.current_count = current_opening_count - 1
7935         t[current_position].current_count = current_closing_count - 1
7936         fill_emph(t, opener_position, current_position)
7937     end
7938
7939     ::continue::
7940 end
7941 end
7942
7943 local cont = lpeg.R("\128\191") -- continuation byte
7944

```

Match a UTF-8 character of byte length `n`.

```

7945 local function utf8_by_byte_count(n)
7946     if (n == 1) then
7947         return lpeg.R("\0\127")
7948     end
7949     if (n == 2) then
7950         return lpeg.R("\194\223") * cont
7951     end
7952     if (n == 3) then
7953         return lpeg.R("\224\239") * cont * cont
7954     end
7955     if (n == 4) then
7956         return lpeg.R("\240\244") * cont * cont * cont
7957     end
7958 end
7959

```

Check if there is a character of a type `chartype` between the start position `start_pos` and end position `end_pos` in a string `s` relative to current index `i`.

```

7960 local function check_unicode_type(s, i, start_pos, end_pos, chartype)
7961     local c
7962     local char_length
7963     for pos = start_pos, end_pos, 1 do

```

```

7964     if (start_pos < 0) then
7965         char_length = -pos
7966     else
7967         char_length = pos + 1
7968     end
7969     c = lpeg.match({ C(utf8_by_byte_count(char_length)) },s,i+pos)
7970     if (c ~= nil) and (unicode.utf8.match(c, chartype)) then
7971         return i
7972     end
7973 end
7974 end
7975
7976 local function check_preceding_unicode_punctuation(s, i)
7977     return check_unicode_type(s, i, -4, -1, "%p")
7978 end
7979
7980 local function check_preceding_unicode_whitespace(s, i)
7981     return check_unicode_type(s, i, -4, -1, "%s")
7982 end
7983
7984 local function check_following_unicode_punctuation(s, i)
7985     return check_unicode_type(s, i, 0, 3, "%p")
7986 end
7987
7988 local function check_following_unicode_whitespace(s, i)
7989     return check_unicode_type(s, i, 0, 3, "%s")
7990 end
7991
7992 parsers.unicode_preceding_punctuation = B(parsers.escapable)
7993                                     + Cmt(parsers.succeed, check_preceding_unicode
7994
7995 parsers.unicode_preceding_whitespace = Cmt(parsers.succeed, check_preceding_unicode
7996
7997 parsers.unicode_following_punctuation = #parsers.escapable
7998                                     + Cmt(parsers.succeed, check_following_unicon
7999
8000 parsers.unicode_following_whitespace = Cmt(parsers.succeed, check_following_unicon
8001
8002 parsers.delimiter_run = function(character)
8003     return (B(parsers.backslash * character) + ~B(character))
8004           * character~1
8005           * ~#character
8006 end
8007
8008 parsers.left_flanking_delimiter_run = function(character)
8009     return (B( parsers.any)
8010           * (parsers.unicode_preceding_punctuation + parsers.unicode_preceding_wh

```

```

8011         + -B(parsers.any))
8012     * parsers.delimiter_run(character)
8013     * parsers.unicode_following_punctuation
8014     + parsers.delimiter_run(character)
8015     * -(parsers.unicode_following_punctuation + parsers.unicode_following_whites
8016         + parsers.eof)
8017 end
8018
8019 parsers.right_flanking_delimiter_run = function(character)
8020     return parsers.unicode_preceding_punctuation
8021     * parsers.delimiter_run(character)
8022     * (parsers.unicode_following_punctuation + parsers.unicode_following_whites
8023     + parsers.eof)
8024     + (B(parsers.any)
8025     * -(parsers.unicode_preceding_punctuation + parsers.unicode_preceding_whites
8026     * parsers.delimiter_run(character)
8027 end
8028
8029 if options.underscores then
8030     parsers.emph_start = parsers.left_flanking_delimiter_run(parsers.asterisk)
8031     + (-#parsers.right_flanking_delimiter_run(parsers.underscore)
8032     + (parsers.unicode_preceding_punctuation
8033     * #parsers.right_flanking_delimiter_run(parsers.underscore)
8034     * parsers.left_flanking_delimiter_run(parsers.underscore)
8035
8036     parsers.emph_end = parsers.right_flanking_delimiter_run(parsers.asterisk)
8037     + (-#parsers.left_flanking_delimiter_run(parsers.underscore)
8038     + #(parsers.left_flanking_delimiter_run(parsers.underscore)
8039     * parsers.unicode_following_punctuation))
8040     * parsers.right_flanking_delimiter_run(parsers.underscore)
8041 else
8042     parsers.emph_start = parsers.left_flanking_delimiter_run(parsers.asterisk)
8043
8044     parsers.emph_end = parsers.right_flanking_delimiter_run(parsers.asterisk)
8045 end
8046
8047 parsers.emph_capturing_open_and_close = #parsers.emph_start * #parsers.emph_end
8048     * Ct( Cg(Cc("delimiter"), "type")
8049     * Cg(Cc("emphasis"), "element")
8050     * Cg(C(parsers.emph_start), "content")
8051     * Cg(Cc(true), "is_opening")
8052     * Cg(Cc(true), "is_closing"))
8053
8054 parsers.emph_capturing_open = Ct( Cg(Cc("delimiter"), "type")
8055     * Cg(Cc("emphasis"), "element")
8056     * Cg(C(parsers.emph_start), "content")
8057     * Cg(Cc(true), "is_opening"))

```



```

8058             * Cg(Cc(false), "is_closing"))
8059
8060     parsers.emph_capturing_close = Ct( Cg(Cc("delimiter"), "type")
8061             * Cg(Cc("emphasis"), "element")
8062             * Cg(C(parsers.emph_end), "content")
8063             * Cg(Cc(false), "is_opening")
8064             * Cg(Cc(true), "is_closing"))
8065
8066     parsers.emph_open_or_close = parsers.emph_capturing_open_and_close
8067             + parsers.emph_capturing_open
8068             + parsers.emph_capturing_close
8069
8070     parsers.emph_open = parsers.emph_capturing_open_and_close
8071             + parsers.emph_capturing_open
8072
8073     parsers.emph_close = parsers.emph_capturing_open_and_close
8074             + parsers.emph_capturing_close
8075

```

### 3.1.5.7 Helpers for Links and Link Reference Definitions (local)

```

8076     -- List of references defined in the document
8077     local references
8078

```

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.

```

8079     function self.register_link(_, tag, url, title,
8080             attributes)
8081         local normalized_tag = self.normalize_tag(tag)
8082         if references[normalized_tag] == nil then
8083             references[normalized_tag] = {
8084                 url = url,
8085                 title = title,
8086                 attributes = attributes
8087             }
8088         end
8089         return ""
8090     end
8091

```

The `reader->lookup_reference` method looks up a reference with link label `tag`.

```

8092     function self.lookup_reference(tag)
8093         return references[self.normalize_tag(tag)]
8094     end
8095
8096     parsers.title_s_direct_ref = parsers.squote

```

```

8097         * Cs((parsers.html_entities
8098             + (parsers.anyescaped - parsers.squote - parsers.bl
8099         * parsers.squote
8100
8101     parsers.title_d_direct_ref = parsers.dquote
8102         * Cs((parsers.html_entities
8103             + (parsers.anyescaped - parsers.dquote - parsers.bl
8104         * parsers.dquote
8105
8106     parsers.title_p_direct_ref = parsers.lparent
8107         * Cs((parsers.html_entities
8108             + (parsers.anyescaped - parsers.lparent - parsers.r
8109         * parsers.rparent
8110
8111     parsers.title_direct_ref = parsers.title_s_direct_ref
8112         + parsers.title_d_direct_ref
8113         + parsers.title_p_direct_ref
8114
8115     parsers.inline_direct_ref_inside = parsers.lparent * parsers.spnl
8116         * Cg(parsers.url + Cc(""), "url")
8117         * parsers.spnl
8118         * Cg(parsers.title_direct_ref + Cc(""), "title")
8119         * parsers.spnl * parsers.rparent
8120
8121     parsers.inline_direct_ref = parsers.lparent * parsers.spnlc
8122         * Cg(parsers.url + Cc(""), "url")
8123         * parsers.spnlc
8124         * Cg(parsers.title + Cc(""), "title")
8125         * parsers.spnlc * parsers.rparent
8126
8127     parsers.empty_link = parsers.lbracket
8128         * parsers.rbracket
8129
8130     parsers.inline_link = parsers.link_text
8131         * parsers.inline_direct_ref
8132
8133     parsers.full_link = parsers.link_text
8134         * parsers.link_label
8135
8136     parsers.shortcut_link = parsers.link_label
8137         * -(parsers.empty_link + parsers.link_label)
8138
8139     parsers.collapsed_link = parsers.link_label
8140         * parsers.empty_link
8141
8142     parsers.image_opening = #(parsers.exclamation * parsers.inline_link)
8143         * Cg(Cc("inline"), "link_type")

```

```

8144         + #(parsers.exclamation * parsers.full_link)
8145         * Cg(Cc("full"), "link_type")
8146         + #(parsers.exclamation * parsers.collapsed_link)
8147         * Cg(Cc("collapsed"), "link_type")
8148         + #(parsers.exclamation * parsers.shortcut_link)
8149         * Cg(Cc("shortcut"), "link_type")
8150         + #(parsers.exclamation * parsers.empty_link)
8151         * Cg(Cc("empty"), "link_type")
8152
8153     parsers.link_opening = #parsers.inline_link
8154         * Cg(Cc("inline"), "link_type")
8155         + #parsers.full_link
8156         * Cg(Cc("full"), "link_type")
8157         + #parsers.collapsed_link
8158         * Cg(Cc("collapsed"), "link_type")
8159         + #parsers.shortcut_link
8160         * Cg(Cc("shortcut"), "link_type")
8161         + #parsers.empty_link
8162         * Cg(Cc("empty_link"), "link_type")
8163         + #parsers.link_text
8164         * Cg(Cc("link_text"), "link_type")
8165
8166     parsers.link_image_opening = Ct( Cg(Cc("delimiter"), "type")
8167         * Cg(Cc(true), "is_opening")
8168         * Cg(Cc(false), "is_closing")
8169         * ( Cg(Cc("image"), "element")
8170             * parsers.image_opening
8171             * Cg(parsers.exclamation * parsers.lbracket, "content")
8172             + Cg(Cc("link"), "element")
8173             * parsers.link_opening
8174             * Cg(parsers.lbracket, "content")))
8175
8176     parsers.link_image_closing = Ct( Cg(Cc("delimiter"), "type")
8177         * Cg(Cc("link"), "element")
8178         * Cg(Cc(false), "is_opening")
8179         * Cg(Cc(true), "is_closing")
8180         * ( Cg(Cc(true), "is_direct")
8181             * Cg(parsers.rbracket * #parsers.inline_direct_re, "content")
8182             + Cg(Cc(false), "is_direct")
8183             * Cg(parsers.rbracket, "content")))
8184
8185     parsers.link_image_open_or_close = parsers.link_image_opening
8186         + parsers.link_image_closing
8187
8188     if options.html then
8189         parsers.link_emph_precedence = parsers.inticks
8190         + parsers.autolink

```

```

8191                                     + parsers.html_inline_tags
8192     else
8193         parsers.link_emph_precedence = parsers.inticks
8194                                     + parsers.autolink
8195     end
8196
8197     parsers.link_and_emph_endline = parsers.newline
8198                                   * ((parsers.check_minimal_indent
8199                                       * -V("EndlineExceptions")
8200                                       + parsers.check_optional_indent
8201                                       * -V("EndlineExceptions")
8202                                       * -parsers.starter) / "")
8203                                   * parsers.spacechar^0 / "\n"
8204
8205     parsers.link_and_emph_content = Ct( Cg(Cc("content"), "type")
8206                                         * Cg(Cs(( parsers.link_emph_precedence
8207                                             + parsers.backslash * parsers.any
8208                                             + parsers.link_and_emph_endline
8209                                             + (parsers.linechar
8210                                                 - parsers.blankline^2
8211                                                 - parsers.link_image_open_or_close
8212                                                 - parsers.emph_open_or_close))^0), "con
8213
8214     parsers.link_and_emph_table = (parsers.link_image_opening + parsers.emph_open)
8215                                   * parsers.link_and_emph_content
8216                                   * ((parsers.link_image_open_or_close + parsers.emph_ope
8217                                   * parsers.link_and_emph_content)^1
8218

```

Collect the content between the [opening\\_index](#) and [closing\\_index](#) in the delimiter table [t](#).

```

8219     local function collect_link_content(t, opening_index, closing_index)
8220         local content = {}
8221         for i = opening_index, closing_index do
8222             content[#content + 1] = t[i].content
8223         end
8224         return util.ropetostring(content)
8225     end
8226

```

Look for the closest potential link opener in the delimiter table [t](#) in the range from [bottom\\_index](#) to [latest\\_index](#).

```

8227     local function find_link_opener(t, bottom_index, latest_index)
8228         for i = latest_index, bottom_index, -1 do
8229             local value = t[i]
8230             if value.type == "delimiter" and
8231                value.is_opening and
8232                (value.element == "link" or value.element == "image")

```

```

8233         and not value.removed then
8234         if value.is_active then
8235             return i
8236         end
8237         value.removed = true
8238         return nil
8239     end
8240 end
8241 end
8242

```

Find the position of a delimiter that closes a full link after an index `latest_index` in the delimiter table `t`.

```

8243 local function find_next_link_closing_index(t, latest_index)
8244     for i = latest_index, #t do
8245         local value = t[i]
8246         if value.is_closing and
8247             value.element == "link" and
8248             not value.removed then
8249             return i
8250         end
8251     end
8252 end
8253

```

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.

```

8254 local function disable_previous_link_openers(t, opening_index)
8255     if t[opening_index].element == "image" then
8256         return
8257     end
8258
8259     for i = opening_index, 1, -1 do
8260         local value = t[i]
8261         if value.is_active and
8262             value.type == "delimiter" and
8263             value.is_opening and
8264             value.element == "link" then
8265             value.is_active = false
8266         end
8267     end
8268 end
8269

```

Disable the delimiters between the `opening_index` and `closing_index` in the delimiter table `t` by marking them inactive with the `is_active` property.

```

8270 local function disable_range(t, opening_index, closing_index)
8271     for i = opening_index, closing_index do

```

```

8272     local value = t[i]
8273     if value.is_active then
8274         value.is_active = false
8275         if value.type == "delimiter" then
8276             value.removed = true
8277         end
8278     end
8279 end
8280 end
8281

```

Clear the parsed content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

8282 local function delete_parsed_content_in_range(t, opening_index, closing_index)
8283     for i = opening_index, closing_index do
8284         t[i].rendered = nil
8285     end
8286 end
8287

```

Clear the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

8288 local function empty_content_in_range(t, opening_index, closing_index)
8289     for i = opening_index, closing_index do
8290         t[i].content = ''
8291     end
8292 end
8293

```

Join the attributes from the link reference definition `reference_attributes` with the link's own attributes `own_attributes`.

```

8294 local function join_attributes(reference_attributes, own_attributes)
8295     local merged_attributes = {}
8296     for _, attribute in ipairs(reference_attributes or {}) do
8297         table.insert(merged_attributes, attribute)
8298     end
8299     for _, attribute in ipairs(own_attributes or {}) do
8300         table.insert(merged_attributes, attribute)
8301     end
8302     if next(merged_attributes) == nil then
8303         merged_attributes = nil
8304     end
8305     return merged_attributes
8306 end
8307

```

Parse content between two delimiters in the delimiter table `t`. Produce the respective link and image macros.

```

8308 local function render_link_or_image(t, opening_index, closing_index, content_end_index)
8309     process_emphasis(t, opening_index, content_end_index)
8310     local mapped = collect_emphasis_content(t, opening_index + 1, content_end_index - 1)
8311
8312     local rendered = {}
8313     if (t[opening_index].element == "link") then
8314         rendered = writer.link(mapped, reference.url, reference.title, reference.attributes)
8315     end
8316
8317     if (t[opening_index].element == "image") then
8318         rendered = writer.image(mapped, reference.url, reference.title, reference.attributes)
8319     end
8320
8321     t[opening_index].rendered = rendered
8322     delete_parsed_content_in_range(t, opening_index + 1, closing_index)
8323     empty_content_in_range(t, opening_index, closing_index)
8324     disable_previous_link_openers(t, opening_index)
8325     disable_range(t, opening_index, closing_index)
8326 end
8327

```

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.

```

8328 local function resolve_inline_following_content(t, closing_index, match_reference,
8329     local content = ""
8330     for i = closing_index + 1, #t do
8331         content = content .. t[i].content
8332     end
8333
8334     local matching_content = parsers.succeed
8335
8336     if match_reference then
8337         matching_content = matching_content * parsers.inline_direct_ref_inside
8338     end
8339
8340     if match_link_attributes then
8341         matching_content = matching_content * Cg(Ct(parsers.attributes^1), "attributes")
8342     end
8343
8344     local matched = lpeg.match(Ct(matching_content * Cg(Cp(), "end_position")), content)
8345
8346     local matched_count = matched.end_position - 1
8347     for i = closing_index + 1, #t do
8348         local value = t[i]
8349

```

```

8350     local chars_left = matched_count
8351     matched_count = matched_count - #value.content
8352
8353     if matched_count <= 0 then
8354         value.content = value.content:sub(chars_left + 1)
8355         break
8356     end
8357
8358     value.content = ''
8359     value.is_active = false
8360 end
8361
8362 local attributes = matched.attributes
8363 if attributes == nil or next(attributes) == nil then
8364     attributes = nil
8365 end
8366
8367 return {
8368     url = matched.url or "",
8369     title = matched.title or "",
8370     attributes = attributes
8371 }
8372 end
8373

```

Resolve an inline link a<sup>32</sup> 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.

```

8374 local function resolve_inline_link(t, opening_index, closing_index)
8375     local inline_content = resolve_inline_following_content(t, closing_index, true, t)
8376     render_link_or_image(t, opening_index, closing_index, closing_index, inline_content)
8377 end
8378

```

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.

```

8379 local function resolve_shortcut_link(t, opening_index, closing_index)
8380     local content = collect_link_content(t, opening_index + 1, closing_index - 1)
8381     local r = self.lookup_reference(content)
8382
8383     if r then
8384         local inline_content = resolve_inline_following_content(t, closing_index, false, r)
8385         r.attributes = join_attributes(r.attributes, inline_content.attributes)
8386         render_link_or_image(t, opening_index, closing_index, closing_index, r)
8387     end
8388 end

```

---

<sup>32</sup>See [b](#).



8389

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.

```
8390 local function resolve_full_link(t, opening_index, closing_index)
8391     local next_link_closing_index = find_next_link_closing_index(t, closing_index + 4)
8392     local next_link_content = collect_link_content(t, closing_index + 3, next_link_closing_index)
8393     local r = self.lookup_reference(next_link_content)
8394
8395     if r then
8396         local inline_content = resolve_inline_following_content(t, next_link_closing_index,
8397                                                                 t.match_link_attributes)
8398         r.attributes = join_attributes(r.attributes, inline_content.attributes)
8399         render_link_or_image(t, opening_index, next_link_closing_index, closing_index,
8400                             r)
8401     end
8402 end
```

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.

```
8403 local function resolve_collapsed_link(t, opening_index, closing_index)
8404     local next_link_closing_index = find_next_link_closing_index(t, closing_index + 4)
8405     local content = collect_link_content(t, opening_index + 1, closing_index - 1)
8406     local r = self.lookup_reference(content)
8407
8408     if r then
8409         local inline_content = resolve_inline_following_content(t, closing_index, false,
8410                                                                 r.attributes)
8411         r.attributes = join_attributes(r.attributes, inline_content.attributes)
8412         render_link_or_image(t, opening_index, next_link_closing_index, closing_index,
8413                             r)
8414     end
8415 end
```

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.

```
8415 local function process_links_and_emphasis(t)
8416     for _,value in ipairs(t) do
8417         value.is_active = true
8418     end
8419
8420     for i,value in ipairs(t) do
8421         if not value.is_closing or
8422            value.type ~= "delimiter" or
8423            not (value.element == "link" or value.element == "image") then
8424             goto continue
8425         end
8426     end
```

```

8425     end
8426
8427     local opener_position = find_link_opener(t, 1, i - 1)
8428     if (opener_position == nil) then
8429         goto continue
8430     end
8431
8432     local opening_delimiter = t[opener_position]
8433     opening_delimiter.removed = true
8434
8435     local link_type = opening_delimiter.link_type
8436
8437     if (link_type == "inline") then
8438         resolve_inline_link(t, opener_position, i)
8439     end
8440     if (link_type == "shortcut") then
8441         resolve_shortcut_link(t, opener_position, i)
8442     end
8443     if (link_type == "full") then
8444         resolve_full_link(t, opener_position, i)
8445     end
8446     if (link_type == "collapsed") then
8447         resolve_collapsed_link(t, opener_position, i)
8448     end
8449
8450     ::continue::
8451 end
8452
8453 t[#t].content = t[#t].content:gsub("%s*$", "")
8454
8455 process_emphasis(t, 1, #t)
8456 local final_result = collect_emphasis_content(t, 1, #t)
8457 return final_result
8458 end
8459
8460 function self.defer_link_and_emphasis_processing(delimiter_table)
8461     return writer.defer_call(function()
8462         return process_links_and_emphasis(delimiter_table)
8463     end)
8464 end
8465

```

### 3.1.5.8 Inline Elements (local)

```

8466 parsers.Str      = (parsers.normalchar * (parsers.normalchar + parsers.at)^0)
8467                  / writer.string
8468

```

```

8469 parsers.Symbol    = (parsers.backtick^1 + V("SpecialChar"))
8470                    / writer.string
8471
8472 parsers.Ellipsis   = P("...") / writer.ellipsis
8473
8474 parsers.Smart      = parsers.Ellipsis
8475
8476 parsers.Code       = parsers.inticks / writer.code
8477
8478 if options.blankBeforeBlockquote then
8479   parsers.bqstart = parsers.fail
8480 else
8481   parsers.bqstart = parsers.blockquote_start
8482 end
8483
8484 if options.blankBeforeHeading then
8485   parsers.headerstart = parsers.fail
8486 else
8487   parsers.headerstart = parsers.atx_heading
8488 end
8489
8490 if options.blankBeforeList then
8491   parsers.interrupting_bullets = parsers.fail
8492   parsers.interrupting_enumerators = parsers.fail
8493 else
8494   parsers.interrupting_bullets = parsers.bullet(parsers.dash, true)
8495                               + parsers.bullet(parsers.asterisk, true)
8496                               + parsers.bullet(parsers.plus, true)
8497
8498   parsers.interrupting_enumerators = parsers.enumerator(parsers.period, true)
8499                                   + parsers.enumerator(parsers.rparent, true)
8500 end
8501
8502 if options.html then
8503   parsers.html_interrupting = parsers.check_trail
8504                               * ( parsers.html_incomplete_open_tag
8505                                   + parsers.html_incomplete_close_tag
8506                                   + parsers.html_incomplete_open_special_tag
8507                                   + parsers.html_comment_start
8508                                   + parsers.html_cdatasection_start
8509                                   + parsers.html_declaration_start
8510                                   + parsers.html_instruction_start
8511                                   - parsers.html_close_special_tag
8512                                   - parsers.html_empty_special_tag)
8513 else
8514   parsers.html_interrupting = parsers.fail
8515 end

```

```

8516
8517 parsers.EndlineExceptions
8518         = parsers.blankline -- paragraph break
8519         + parsers.eof       -- end of document
8520         + parsers.bqstart
8521         + parsers.thematic_break_lines
8522         + parsers.interrupting_bullets
8523         + parsers.interrupting_enumerators
8524         + parsers.headerstart
8525         + parsers.html_interrupting
8526
8527 parsers.NoSoftLineBreakEndlineExceptions = parsers.EndlineExceptions
8528
8529 parsers.endline = parsers.newline
8530         * (parsers.check_minimal_indent
8531         * -V("EndlineExceptions")
8532         + parsers.check_optional_indent
8533         * -V("EndlineExceptions")
8534         * -parsers.starter)
8535         * parsers.spacechar^0
8536
8537 parsers.Endline = parsers.endline
8538         / writer.soft_line_break
8539
8540 parsers.EndlineNoSub = parsers.endline
8541
8542 parsers.NoSoftLineBreakEndline
8543         = parsers.newline
8544         * (parsers.check_minimal_indent
8545         * -V("NoSoftLineBreakEndlineExceptions")
8546         + parsers.check_optional_indent
8547         * -V("NoSoftLineBreakEndlineExceptions")
8548         * -parsers.starter)
8549         * parsers.spacechar^0
8550         / writer.space
8551
8552 parsers.EndlineBreak = parsers.backslash * parsers.Endline
8553         / writer.hard_line_break
8554
8555 parsers.OptionalIndent
8556         = parsers.spacechar^1 / writer.space
8557
8558 parsers.Space = parsers.spacechar^2 * parsers.Endline
8559         / writer.hard_line_break
8560         + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / self.
8561         + parsers.spacechar^1 * parsers.Endline
8562         / writer.soft_line_break

```

```

8563             + parsers.spacechar^1 * -parsers.newline / self.expandtabs
8564
8565     parsers.NoSoftLineBreakSpace
8566         = parsers.spacechar^2 * parsers.Endline
8567           / writer.hard_line_break
8568         + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / self.
8569         + parsers.spacechar^1 * parsers.Endline
8570           / writer.soft_line_break
8571         + parsers.spacechar^1 * -parsers.newline / self.expandtabs
8572
8573     parsers.NonbreakingEndline
8574         = parsers.endline
8575         / writer.soft_line_break
8576
8577     parsers.NonbreakingSpace
8578         = parsers.spacechar^2 * parsers.Endline
8579           / writer.hard_line_break
8580         + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
8581         + parsers.spacechar^1 * parsers.Endline
8582           * parsers.optionalspace
8583           / writer.soft_line_break
8584         + parsers.spacechar^1 * parsers.optionalspace
8585           / writer.nbsp
8586

```

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.

```

8587 function self.auto_link_url(url, attributes)
8588     return writer.link(writer.escape(url),
8589                       url, nil, attributes)
8590 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.

```

8591 function self.auto_link_email(email, attributes)
8592     return writer.link(writer.escape(email),
8593                       "mailto:".email,
8594                       nil, attributes)
8595 end
8596
8597 parsers.AutoLinkUrl = parsers.auto_link_url
8598                     / self.auto_link_url
8599
8600 parsers.AutoLinkEmail
8601     = parsers.auto_link_email

```

```

8602             / self.auto_link_email
8603
8604     parsers.AutoLinkRelativeReference
8605         = parsers.auto_link_relative_reference
8606         / self.auto_link_url
8607
8608     parsers.LinkAndEmph = Ct(parsers.link_and_emph_table)
8609         / self.defer_link_and_emphasis_processing
8610
8611     parsers.EscapedChar = parsers.backslash * C(parsers.escapable) / writer.string
8612
8613     parsers.InlineHtml = Cs(parsers.html_inline_comment) / writer.inline_html_comment
8614         + Cs(parsers.html_any_empty_inline_tag
8615             + parsers.html_inline_instruction
8616             + parsers.html_inline_cdatasection
8617             + parsers.html_inline_declaration
8618             + parsers.html_any_open_inline_tag
8619             + parsers.html_any_close_tag)
8620         / writer.inline_html_tag
8621
8622     parsers.HtmlEntity = parsers.html_entities / writer.string

```

### 3.1.5.9 Block Elements (local)

```

8623     parsers.DisplayHtml = Cs(parsers.check_trail
8624         * ( parsers.html_comment
8625           + parsers.html_special_block
8626           + parsers.html_block
8627           + parsers.html_any_block
8628           + parsers.html_instruction
8629           + parsers.html_cdatasection
8630           + parsers.html_declaration))
8631         / writer.block_html_element
8632
8633     parsers.indented_non_blank_line = parsers.indentedline - parsers.blankline
8634
8635     parsers.Verbatim = Cs(
8636         parsers.check_code_trail
8637         * (parsers.line - parsers.blankline)
8638         * ((parsers.check_minimal_blank_indent_and_full_code_trail * pa
8639           * ((parsers.check_minimal_indent / "") * parsers.check_code_t
8640             * (parsers.line - parsers.blankline))^1)^0
8641         ) / self.expandtabs / writer.verbatim
8642
8643     parsers.Blockquote = parsers.blockquote_body
8644         / writer.blockquote
8645

```

```

8646 parsers.ThematicBreak = parsers.thematic_break_lines
8647                        / writer.thematic_break
8648
8649 parsers.Reference      = parsers.define_reference_parser
8650                        / self.register_link
8651
8652 parsers.Paragraph      = parsers.freeze_trail
8653                        * (Ct((parsers.Inline)^1)
8654                        * (parsers.newline + parsers.eof)
8655                        * parsers.unfreeze_trail
8656                        / writer.paragraph)
8657
8658 parsers.Plain           = parsers.nonindentspace * Ct(parsers.Inline^1)
8659                        / writer.plain

```

### 3.1.5.10 Lists (local)

```

8660
8661 if options.taskLists then
8662   parsers.tickbox = ( parsers.ticked_box
8663                     + parsers.halfticked_box
8664                     + parsers.unticked_box
8665                     ) / writer.tickbox
8666 else
8667   parsers.tickbox = parsers.fail
8668 end
8669
8670 parsers.list_blank = parsers.conditionally_indented_blankline
8671
8672 parsers.ref_or_block_list_separated = parsers.sep_group_no_output(parsers.list_blank
8673                          * parsers.minimally_indented_ref
8674                          + parsers.block_sep_group(parsers.list_blank)
8675                          * parsers.minimally_indented_block
8676
8677 parsers.ref_or_block_non_separated  = parsers.minimally_indented_ref
8678                          + (parsers.succeed / writer.interblocksep)
8679                          * parsers.minimally_indented_block
8680                          - parsers.minimally_indented_blankline
8681
8682 parsers.tight_list_loop_body_pair =
8683   parsers.create_loop_body_pair(parsers.ref_or_block_non_separated,
8684                                parsers.minimally_indented_par_or_plain_no_blank,
8685                                (parsers.succeed / writer.interblocksep),
8686                                (parsers.succeed / writer.paragraphsep))
8687
8688 parsers.loose_list_loop_body_pair =
8689   parsers.create_loop_body_pair(parsers.ref_or_block_list_separated,

```

```

8690         parsers.minimally_indented_par_or_plain,
8691         parsers.block_sep_group(parsers.list_blank),
8692         parsers.par_sep_group(parsers.list_blank))
8693
8694     parsers.tight_list_content_loop = V("Block")
8695         * parsers.tight_list_loop_body_pair.block^0
8696         + (V("Paragraph") + V("Plain"))
8697         * parsers.ref_or_block_non_separated
8698         * parsers.tight_list_loop_body_pair.block^0
8699         + (V("Paragraph") + V("Plain"))
8700         * parsers.tight_list_loop_body_pair.par^0
8701
8702     parsers.loose_list_content_loop = V("Block")
8703         * parsers.loose_list_loop_body_pair.block^0
8704         + (V("Paragraph") + V("Plain"))
8705         * parsers.ref_or_block_list_separated
8706         * parsers.loose_list_loop_body_pair.block^0
8707         + (V("Paragraph") + V("Plain"))
8708         * parsers.loose_list_loop_body_pair.par^0
8709
8710     parsers.list_item_tightness_condition = -( parsers.list_blank^0
8711         * parsers.minimally_indented_ref_or_block
8712         * remove_indent("li")
8713         + remove_indent("li")
8714         * parsers.fail
8715
8716     parsers.indented_content_tight = Ct( (parsers.blankline / "")
8717         * #parsers.list_blank
8718         * remove_indent("li")
8719         + ( (V("Reference") + (parsers.blankline / ""))
8720         * parsers.check_minimal_indent
8721         * parsers.tight_list_content_loop
8722         + (V("Reference") + (parsers.blankline / ""))
8723         + (parsers.tickbox^-1 / writer.escape)
8724         * parsers.tight_list_content_loop
8725         )
8726         * parsers.list_item_tightness_condition
8727     )
8728
8729     parsers.indented_content_loose = Ct( (parsers.blankline / "")
8730         * #parsers.list_blank
8731         + ( (V("Reference") + (parsers.blankline / ""))
8732         * parsers.check_minimal_indent
8733         * parsers.loose_list_content_loop
8734         + (V("Reference") + (parsers.blankline / ""))
8735         + (parsers.tickbox^-1 / writer.escape)
8736         * parsers.loose_list_content_loop

```



```

8737         )
8738     )
8739
8740     parsers.TightListItem = function(starter)
8741     return -parsers.ThematicBreak
8742         * parsers.add_indent(starter, "li")
8743         * parsers.indented_content_tight
8744     end
8745
8746     parsers.LooseListItem = function(starter)
8747     return -parsers.ThematicBreak
8748         * parsers.add_indent(starter, "li")
8749         * parsers.indented_content_loose
8750         * remove_indent("li")
8751     end
8752
8753     parsers.BulletListOfType = function(bullet_type)
8754     local bullet = parsers.bullet(bullet_type)
8755     return ( Ct( parsers.TightListItem(bullet)
8756         * ( (parsers.check_minimal_indent / "")
8757         * parsers.TightListItem(bullet)
8758         )^0
8759     )
8760     * Cc(true)
8761     * -#( (parsers.list_blank^0 / "")
8762         * parsers.check_minimal_indent
8763         * (bullet - parsers.ThematicBreak)
8764     )
8765     + Ct( parsers.LooseListItem(bullet)
8766         * ( (parsers.list_blank^0 / "")
8767         * (parsers.check_minimal_indent / "")
8768         * parsers.LooseListItem(bullet)
8769         )^0
8770     )
8771     * Cc(false)
8772     ) / writer.bulletlist
8773 end
8774
8775 parsers.BulletList = parsers.BulletListOfType(parsers.dash)
8776     + parsers.BulletListOfType(parsers.asterisk)
8777     + parsers.BulletListOfType(parsers.plus)
8778
8779 local function ordered_list(items,tight,starter)
8780     local startnum = starter[2][1]
8781     if options.startNumber then
8782         startnum = tonumber(startnum) or 1 -- fallback for '#'
8783         if startnum ~= nil then

```

```

8784         startnum = math.floor(startnum)
8785     end
8786 else
8787     startnum = nil
8788 end
8789 return writer.orderedlist(items,tight,startnum)
8790 end
8791
8792 parsers.OrderedListOfType = function(delimiter_type)
8793     local enumerator = parsers.enumerator(delimiter_type)
8794     return Cg(enumerator, "listtype")
8795         * (Ct( parsers.TightListItem(Cb("listtype"))
8796             * ((parsers.check_minimal_indent / "") * parsers.TightListItem(enumerat
8797             * Cc(true)
8798             * -#((parsers.list_blank^0 / "")
8799             * parsers.check_minimal_indent * enumerator)
8800         + Ct( parsers.LooseListItem(Cb("listtype"))
8801             * ((parsers.list_blank^0 / "")
8802             * (parsers.check_minimal_indent / "") * parsers.LooseListItem(enumerat
8803             * Cc(false)
8804         ) * Ct(Cb("listtype"))) / ordered_list
8805 end
8806
8807 parsers.OrderedList = parsers.OrderedListOfType(parsers.period)
8808     + parsers.OrderedListOfType(parsers.rparent)

```

### 3.1.5.11 Blank (local)

```

8809 parsers.Blank          = parsers.blankline / ""
8810                        + V("Reference")

```

### 3.1.5.12 Headings (local)

```

8811 function parsers.parse_heading_text(s)
8812     local inlines = self.parser_functions.parse_inlines(s)
8813     local flatten_inlines = self.writer.flatten_inlines
8814     self.writer.flatten_inlines = true
8815     local flat_text = self.parser_functions.parse_inlines(s)
8816     flat_text = util.rope_to_string(flat_text)
8817     self.writer.flatten_inlines = flatten_inlines
8818     return {flat_text, inlines}
8819 end
8820
8821 -- parse atx header
8822 parsers.AtxHeading = parsers.check_trail_no_rem
8823                     * Cg(parsers.heading_start, "level")
8824                     * ((C( parsers.optionalspace
8825                         * parsers.hash^0

```

```

8826             * parsers.optionalspace
8827             * parsers.newline)
8828         + parsers.spacechar~1
8829         * C(parsers.line))
8830     / strip_atx_end
8831     / parsers.parse_heading_text)
8832 * Cb("level")
8833 / writer.heading
8834
8835 parsers.heading_line = parsers.linechar~1
8836                       - parsers.thematic_break_lines
8837
8838 parsers.heading_text = parsers.heading_line
8839                       * ((V("Endline") / "\n") * (parsers.heading_line - parsers.heading_line))
8840                       * parsers.newline~-1
8841
8842 parsers.SettextHeading = parsers.freeze_trail * parsers.check_trail_no_rem
8843                       * #(parsers.heading_text
8844                           * parsers.check_minimal_indent * parsers.check_trail * parsers.heading_text)
8845                       * Cs(parsers.heading_text)
8846                       / parsers.parse_heading_text
8847                       * parsers.check_minimal_indent_and_trail * parsers.heading_line
8848                       * parsers.newline
8849                       * parsers.unfreeze_trail
8850                       / writer.heading
8851
8852 parsers.Heading = parsers.AtHeading + parsers.SettextHeading

```

### 3.1.5.13 Syntax Specification

Define `reader->finalize_grammar` as a function that constructs the PEG grammar of markdown, applies syntax extensions `extensions` and returns a conversion function that takes a markdown string and turns it into a plain  $\text{\TeX}$  output.

```

8853 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.

```

8854     local walkable_syntax = (function(global_walkable_syntax)
8855         local local_walkable_syntax = {}
8856         for lhs, rule in pairs(global_walkable_syntax) do
8857             local_walkable_syntax[lhs] = util.table_copy(rule)
8858         end
8859         return local_walkable_syntax
8860     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.

```

8861     local current_extension_name = nil
8862     self.insert_pattern = function(selector, pattern, pattern_name)
8863         assert(pattern_name == nil or type(pattern_name) == "string")
8864         local _, _, lhs, pos, rhs = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
8865         assert(lhs ~= nil,
8866             [[Expected selector in form "LHS (before|after|instead of) RHS", not "]]
8867             .. selector .. [["]]
8868         assert(walkable_syntax[lhs] ~= nil,
8869             [[Rule ]] .. lhs .. [[ -> ... does not exist in markdown grammar]])
8870         assert(pos == "before" or pos == "after" or pos == "instead of",
8871             [[Expected positional specifier "before", "after", or "instead of", not "]]
8872             .. pos .. [["]]
8873         local rule = walkable_syntax[lhs]
8874         local index = nil
8875         for current_index, current_rhs in ipairs(rule) do
8876             if type(current_rhs) == "string" and current_rhs == rhs then
8877                 index = current_index
8878                 if pos == "after" then
8879                     index = index + 1
8880                 end
8881                 break
8882             end
8883         end
8884         assert(index ~= nil,
8885             [[Rule ]] .. lhs .. [[ -> ]] .. rhs
8886             .. [[ does not exist in markdown grammar]])
8887         local accountable_pattern
8888         if current_extension_name then
8889             accountable_pattern = { pattern, current_extension_name, pattern_name }
8890         else
8891             assert(type(pattern) == "string",
8892                 [[reader->insert_pattern() was called outside an extension with ]]
8893                 .. [[a PEG pattern instead of a rule name]])
8894             accountable_pattern = pattern
8895         end
8896         if pos == "instead of" then
8897             rule[index] = accountable_pattern
8898         else
8899             table.insert(rule, index, accountable_pattern)
8900         end
8901     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.

```

8902     local syntax =
8903         { "Blocks",
8904
8905           Blocks = V("InitializeState")
8906                 * ( V("ExpectedJekyllData")
8907                   * (V("Blank")^0 / writer.interblocksep)
8908                     )^-1
8909                 * 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.

```

8910                 * ( V("Block")
8911                   * ( V("Blank")^0 * parsers.eof
8912                     + ( V("Blank")^2 / writer.paragraphsep
8913                       + V("Blank")^0 / writer.interblocksep
8914                       )
8915                   )
8916                   + ( V("Paragraph") + V("Plain") )
8917                 * ( V("Blank")^0 * parsers.eof
8918                   + ( V("Blank")^2 / writer.paragraphsep
8919                     + V("Blank")^0 / writer.interblocksep
8920                     )
8921                 )
8922                 * V("Block")
8923                 * ( V("Blank")^0 * parsers.eof
8924                   + ( V("Blank")^2 / writer.paragraphsep
8925                     + V("Blank")^0 / writer.interblocksep
8926                     )
8927                 )
8928                 + ( V("Paragraph") + V("Plain") )
8929                 * ( V("Blank")^0 * parsers.eof
8930                   + V("Blank")^0 / writer.paragraphsep
8931                 )
8932             )^0,
8933
8934     ExpectedJekyllData = parsers.fail,
8935
8936     Blank = parsers.Blank,
8937     Reference = parsers.Reference,
8938
8939     Blockquote = parsers.Blockquote,
8940     Verbatim = parsers.Verbatim,
8941     ThematicBreak = parsers.ThematicBreak,
8942     BulletList = parsers.BulletList,
8943     OrderedList = parsers.OrderedList,
8944     DisplayHtml = parsers.DisplayHtml,

```

```

8945     Heading                = parsers Heading,
8946     Paragraph               = parsers Paragraph,
8947     Plain                   = parsers Plain,
8948
8949     EndlineExceptions       = parsers EndlineExceptions,
8950     NoSoftLineBreakEndlineExceptions
8951                             = parsers.NoSoftLineBreakEndlineExceptions,
8952
8953     Str                      = parsers.Str,
8954     Space                   = parsers.Space,
8955     NoSoftLineBreakSpace    = parsers.NoSoftLineBreakSpace,
8956     OptionalIndent          = parsers.OptionalIndent,
8957     Endline                  = parsers.Endline,
8958     EndlineNoSub             = parsers.EndlineNoSub,
8959     NoSoftLineBreakEndline
8960                             = parsers.NoSoftLineBreakEndline,
8961     EndlineBreak             = parsers.EndlineBreak,
8962     LinkAndEmph              = parsers.LinkAndEmph,
8963     Code                     = parsers.Code,
8964     AutoLinkUrl              = parsers.AutoLinkUrl,
8965     AutoLinkEmail            = parsers.AutoLinkEmail,
8966     AutoLinkRelativeReference
8967                             = parsers.AutoLinkRelativeReference,
8968     InlineHtml               = parsers.InlineHtml,
8969     HtmlEntity               = parsers.HtmlEntity,
8970     EscapedChar              = parsers.EscapedChar,
8971     Smart                    = parsers.Smart,
8972     Symbol                   = parsers.Symbol,
8973     SpecialChar              = parsers.fail,
8974     InitializeState          = parsers.succeed,
8975 }

```

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]`.

```

8976     self.update_rule = function(rule_name, get_pattern)
8977         assert(current_extension_name ~= nil)
8978         assert(syntax[rule_name] ~= nil,
8979             [[Rule ]] .. rule_name .. [[ -> ... does not exist in markdown grammar]])
8980         local previous_pattern
8981         local extension_name
8982         if walkable_syntax[rule_name] then
8983             local previous_accountable_pattern = walkable_syntax[rule_name][1]
8984             previous_pattern = previous_accountable_pattern[1]
8985             extension_name = previous_accountable_pattern[2] .. ", " .. current_extension

```

```

8986     else
8987         previous_pattern = nil
8988         extension_name = current_extension_name
8989     end
8990     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

```

```

8991     if type(get_pattern) == "function" then
8992         pattern = get_pattern(previous_pattern)
8993     else
8994         assert(previous_pattern == nil,
8995             [[Rule ]] .. rule_name ..
8996             [[ has already been updated by ]] .. extension_name)
8997         pattern = get_pattern
8998     end
8999     local accountable_pattern = { pattern, extension_name, rule_name }
9000     walkable_syntax[rule_name] = { accountable_pattern }
9001 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.

```

9002     local special_characters = {}
9003     self.add_special_character = function(c)
9004         table.insert(special_characters, c)
9005         syntax.SpecialChar = S(table.concat(special_characters, ""))
9006     end
9007
9008     self.add_special_character("*")
9009     self.add_special_character("[")
9010     self.add_special_character("]")
9011     self.add_special_character("<")
9012     self.add_special_character("!")
9013     self.add_special_character("\\")

```

Add method `reader->initialize_named_group` that defines named groups with a default capture value.

```

9014     self.initialize_named_group = function(name, value)

```

```

9015     local pattern = Ct("")
9016     if value ~= nil then
9017         pattern = pattern / value
9018     end
9019     syntax.InitializeState = syntax.InitializeState
9020                             * Cg(pattern, name)
9021 end

```

Add a named group for indentation.

```

9022     self.initialize_named_group("indent_info")

```

Apply syntax extensions.

```

9023     for _, extension in ipairs(extensions) do
9024         current_extension_name = extension.name
9025         extension.extend_writer(writer)
9026         extension.extend_reader(self)
9027     end
9028     current_extension_name = nil

```

If the [debugExtensions](#) option is enabled, serialize [walkable\\_syntax](#) to a JSON for debugging purposes.

```

9029     if options.debugExtensions then
9030         local sorted_lhs = {}
9031         for lhs, _ in pairs(walkable_syntax) do
9032             table.insert(sorted_lhs, lhs)
9033         end
9034         table.sort(sorted_lhs)
9035
9036         local output_lines = {"{"}
9037         for lhs_index, lhs in ipairs(sorted_lhs) do
9038             local encoded_lhs = util.encode_json_string(lhs)
9039             table.insert(output_lines, [{" "] .. encoded_lhs .. [{" ": []}])
9040             local rule = walkable_syntax[lhs]
9041             for rhs_index, rhs in ipairs(rule) do
9042                 local human_readable_rhs
9043                 if type(rhs) == "string" then
9044                     human_readable_rhs = rhs
9045                 else
9046                     local pattern_name
9047                     if rhs[3] then
9048                         pattern_name = rhs[3]
9049                     else
9050                         pattern_name = "Anonymous Pattern"
9051                     end
9052                     local extension_name = rhs[2]
9053                     human_readable_rhs = pattern_name .. [{" ("] .. extension_name .. [{" )"]}
9054                 end
9055                 local encoded_rhs = util.encode_json_string(human_readable_rhs)

```



```

9056         local output_line = [[          ]] .. encoded_rhs
9057         if rhs_index < #rule then
9058             output_line = output_line .. ","
9059         end
9060         table.insert(output_lines, output_line)
9061     end
9062     local output_line = "    ]"
9063     if lhs_index < #sorted_lhs then
9064         output_line = output_line .. ","
9065     end
9066     table.insert(output_lines, output_line)
9067 end
9068 table.insert(output_lines, "}")
9069
9070 local output = table.concat(output_lines, "\n")
9071 local output_filename = options.debugExtensionsFileName
9072 local output_file = assert(io.open(output_filename, "w"),
9073     [[Could not open file ]] .. output_filename .. [[ for writing]])
9074 assert(output_file:write(output))
9075 assert(output_file:close())
9076 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](#).

```

9077     for lhs, rule in pairs(walkable_syntax) do
9078         syntax[lhs] = parsers.fail
9079         for _, rhs in ipairs(rule) do
9080             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.

```

9081         if type(rhs) == "string" then
9082             pattern = V(rhs)
9083         else
9084             pattern = rhs[1]
9085             if type(pattern) == "string" then
9086                 pattern = V(pattern)
9087             end
9088         end
9089         syntax[lhs] = syntax[lhs] + pattern
9090     end
9091 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.

```
9092     if options.underscores then
9093         self.add_special_character("_")
9094     end
9095
9096     if not options.codeSpans then
9097         syntax.Code = parsers.fail
9098     else
9099         self.add_special_character("`")
9100     end
9101
9102     if not options.html then
9103         syntax.DisplayHtml = parsers.fail
9104         syntax.InlineHtml = parsers.fail
9105         syntax.HtmlEntity = parsers.fail
9106     else
9107         self.add_special_character("&")
9108     end
9109
9110     if options.preserveTabs then
9111         options.stripIndent = false
9112     end
9113
9114     if not options.smartEllipses then
9115         syntax.Smart = parsers.fail
9116     else
9117         self.add_special_character(".")
9118     end
9119
9120     if not options.relativeReferences then
9121         syntax.AutoLinkRelativeReference = parsers.fail
9122     end
9123
9124     if options.contentLevel == "inline" then
9125         syntax[1] = "Inlines"
9126         syntax.Inlines = V("InitializeState")
9127             * parsers.Inline^0
9128             * ( parsers.spacing^0
9129             * parsers.eof / "")
9130         syntax.Space = parsers.Space + parsers.blankline / writer.space
9131     end
9132
9133     local blocks_nested_t = util.table_copy(syntax)
9134     blocks_nested_t.ExpectedJekyllData = parsers.fail
9135     parsers.blocks_nested = Ct(blocks_nested_t)
```

```

9136
9137     parsers.blocks = Ct(syntax)
9138
9139     local inlines_t = util.table_copy(syntax)
9140     inlines_t[1] = "Inlines"
9141     inlines_t.Inlines = V("InitializeState")
9142         * parsers.Inline~0
9143         * ( parsers.spacing~0
9144           * parsers.eof / "")
9145     parsers.inlines = Ct(inlines_t)
9146
9147     local inlines_no_inline_note_t = util.table_copy(inlines_t)
9148     inlines_no_inline_note_t.InlineNote = parsers.fail
9149     parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
9150
9151     local inlines_no_html_t = util.table_copy(inlines_t)
9152     inlines_no_html_t.DisplayHtml = parsers.fail
9153     inlines_no_html_t.InlineHtml = parsers.fail
9154     inlines_no_html_t.HtmlEntity = parsers.fail
9155     parsers.inlines_no_html = Ct(inlines_no_html_t)
9156
9157     local inlines_nbsp_t = util.table_copy(inlines_t)
9158     inlines_nbsp_t.Endline = parsers.NonbreakingEndline
9159     inlines_nbsp_t.Space = parsers.NonbreakingSpace
9160     parsers.inlines_nbsp = Ct(inlines_nbsp_t)
9161
9162     local inlines_no_link_or_emphasis_t = util.table_copy(inlines_t)
9163     inlines_no_link_or_emphasis_t.LinkAndEmph = parsers.fail
9164     inlines_no_link_or_emphasis_t.EndlineExceptions = parsers.EndlineExceptions - par
9165     parsers.inlines_no_link_or_emphasis = Ct(inlines_no_link_or_emphasis_t)

```

Return a function that converts markdown string `input` into a plain T<sub>E</sub>X output and returns it..

```

9166     return function(input)

```

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.

```

9167         input = input:gsub("\r\n?", "\n")
9168         if input:sub(-1) ~= "\n" then
9169             input = input .. "\n"
9170         end

```

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). The `cacheDir` option is disregarded.

```

9171         references = {}
9172         local opt_string = {}
9173         for k, _ in pairs(defaultOptions) do

```

```

9174     local v = options[k]
9175     if type(v) == "table" then
9176         for _, i in ipairs(v) do
9177             opt_string[#opt_string+1] = k .. "=" .. tostring(i)
9178         end
9179     elseif k ~= "cacheDir" then
9180         opt_string[#opt_string+1] = k .. "=" .. tostring(v)
9181     end
9182 end
9183 table.sort(opt_string)
9184 local salt = table.concat(opt_string, ",") .. "," .. metadata.version
9185 local output

```

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

```

9186     local function convert(input)
9187         local document = self.parser_functions.parse_blocks(input)
9188         return util.rope_to_string(writer.document(document))
9189     end
9190     if options.eagerCache or options.finalizeCache then
9191         local name = util.cache(options.cacheDir, input, salt, convert,
9192                                ".md" .. writer.suffix)
9193         output = writer.pack(name)

```

Otherwise, return the result of the conversion directly.

```

9194     else
9195         output = convert(input)
9196     end

```

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

```

9197     if options.finalizeCache then
9198         local file, mode
9199         if options.frozenCacheCounter > 0 then
9200             mode = "a"
9201         else
9202             mode = "w"
9203         end
9204         file = assert(io.open(options.frozenCacheFileName, mode),
9205                        [[Could not open file ]] .. options.frozenCacheFileName
9206                        .. [[ for writing]])
9207         assert(file:write([[\\expandafter\\global\\expandafter\\def\\csname ]]
9208                           .. [[markdownFrozenCache]] .. options.frozenCacheCounter
9209                           .. [[\\endcsname{]] .. output .. [[]]] .. "\\n"))
9210         assert(file:close())
9211     end
9212     return output

```

```

9213     end
9214   end
9215   return self
9216 end

```

### 3.1.6 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.

```

9217 M.extensions = {}

```

#### 3.1.6.1 Bracketed Spans

The `extensions.bracketed_spans` function implements the Pandoc bracketed span syntax extension.

```

9218 M.extensions.bracketed_spans = function()
9219   return {
9220     name = "built-in bracketed_spans syntax extension",
9221     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.

```

9222       function self.span(s, attr)
9223         if self.flatten_inlines then return s end
9224         return {"\\markdownRendererBracketedSpanAttributeContextBegin",
9225               self.attributes(attr),
9226               s,
9227               "\\markdownRendererBracketedSpanAttributeContextEnd{}}"}
9228       end
9229   end, extend_reader = function(self)
9230     local parsers = self.parsers
9231     local writer = self.writer
9232
9233     local span_label = parsers.lbracket
9234                       * (Cs((parsers.alphanumeric~1
9235                           + parsers.inticks
9236                           + parsers.autolink
9237                           + V("InlineHtml")
9238                           + ( parsers.backslash * parsers.backslash)
9239                           + ( parsers.backslash * (parsers.lbracket + parsers.rbracket)
9240                           + V("Space") + V("Endline")
9241                           + (parsers.any
9242                             - (parsers.newline + parsers.lbracket + parsers.rbracket
9243                               + parsers.blankline~2))))~1)

```

```

9244             / self.parser_functions.parse_inlines)
9245             * parsers.rbracket
9246
9247         local Span = span_label
9248             * Ct(parsers.attributes)
9249             / writer.span
9250
9251         self.insert_pattern("Inline before LinkAndEmph",
9252             Span, "Span")
9253     end
9254 }
9255 end

```

### 3.1.6.2 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.

```

9256 M.extensions.citations = function(citation_nbsps)
9257     return {
9258         name = "built-in citations syntax extension",
9259         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.

```

9260     function self.citations(text_cites, cites)
9261         local buffer = {}
9262         if self.flatten_inlines then
9263             for _,cite in ipairs(cites) do
9264                 if cite.prenote then
9265                     table.insert(buffer, {cite.prenote, " "})
9266                 end
9267                 table.insert(buffer, cite.name)

```

```

9268         if cite.postnote then
9269             table.insert(buffer, {" ", cite.postnote})
9270         end
9271     end
9272 else
9273     table.insert(buffer, {"\\markdownRenderer", text_cites and "TextCite" or "Cite"})
9274     {"", #cites, ""})
9275     for _,cite in ipairs(cites) do
9276         table.insert(buffer, {cite.suppress_author and "-" or "+", "{",
9277             cite.prenote or "", "}{" , cite.postnote or "", "}{" , cite.name, ""})
9278     end
9279 end
9280 return buffer
9281 end
9282 end, extend_reader = function(self)
9283     local parsers = self.parsers
9284     local writer = self.writer
9285
9286     local citation_chars
9287         = parsers.alphanumeric
9288         + S("#$%&-+<>~/_")
9289
9290     local citation_name
9291         = Cs(parsers.dash~-1) * parsers.at
9292         * Cs(citation_chars
9293             * (((citation_chars + parsers.internal_punctuation
9294                 - parsers.comma - parsers.semicolon)
9295                 * -#((parsers.internal_punctuation - parsers.comma
9296                     - parsers.semicolon)^0
9297                     * -(citation_chars + parsers.internal_punctuation
9298                         - parsers.comma - parsers.semicolon)))^0
9299                 * citation_chars)^-1)
9300
9301     local citation_body_prenote
9302         = Cs((parsers.alphanumeric^1
9303             + parsers.bracketed
9304             + parsers.inticks
9305             + parsers.autolink
9306             + V("InlineHtml")
9307             + V("Space") + V("Endline")
9308             + (parsers.anyescaped
9309                 - (parsers.newline + parsers.rbracket + parsers.blankline^0
9310                     - (parsers.spnl * parsers.dash~-1 * parsers.at))^0)
9311
9312     local citation_body_postnote
9313         = Cs((parsers.alphanumeric^1
9314             + parsers.bracketed

```

```

9315         + parsers.inticks
9316         + parsers.autolink
9317         + V("InlineHtml")
9318         + V("Space") + V("Endline")
9319         + (parsers.anyescaped
9320           - (parsers.newline + parsers.rbracket + parsers.semicolon
9321             + parsers.blankline^2))
9322         - (parsers.spnl * parsers.rbracket))^0)
9323
9324     local citation_body_chunk
9325         = citation_body_prenote
9326         * parsers.spnlc * citation_name
9327         * (parsers.internal_punctuation - parsers.semicolon)^-
1
9328         * parsers.spnlc * citation_body_postnote
9329
9330     local citation_body
9331         = citation_body_chunk
9332         * (parsers.semicolon * parsers.spnlc
9333           * citation_body_chunk)^0
9334
9335     local citation_headless_body_postnote
9336         = Cs((parsers.alphanumeric^1
9337             + parsers.bracketed
9338             + parsers.inticks
9339             + parsers.autolink
9340             + V("InlineHtml")
9341             + V("Space") + V("Endline")
9342             + (parsers.anyescaped
9343               - (parsers.newline + parsers.rbracket + parsers.at
9344                 + parsers.semicolon + parsers.blankline^2))
9345             - (parsers.spnl * parsers.rbracket))^0)
9346
9347     local citation_headless_body
9348         = citation_headless_body_postnote
9349         * (parsers.sp * parsers.semicolon * parsers.spnlc
9350           * citation_body_chunk)^0
9351
9352     local citations
9353         = function(text_cites, raw_cites)
9354         local function normalize(str)
9355             if str == "" then
9356                 str = nil
9357             else
9358                 str = (citation_nbsps and
9359                     self.parser_functions.parse_inlines_nbsp or
9360                     self.parser_functions.parse_inlines)(str)

```



```

9361         end
9362         return str
9363     end
9364
9365     local cites = {}
9366     for i = 1,#raw_cites,4 do
9367         cites[#cites+1] = {
9368             prenote = normalize(raw_cites[i]),
9369             suppress_author = raw_cites[i+1] == "-",
9370             name = writer.identifier(raw_cites[i+2]),
9371             postnote = normalize(raw_cites[i+3]),
9372         }
9373     end
9374     return writer.citations(text_cites, cites)
9375 end
9376
9377 local TextCitations
9378     = Ct((parsers.spnlc
9379         * Cc("")
9380         * citation_name
9381         * ((parsers.spnlc
9382             * parsers.lbracket
9383             * citation_headless_body
9384             * parsers.rbracket) + Cc("")))~1)
9385 / function(raw_cites)
9386     return citations(true, raw_cites)
9387 end
9388
9389 local ParenthesizedCitations
9390     = Ct((parsers.spnlc
9391         * parsers.lbracket
9392         * citation_body
9393         * parsers.rbracket)^1)
9394 / function(raw_cites)
9395     return citations(false, raw_cites)
9396 end
9397
9398 local Citations = TextCitations + ParenthesizedCitations
9399
9400 self.insert_pattern("Inline before LinkAndEmph",
9401     Citations, "Citations")
9402
9403 self.add_special_character("@")
9404 self.add_special_character("-")
9405 end
9406 }
9407 end

```

### 3.1.6.3 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.

```
9408 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.

```
9409   local languages_json = (function()
9410     local base, prev, curr
9411     for _, pathname in ipairs{util.lookup_files(language_map, { all=true })} do
9412       local file = io.open(pathname, "r")
9413       if not file then goto continue end
9414       local input = assert(file:read("*a"))
9415       assert(file:close())
9416       local json = input:gsub('("[^\\n]-"):', '[%1]=')
9417       curr = load("_ENV = {}; return "..json")()
9418       if type(curr) == "table" then
9419         if base == nil then
9420           base = curr
9421         else
9422           setmetatable(prev, { __index = curr })
9423           end
9424         prev = curr
9425       end
9426       ::continue::
9427     end
9428     return base or {}
9429   end)()
9430
9431   return {
9432     name = "built-in content_blocks syntax extension",
9433     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.

```
9434     function self.contentblock(src,suf,type,tit)
9435       if not self.is_writing then return "" end
9436       src = src..".."..suf
9437       suf = suf:lower()
9438       if type == "onlineimage" then
9439         return {"\\markdownRendererContentBlockOnlineImage{",suf,"}",
9440               {"",self.string(src),""},
```

```

9441             {"",self.uri(src),""},
9442             {"",self.string(tit or ""),""}
9443     elseif languages_json[suf] then
9444         return {"\\markdownRendererContentBlockCode{"",suf,""},
9445             {"",self.string(languages_json[suf]),""},
9446             {"",self.string(src),""},
9447             {"",self.uri(src),""},
9448             {"",self.string(tit or ""),""}
9449     else
9450         return {"\\markdownRendererContentBlock{"",suf,""},
9451             {"",self.string(src),""},
9452             {"",self.uri(src),""},
9453             {"",self.string(tit or ""),""}
9454     end
9455 end
9456 end, extend_reader = function(self)
9457     local parsers = self.parsers
9458     local writer = self.writer
9459
9460     local contentblock_tail
9461         = parsers.optionaltitle
9462         * (parsers.newline + parsers.eof)
9463
9464     -- case insensitive online image suffix:
9465     local onlineimagesuffix
9466         = (function(...)
9467             local parser = nil
9468             for _, suffix in ipairs({...}) do
9469                 local pattern=nil
9470                 for i=1,#suffix do
9471                     local char=suffix:sub(i,i)
9472                     char = S(char:lower()..char:upper())
9473                     if pattern == nil then
9474                         pattern = char
9475                     else
9476                         pattern = pattern * char
9477                     end
9478                 end
9479                 if parser == nil then
9480                     parser = pattern
9481                 else
9482                     parser = parser + pattern
9483                 end
9484             end
9485             return parser
9486         end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
9487

```

```

9488      -- online image url for iA Writer content blocks with mandatory suffix,
9489      -- allowing nested brackets:
9490      local onlineimageurl
9491          = (parsers.less
9492              * Cs((parsers.anyescaped
9493                  - parsers.more
9494                  - parsers.spacing
9495                  - #(parsers.period
9496                      * onlineimagesuffix
9497                      * parsers.more
9498                      * contentblock_tail)))^0)
9499              * parsers.period
9500              * Cs(onlineimagesuffix)
9501              * parsers.more
9502              + (Cs((parsers.inparens
9503                  + (parsers.anyescaped
9504                      - parsers.spacing
9505                      - parsers.rparent
9506                      - #(parsers.period
9507                          * onlineimagesuffix
9508                          * contentblock_tail))))^0)
9509                  * parsers.period
9510                  * Cs(onlineimagesuffix))
9511              ) * Cc("onlineimage")
9512
9513      -- filename for iA Writer content blocks with mandatory suffix:
9514      local localfilepath
9515          = parsers.slash
9516          * Cs((parsers.anyescaped
9517              - parsers.tab
9518              - parsers.newline
9519              - #(parsers.period
9520                  * parsers.alphanumeric^1
9521                  * contentblock_tail))^1)
9522          * parsers.period
9523          * Cs(parsers.alphanumeric^1)
9524          * Cc("localfile")
9525
9526      local ContentBlock
9527          = parsers.check_trail_no_rem
9528          * (localfilepath + onlineimageurl)
9529          * contentblock_tail
9530          / writer.contentblock
9531
9532      self.insert_pattern("Block before Blockquote",
9533                          ContentBlock, "ContentBlock")
9534  end

```

```

9535 }
9536 end

```

### 3.1.6.4 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.

```

9537 M.extensions.definition_lists = function(tight_lists)
9538   return {
9539     name = "built-in definition_lists syntax extension",
9540     extend_writer = function(self)

```

Define `writer->definitionlist` as a function that will transform an input definition list to the output format, where `items` is an array of tables, each of the form `{ term = t, definitions = defs }`, where `t` is a term and `defs` is an array of definitions. `tight` specifies, whether the list is tight or not.

```

9541     local function dlistem(term, defs)
9542       local retVal = {"\\markdownRendererDlItem{",term,""}
9543       for _, def in ipairs(defs) do
9544         retVal[#retVal+1] = {"\\markdownRendererDlDefinitionBegin ",def,
9545                               "\\markdownRendererDlDefinitionEnd "}
9546       end
9547       retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
9548       return retVal
9549     end
9550
9551     function self.definitionlist(items,tight)
9552       if not self.is_writing then return "" end
9553       local buffer = {}
9554       for _,item in ipairs(items) do
9555         buffer[#buffer + 1] = dlistem(item.term, item.definitions)
9556       end
9557       if tight and tight_lists then
9558         return {"\\markdownRendererDlBeginTight\\n", buffer,
9559               "\\n\\markdownRendererDlEndTight"}
9560       else
9561         return {"\\markdownRendererDlBegin\\n", buffer,
9562               "\\n\\markdownRendererDlEnd"}
9563       end
9564     end
9565   end, extend_reader = function(self)
9566     local parsers = self.parsers
9567     local writer = self.writer
9568
9569     local defstartchar = S("~:")
9570

```

```

9571     local defstart  = parsers.check_trail_length(0) * defstartchar * #parsers.spaci
9572                               * (parsers.tab + parsers.space^-
3)
9573                               + parsers.check_trail_length(1) * defstartchar * #parsers.spaci
9574                               * (parsers.tab + parsers.space^-
2)
9575                               + parsers.check_trail_length(2) * defstartchar * #parsers.spaci
9576                               * (parsers.tab + parsers.space^-
1)
9577                               + parsers.check_trail_length(3) * defstartchar * #parsers.spaci
9578
9579     local indented_line = (parsers.check_minimal_indent / "") * parsers.check_code_
9580
9581     local blank = parsers.check_minimal_blank_indent_and_any_trail * parsers.option
9582
9583     local dlchunk = Cs(parsers.line * (indented_line - blank)^0)
9584
9585     local indented_blocks = function(bl)
9586         return Cs( bl
9587             * (blank^1 * (parsers.check_minimal_indent / "")
9588             * parsers.check_code_trail * -parsers.blankline * bl)^0
9589             * (blank^1 + parsers.eof))
9590     end
9591
9592     local function definition_list_item(term, defs, _)
9593         return { term = self.parser_functions.parse_inlines(term),
9594             definitions = defs }
9595     end
9596
9597     local DefinitionListItemLoose
9598         = C(parsers.line) * blank^0
9599         * Ct((parsers.check_minimal_indent * (defstart
9600             * indented_blocks(dlchunk)
9601             / self.parser_functions.parse_blocks_nested))^1)
9602         * Cc(false) / definition_list_item
9603
9604     local DefinitionListItemTight
9605         = C(parsers.line)
9606         * Ct((parsers.check_minimal_indent * (defstart * dlchunk
9607             / self.parser_functions.parse_blocks_nested))^1)
9608         * Cc(true) / definition_list_item
9609
9610     local DefinitionList
9611         = ( Ct(DefinitionListItemLoose^1) * Cc(false)
9612         + Ct(DefinitionListItemTight^1)
9613         * (blank^0
9614         * -DefinitionListItemLoose * Cc(true))

```

```

9615             ) / writer.definitionlist
9616
9617         self.insert_pattern("Block after Heading",
9618                             DefinitionList, "DefinitionList")
9619     end
9620 }
9621 end

```

### 3.1.6.5 Fancy Lists

The `extensions.fancy_lists` function implements the Pandoc fancy list syntax extension.

```

9622 M.extensions.fancy_lists = function()
9623     return {
9624         name = "built-in fancy_lists syntax extension",
9625         extend_writer = function(self)
9626             local options = self.options
9627

```

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.

```

9628         function self.fancylist(items,tight,startnum,numstyle,numdelim)
9629             if not self.is_writing then return "" end
9630             local buffer = {}
9631             local num = startnum
9632             for _,item in ipairs(items) do
9633                 if item ~= "" then

```

```

9634         buffer[#buffer + 1] = self.fancyitem(item,num)
9635     end
9636     if num ~= nil and item ~= "" then
9637         num = num + 1
9638     end
9639 end
9640 local contents = util.intersperse(buffer,"\n")
9641 if tight and options.tightLists then
9642     return {"\\markdownRendererFancy0lBeginTight{",
9643             numstyle,"}{",numdelim,"}",contents,
9644             "\n\\markdownRendererFancy0lEndTight "}
9645 else
9646     return {"\\markdownRendererFancy0lBegin{",
9647             numstyle,"}{",numdelim,"}",contents,
9648             "\n\\markdownRendererFancy0lEnd "}
9649 end
9650 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.

```

9651     function self.fancyitem(s,num)
9652         if num ~= nil then
9653             return {"\\markdownRendererFancy0lItemWithNumber{",num,"}",s,
9654                     "\\markdownRendererFancy0lItemEnd "}
9655         else
9656             return {"\\markdownRendererFancy0lItem ",s,"\\markdownRendererFancy0lItemEnd "}
9657         end
9658     end
9659 end, extend_reader = function(self)
9660     local parsers = self.parsers
9661     local options = self.options
9662     local writer = self.writer
9663
9664     local function combine_markers_and_delims(markers, delims)
9665         local markers_table = {}
9666         for _,marker in ipairs(markers) do
9667             local start_marker
9668             local continuation_marker
9669             if type(marker) == "table" then
9670                 start_marker = marker[1]
9671                 continuation_marker = marker[2]
9672             else
9673                 start_marker = marker
9674                 continuation_marker = marker
9675             end
9676             for _,delim in ipairs(delims) do

```



```

9677         table.insert(markers_table, {start_marker, continuation_marker, delim})
9678     end
9679 end
9680 return markers_table
9681 end
9682
9683 local function join_table_with_func(func, markers_table)
9684     local pattern = func(table.unpack(markers_table[1]))
9685     for i = 2, #markers_table do
9686         pattern = pattern + func(table.unpack(markers_table[i]))
9687     end
9688     return pattern
9689 end
9690
9691 local lowercase_letter_marker = R("az")
9692 local uppercase_letter_marker = R("AZ")
9693
9694 local roman_marker = function(chars)
9695     local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])
9696     local l, x, v, i = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])
9697     return m^-3
9698         * (c*m + c*d + d^-1 * c^-3)
9699         * (x*c + x*l + l^-1 * x^-3)
9700         * (i*x + i*v + v^-1 * i^-3)
9701 end
9702
9703 local lowercase_roman_marker = roman_marker({"m", "d", "c", "l", "x", "v", "i"})
9704 local uppercase_roman_marker = roman_marker({"M", "D", "C", "L", "X", "V", "I"})
9705
9706 local lowercase_opening_roman_marker = P("i")
9707 local uppercase_opening_roman_marker = P("I")
9708
9709 local digit_marker = parsers.dig * parsers.dig^-8
9710
9711 local markers = {
9712     {lowercase_opening_roman_marker, lowercase_roman_marker},
9713     {uppercase_opening_roman_marker, uppercase_roman_marker},
9714     lowercase_letter_marker,
9715     uppercase_letter_marker,
9716     lowercase_roman_marker,
9717     uppercase_roman_marker,
9718     digit_marker
9719 }
9720
9721 local delims = {
9722     parsers.period,
9723     parsers.rparent

```

```

9724     }
9725
9726     local markers_table = combine_markers_and_delims(markers, delims)
9727
9728     local function enumerator(start_marker, _, delimiter_type, interrupting)
9729         local delimiter_range
9730         local allowed_end
9731         if interrupting then
9732             delimiter_range = P("1")
9733             allowed_end = C(parsers.spacechar~1) * #parsers.linechar
9734         else
9735             delimiter_range = start_marker
9736             allowed_end = C(parsers.spacechar~1) + #(parsers.newline + parsers.eof)
9737         end
9738
9739         return parsers.check_trail
9740             * Ct(C(delimiter_range) * C(delimiter_type))
9741             * allowed_end
9742     end
9743
9744     local starter = join_table_with_func(enumerator, markers_table)
9745
9746     local TightListItem = function(starter)
9747         return parsers.add_indent(starter, "li")
9748             * parsers.indented_content_tight
9749     end
9750
9751     local LooseListItem = function(starter)
9752         return parsers.add_indent(starter, "li")
9753             * parsers.indented_content_loose
9754             * remove_indent("li")
9755     end
9756
9757     local function roman2number(roman)
9758         local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100, ["L"] = 50, ["X"] =
9759         local numeral = 0
9760
9761         local i = 1
9762         local len = string.len(roman)
9763         while i < len do
9764             local z1, z2 = romans[ string.sub(roman, i, i) ], romans[ string.sub(roman,
9765             if z1 < z2 then
9766                 numeral = numeral + (z2 - z1)
9767                 i = i + 2
9768             else
9769                 numeral = numeral + z1
9770                 i = i + 1

```

```

9771         end
9772     end
9773     if i <= len then numeral = numeral + romans[ string.sub(roman,i,i) ] end
9774     return numeral
9775 end
9776
9777 local function sniffstyle(numstr, delimend)
9778     local numdelim
9779     if delimend == ")" then
9780         numdelim = "OneParen"
9781     elseif delimend == "." then
9782         numdelim = "Period"
9783     else
9784         numdelim = "Default"
9785     end
9786
9787     local num
9788     num = numstr:match("^([I])$")
9789     if num then
9790         return roman2number(num), "UpperRoman", numdelim
9791     end
9792     num = numstr:match("^([i])$")
9793     if num then
9794         return roman2number(string.upper(num)), "LowerRoman", numdelim
9795     end
9796     num = numstr:match("^([A-Z])$")
9797     if num then
9798         return string.byte(num) - string.byte("A") + 1, "UpperAlpha", numdelim
9799     end
9800     num = numstr:match("^([a-z])$")
9801     if num then
9802         return string.byte(num) - string.byte("a") + 1, "LowerAlpha", numdelim
9803     end
9804     num = numstr:match("^([IVXLCDM]+)")
9805     if num then
9806         return roman2number(num), "UpperRoman", numdelim
9807     end
9808     num = numstr:match("^([ivxlcdm]+)")
9809     if num then
9810         return roman2number(string.upper(num)), "LowerRoman", numdelim
9811     end
9812     return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
9813 end
9814
9815 local function fancylist(items,tight,start)
9816     local startnum, numstyle, numdelim = sniffstyle(start[2][1], start[2][2])
9817     return writer.fancylist(items,tight,

```

```

9818             options.startNumber and startnum or 1,
9819             numstyle or "Decimal",
9820             numdelim or "Default")
9821     end
9822
9823     local FancyListOfType = function(start_marker, continuation_marker, delimiter_t
9824         local enumerator_start = enumerator(start_marker, continuation_marker, delimi
9825         local enumerator_cont = enumerator(continuation_marker, continuation_marker,
9826         return Cg(enumerator_start, "listtype")
9827             * (Ct( TightListItem(Cb("listtype"))
9828                 * ((parsers.check_minimal_indent / "") * TightListItem(enumerator_co
9829             * Cc(true)
9830             * -#((parsers.conditionally_indented_blankline^0 / "")
9831                 * parsers.check_minimal_indent * enumerator_cont)
9832             + Ct( LooseListItem(Cb("listtype"))
9833                 * ((parsers.conditionally_indented_blankline^0 / "")
9834                 * (parsers.check_minimal_indent / "") * LooseListItem(enumerator_c
9835             * Cc(false)
9836             ) * Ct(Cb("listtype")) / fancylist
9837     end
9838
9839     local FancyList = join_table_with_func(FancyListOfType, markers_table)
9840
9841     local Endline    = parsers.newline
9842                     * (parsers.check_minimal_indent
9843                       * -parsers.EndlineExceptions
9844                       + parsers.check_optional_indent
9845                       * -parsers.EndlineExceptions
9846                       * -starter)
9847                     * parsers.spacechar^0
9848                     / writer.soft_line_break
9849
9850     self.update_rule("OrderedList", FancyList)
9851     self.update_rule("Endline", Endline)
9852 end
9853 }
9854 end

```

### 3.1.6.6 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.

```

9855 M.extensions.fenced_code = function(blank_before_code_fence,
9856                                     allow_attributes,
9857                                     allow_raw_blocks)
9858   return {
9859     name = "built-in fenced_code syntax extension",
9860     extend_writer = function(self)
9861       local options = self.options
9862

```

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.

```

9863     function self.fencedCode(s, i, attr)
9864       if not self.is_writing then return "" end
9865       s = s:gsub("\n$", "")
9866       local buf = {}
9867       if attr ~= nil then
9868         table.insert(buf, {"\\markdownRendererFencedCodeAttributeContextBegin",
9869                             self.attributes(attr)})
9870       end
9871       local name = util.cache_verbatim(options.cacheDir, s)
9872       table.insert(buf, {"\\markdownRendererInputFencedCode{",
9873                           name,"}{" ,self.string(i),"}{" ,self.infostring(i),"}")
9874       if attr ~= nil then
9875         table.insert(buf, "\\markdownRendererFencedCodeAttributeContextEnd")
9876       end
9877       return buf
9878     end
9879

```

Define `writer->rawBlock` as a function that will transform an input raw block `s` with the raw attribute `attr` to the output format.

```

9880     if allow_raw_blocks then
9881       function self.rawBlock(s, attr)
9882         if not self.is_writing then return "" end
9883         s = s:gsub("\n$", "")
9884         local name = util.cache_verbatim(options.cacheDir, s)
9885         return {"\\markdownRendererInputRawBlock{",
9886                 name,"}{" , self.string(attr),"}")
9887       end
9888     end
9889   end, extend_reader = function(self)
9890     local parsers = self.parsers
9891     local writer = self.writer
9892
9893     local function captures_geq_length(_,i,a,b)

```

```

9894         return #a >= #b and i
9895     end
9896
9897     local function strip_enclosing_whitespace(str)
9898         return str:gsub("^%s*(.)%s*$", "%1")
9899     end
9900
9901     local tilde_infostring = Cs(Cs((V("HtmlEntity")
9902         + parsers.anyescaped
9903         - parsers.newline)^0)
9904         / strip_enclosing_whitespace)
9905
9906     local backtick_infostring = Cs(Cs((V("HtmlEntity")
9907         + (-#(parsers.backslash * parsers.backtick) *
9908         - parsers.newline
9909         - parsers.backtick)^0)
9910         / strip_enclosing_whitespace)
9911
9912     local fenceindent
9913
9914     local function has_trail(indent_table)
9915         return indent_table ~= nil and
9916             indent_table.trail ~= nil and
9917             next(indent_table.trail) ~= nil
9918     end
9919
9920     local function has_indents(indent_table)
9921         return indent_table ~= nil and
9922             indent_table.indents ~= nil and
9923             next(indent_table.indents) ~= nil
9924     end
9925
9926     local function get_last_indent_name(indent_table)
9927         if has_indents(indent_table) then
9928             return indent_table.indents[#indent_table.indents].name
9929         end
9930     end
9931
9932     local function count_fenced_start_indent(_, _, indent_table, trail)
9933         local last_indent_name = get_last_indent_name(indent_table)
9934         fenceindent = 0
9935         if last_indent_name ~= "li" then
9936             fenceindent = #trail
9937         end
9938         return true
9939     end
9940

```

```

9941     local fencehead      = function(char, infostring)
9942         return            Cmt(Cb("indent_info") * parsers.check_trail, count_fence
9943                             * Cg(char^3, "fencelength")
9944                             * parsers.optionalspace
9945                             * infostring
9946                             * (parsers.newline + parsers.eof)
9947     end
9948
9949     local fencetail      = function(char)
9950         return            parsers.check_trail_no_rem
9951                             * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
9952                             * parsers.optionalspace * (parsers.newline + parsers.eof)
9953                             + parsers.eof
9954     end
9955
9956     local function process_fenced_line(s, i, indent_table, line_content, is_blank)
9957         local remainder = ""
9958         if has_trail(indent_table) then
9959             remainder = indent_table.trail.internal_remainder
9960         end
9961
9962         if is_blank and get_last_indent_name(indent_table) == "li" then
9963             remainder = ""
9964         end
9965
9966         local str = remainder .. line_content
9967         local index = 1
9968         local remaining = fenceindent
9969
9970         while true do
9971             local c = str:sub(index, index)
9972             if c == " " and remaining > 0 then
9973                 remaining = remaining - 1
9974                 index = index + 1
9975             elseif c == "\t" and remaining > 3 then
9976                 remaining = remaining - 4
9977                 index = index + 1
9978             else
9979                 break
9980             end
9981         end
9982
9983         return true, str:sub(index)
9984     end
9985
9986     local fencedline = function(char)
9987         return Cmt(Cb("indent_info") * C(parsers.line - fencetail(char)) * Cc(false),

```

```

9988     end
9989
9990     local blankfencedline = Cmt(Cb("indent_info") * C(parsers.blankline) * Cc(true))
9991
9992     local TildeFencedCode
9993         = fencehead(parsers.tilde, tilde_infostring)
9994         * Cs(((parsers.check_minimal_blank_indent / "") * blankfencedline
9995             + (parsers.check_minimal_indent / "") * fencedline(parsers.tilde))
9996         * ((parsers.check_minimal_indent / "") * fencetail(parsers.tilde) + pars
9997
9998     local BacktickFencedCode
9999         = fencehead(parsers.backtick, backtick_infostring)
10000         * Cs(((parsers.check_minimal_blank_indent / "") * blankfencedline
10001             + (parsers.check_minimal_indent / "") * fencedline(parsers.backtick)
10002         * ((parsers.check_minimal_indent / "") * fencetail(parsers.backtick) + p
10003
10004     local infostring_with_attributes
10005         = Ct(C((parsers.linechar
10006             - ( parsers.optionalspace
10007                 * parsers.attributes))^0)
10008             * parsers.optionalspace
10009             * Ct(parsers.attributes))
10010
10011     local FencedCode
10012         = ((TildeFencedCode + BacktickFencedCode)
10013         / function(infostring, code)
10014             local expanded_code = self.expandtabs(code)
10015
10016             if allow_raw_blocks then
10017                 local raw_attr = lpeg.match(parsers.raw_attribute,
10018                                         infostring)
10019
10020                 if raw_attr then
10021                     return writer.rawBlock(expanded_code, raw_attr)
10022                 end
10023             end
10024
10025             local attr = nil
10026             if allow_attributes then
10027                 local match = lpeg.match(infostring_with_attributes,
10028                                         infostring)
10029
10030                 if match then
10031                     infostring, attr = table.unpack(match)
10032                 end
10033             end
10034             return writer.fencedCode(expanded_code, infostring, attr)
10035         end)

```



```

10035     self.insert_pattern("Block after Verbatim",
10036                           FencedCode, "FencedCode")
10037
10038     local fencestart
10039     if blank_before_code_fence then
10040         fencestart = parsers.fail
10041     else
10042         fencestart = fencehead(parsers.backtick, backtick_infostring)
10043                     + fencehead(parsers.tilde, tilde_infostring)
10044     end
10045
10046     self.update_rule("EndlineExceptions", function(previous_pattern)
10047         if previous_pattern == nil then
10048             previous_pattern = parsers.EndlineExceptions
10049         end
10050         return previous_pattern + fencestart
10051     end)
10052
10053     self.add_special_character("`")
10054     self.add_special_character("~")
10055 end
10056 }
10057 end

```

### 3.1.6.7 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.

```

10058 M.extensions.fenced_divs = function(blank_before_div_fence)
10059     return {
10060         name = "built-in fenced_divs syntax extension",
10061         extend_writer = function(self)

```

Define `writer->div_begin` as a function that will transform the beginning of an input fenced div with with attributes `attributes` to the output format.

```

10062         function self.div_begin(attributes)
10063             local start_output = {"\\markdownRendererFencedDivAttributeContextBegin\\n",
10064                                   self.attributes(attributes)}
10065             local end_output = {"\\markdownRendererFencedDivAttributeContextEnd "}
10066             return self.push_attributes("div", attributes, start_output, end_output)
10067         end

```

Define `writer->div_end` as a function that will produce the end of a fenced div in the output format.

```

10068         function self.div_end()

```

```

10069         return self.pop_attributes("div")
10070     end
10071 end, extend_reader = function(self)
10072     local parsers = self.parsers
10073     local writer = self.writer

```

Define basic patterns for matching the opening and the closing tag of a div.

```

10074     local fenced_div_infostring
10075         = C((parsers.linechar
10076             - ( parsers.spacechar^1
10077               * parsers.colon^1))^1)
10078
10079     local fenced_div_begin = parsers.nonindentspace
10080         * parsers.colon^3
10081         * parsers.optionalspace
10082         * fenced_div_infostring
10083         * ( parsers.spacechar^1
10084           * parsers.colon^1)^0
10085         * parsers.optionalspace
10086         * (parsers.newline + parsers.eof)
10087
10088     local fenced_div_end = parsers.nonindentspace
10089         * parsers.colon^3
10090         * parsers.optionalspace
10091         * (parsers.newline + parsers.eof)

```

Initialize a named group named `div_level` for tracking how deep we are nested in divs.

```

10092     self.initialize_named_group("div_level", "0")
10093
10094     local function increment_div_level(increment)
10095         local function update_div_level(s, i, current_level) -- luacheck: ignore s i
10096             current_level = tonumber(current_level)
10097             local next_level = tostring(current_level + increment)
10098             return true, next_level
10099         end
10100
10101         return Cg( Cmt(Cb("div_level"), update_div_level)
10102             , "div_level")
10103     end
10104
10105     local non_fenced_div_block = parsers.check_minimal_indent * V("Block")
10106         - parsers.check_minimal_indent_and_trail * fenced_div_block
10107
10108     local non_fenced_div_paragraph = parsers.check_minimal_indent * V("Paragraph")
10109         - parsers.check_minimal_indent_and_trail * fenced_div_paragraph
10110
10111     local blank = parsers.minimally_indented_blank

```

```

10112
10113     local block_separated = parsers.block_sep_group(blank)
10114                             * non_fenced_div_block
10115
10116     local loop_body_pair = parsers.create_loop_body_pair(block_separated,
10117                                                             non_fenced_div_paragraph,
10118                                                             parsers.block_sep_group(t
10119                                                             parsers.par_sep_group(bla
10120
10121     local content_loop = ( non_fenced_div_block
10122                             * loop_body_pair.block^0
10123                             + non_fenced_div_paragraph
10124                             * block_separated
10125                             * loop_body_pair.block^0
10126                             + non_fenced_div_paragraph
10127                             * loop_body_pair.par^0)
10128                             * blank^0
10129
10130     local FencedDiv = fenced_div_begin
10131                       / function (infostring)
10132                           local attr = lpeg.match(Ct(parsers.attributes), infostring)
10133                           if attr == nil then
10134                               attr = {"." .. infostring}
10135                           end
10136                           return attr
10137                       end
10138                       / writer.div_begin
10139                       * increment_div_level(1)
10140                       * parsers.skipblanklines
10141                       * Ct(content_loop)
10142                       * parsers.minimally_indented_blank^0
10143                       * parsers.check_minimal_indent_and_trail * fenced_div_end * inc
10144
10145     1)
10146
10147     * (Cc("") / writer.div_end)
10148
10149     self.insert_pattern("Block after Verbatim",
10150                       FencedDiv, "FencedDiv")
10151
10152     self.add_special_character(":")
10153
10154

```

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.

```

10151     local function check_div_level(s, i, current_level) -- luacheck: ignore s i
10152         current_level = tonumber(current_level)
10153         return current_level > 0

```

```

10154     end
10155
10156     local is_inside_div = Cmt(Cb("div_level"), check_div_level)
10157     local fencestart = is_inside_div * fenced_div_end
10158
10159     if not blank_before_div_fence then
10160         self.update_rule("EndlineExceptions", function(previous_pattern)
10161             if previous_pattern == nil then
10162                 previous_pattern = parsers.EndlineExceptions
10163             end
10164             return previous_pattern + fencestart
10165         end)
10166     end
10167 end
10168 }
10169 end

```

### 3.1.6.8 Header Attributes

The `extensions.header_attributes` function implements the Pandoc header attribute syntax extension.

```

10170 M.extensions.header_attributes = function()
10171     return {
10172         name = "built-in header_attributes syntax extension",
10173         extend_writer = function()
10174             end, extend_reader = function(self)
10175                 local parsers = self.parsers
10176                 local writer = self.writer
10177
10178                 local function strip_atx_end(s)
10179                     return s:gsub("%s+##%s*$", "")
10180                 end
10181
10182                 local AtxHeading = Cg(parsers.heading_start, "level")
10183                     * parsers.optionalspace
10184                     * (C(((parsers.linechar
10185                         - (parsers.attributes
10186                             * parsers.optionalspace
10187                             * parsers.newline))
10188                         * (parsers.linechar
10189                             - parsers.lbrace)^0)^1)
10190                     / strip_atx_end
10191                     / parsers.parse_heading_text)
10192                 * Cg(Ct(parsers.newline
10193                     + (parsers.attributes
10194                         * parsers.optionalspace
10195                         * parsers.newline)), "attributes")

```

```

10196             * Cb("level")
10197             * Cb("attributes")
10198             / writer.heading
10199
10200     local function strip_trailing_spaces(s)
10201         return s:gsub("%s*$", "")
10202     end
10203
10204     local heading_line = (parsers.linechar
10205                          - (parsers.attributes
10206                            * parsers.optionalspace
10207                            * parsers.newline))^1
10208                          - parsers.thematic_break_lines
10209
10210     local heading_text = heading_line
10211                       * ((V("Endline") / "\n") * (heading_line - parsers.heading_
10212                                                       * parsers.newline^-1
10213
10214     local SettextHeading = parsers.freeze_trail * parsers.check_trail_no_rem
10215                          * #(heading_text
10216                             * (parsers.attributes
10217                               * parsers.optionalspace
10218                               * parsers.newline)^-1
10219                               * parsers.check_minimal_indent * parsers.check_trail *
10220                               * Cs(heading_text) / strip_trailing_spaces
10221                               / parsers.parse_heading_text
10222                               * Cg(Ct((parsers.attributes
10223                                     * parsers.optionalspace
10224                                     * parsers.newline)^-1), "attributes")
10225                               * parsers.check_minimal_indent_and_trail * parsers.heading_
10226                               * Cb("attributes")
10227                               * parsers.newline
10228                               * parsers.unfreeze_trail
10229                               / writer.heading
10230
10231     local Heading = AtxHeading + SettextHeading
10232     self.update_rule("Heading", Heading)
10233 end
10234 }
10235 end

```

### 3.1.6.9 Inline Code Attributes

The `extensions.inline_code_attributes` function implements the Pandoc inline code attribute syntax extension.

```

10236 M.extensions.inline_code_attributes = function()
10237     return {

```

```

10238     name = "built-in inline_code_attributes syntax extension",
10239     extend_writer = function()
10240 end, extend_reader = function(self)
10241     local writer = self.writer
10242
10243     local CodeWithAttributes = parsers.inticks
10244                             * Ct(parsers.attributes)
10245                             / writer.code
10246
10247     self.insert_pattern("Inline before Code",
10248                       CodeWithAttributes,
10249                       "CodeWithAttributes")
10250 end
10251 }
10252 end

```

### 3.1.6.10 Line Blocks

The `extensions.line_blocks` function implements the Pandoc line block syntax extension.

```

10253 M.extensions.line_blocks = function()
10254     return {
10255         name = "built-in line_blocks syntax extension",
10256         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.

```

10257         function self.lineblock(lines)
10258             if not self.is_writing then return "" end
10259             local buffer = {}
10260             for i = 1, #lines - 1 do
10261                 buffer[#buffer + 1] = { lines[i], self.hard_line_break }
10262             end
10263             buffer[#buffer + 1] = lines[#lines]
10264
10265             return {"\\markdownRendererLineBlockBegin\\n"
10266                   ,buffer,
10267                   "\\n\\markdownRendererLineBlockEnd "}
10268         end
10269     end, extend_reader = function(self)
10270         local parsers = self.parsers
10271         local writer = self.writer
10272
10273         local LineBlock = Ct(
10274             (Cs(
10275                 ( (parsers.pipe * parsers.space)/""
10276                 * ((parsers.space)/entities.char_entity("nbsp"))^0
10277                 * parsers.linechar^0 * (parsers.newline/""))

```

```

10278             * (-parsers.pipe
10279             * (parsers.space^1/" ")
10280             * parsers.linechar^1
10281             * (parsers.newline/"")
10282             )^0
10283             * (parsers.blankline/"")^0
10284             ) / self.parser_functions.parse_inlines)^1) / writer.lineblo
10285
10286         self.insert_pattern("Block after Blockquote",
10287                             LineBlock, "LineBlock")
10288     end
10289 }
10290 end

```

### 3.1.6.11 Marked text

The `extensions.mark` function implements the Pandoc mark syntax extension.

```

10291 M.extensions.mark = function()
10292   return {
10293     name = "built-in mark syntax extension",
10294     extend_writer = function(self)

```

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

```

10295     function self.mark(s)
10296       if self.flatten_inlines then return s end
10297       return {"\\markdownRenderMark{", s, "}"}
10298     end
10299   end, extend_reader = function(self)
10300     local parsers = self.parsers
10301     local writer = self.writer
10302
10303     local doubleequals = P("==")
10304
10305     local Mark = parsers.between(V("Inline"), doubleequals, doubleequals)
10306       / function (inlines) return writer.mark(inlines) end
10307
10308     self.add_special_character("=")
10309     self.insert_pattern("Inline before LinkAndEmph",
10310                         Mark, "Mark")
10311   end
10312 }
10313 end

```

### 3.1.6.12 Link Attributes

The `extensions.link_attributes` function implements the Pandoc link attribute syntax extension.

```

10314 M.extensions.link_attributes = function()
10315     return {
10316         name = "built-in link_attributes syntax extension",
10317         extend_writer = function()
10318         end, extend_reader = function(self)
10319             local parsers = self.parsers
10320             local options = self.options
10321

```

The following patterns define link reference definitions with attributes.

```

10322     local define_reference_parser = (parsers.check_trail / "") * parsers.link_label
10323                                     * parsers.spnlc * parsers.url
10324                                     * ( parsers.spnlc_sep * parsers.title * (parsers.
10325                                     * parsers.only_blank
10326                                     + parsers.spnlc_sep * parsers.title * parsers.c
10327                                     + Cc("") * (parsers.spnlc * Ct(parsers.attribut
10328                                     + Cc("") * parsers.only_blank)
10329
10330     local ReferenceWithAttributes = define_reference_parser
10331                                     / self.register_link
10332
10333     self.update_rule("Reference", ReferenceWithAttributes)
10334

```

The following patterns define direct and indirect links with attributes.

```

10335
10336     local LinkWithAttributesAndEmph = Ct(parsers.link_and_emph_table * Cg(Cc(true),
10337                                     / self.defer_link_and_emphasis_processing
10338
10339     self.update_rule("LinkAndEmph", LinkWithAttributesAndEmph)
10340

```

The following patterns define autolinks with attributes.

```

10341     local AutoLinkUrlWithAttributes
10342         = parsers.auto_link_url
10343         * Ct(parsers.attributes)
10344         / self.auto_link_url
10345
10346     self.insert_pattern("Inline before AutoLinkUrl",
10347                         AutoLinkUrlWithAttributes,
10348                         "AutoLinkUrlWithAttributes")
10349
10350     local AutoLinkEmailWithAttributes
10351         = parsers.auto_link_email
10352         * Ct(parsers.attributes)
10353         / self.auto_link_email
10354
10355     self.insert_pattern("Inline before AutoLinkEmail",

```



```

10356             AutoLinkEmailWithAttributes,
10357             "AutoLinkEmailWithAttributes")
10358
10359         if options.relativeReferences then
10360
10361             local AutoLinkRelativeReferenceWithAttributes
10362                 = parsers.auto_link_relative_reference
10363                 * Ct(parsers.attributes)
10364                 / self.auto_link_url
10365
10366             self.insert_pattern(
10367                 "Inline before AutoLinkRelativeReference",
10368                 AutoLinkRelativeReferenceWithAttributes,
10369                 "AutoLinkRelativeReferenceWithAttributes")
10370
10371         end
10372
10373     end
10374 }
10375 end

```

### 3.1.6.13 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.

```

10376 M.extensions.notes = function(notes, inline_notes)
10377     assert(notes or inline_notes)
10378     return {
10379         name = "built-in notes syntax extension",
10380         extend_writer = function(self)

```

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

```

10381         function self.note(s)
10382             if self.flatten_inlines then return "" end
10383             return {"\\markdownRendererNote{",s,""}
10384         end
10385     end, extend_reader = function(self)
10386         local parsers = self.parsers
10387         local writer = self.writer
10388
10389         if inline_notes then
10390             local InlineNote
10391                 = parsers.circumflex
10392                 * (parsers.link_label / self.parser_functions.parse_inlines_no_in

```

```

10393             / writer.note
10394
10395         self.insert_pattern("Inline after LinkAndEmph",
10396                             InlineNote, "InlineNote")
10397     end
10398     if notes then
10399         local function strip_first_char(s)
10400             return s:sub(2)
10401         end
10402
10403         local RawNoteRef
10404             = #(parsers.lbracket * parsers.circumflex)
10405             * parsers.link_label / strip_first_char
10406
10407         local rawnotes = {}
10408
10409         -- like indirect_link
10410         local function lookup_note(ref)
10411             return writer.defer_call(function()
10412                 local found = rawnotes[self.normalize_tag(ref)]
10413                 if found then
10414                     return writer.note(
10415                         self.parser_functions.parse_blocks_nested(found))
10416                 else
10417                     return {"[",
10418                         self.parser_functions.parse_inlines("^" .. ref), "]" }
10419                 end
10420             end)
10421         end
10422
10423         local function register_note(ref, rawnote)
10424             local normalized_tag = self.normalize_tag(ref)
10425             if rawnotes[normalized_tag] == nil then
10426                 rawnotes[normalized_tag] = rawnote
10427             end
10428             return ""
10429         end
10430
10431         local NoteRef = RawNoteRef / lookup_note
10432
10433         local optionally_indented_line = parsers.check_optional_indent_and_any_trail
10434
10435         local blank = parsers.check_optional_blank_indent_and_any_trail * parsers.opt
10436
10437         local chunk = Cs(parsers.line * (optionally_indented_line - blank)^0)
10438
10439         local indented_blocks = function(bl)

```

```

10440         return Cs( bl
10441             * (blank^1 * (parsers.check_optional_indent / ""))
10442             * parsers.check_code_trail * -parsers.blankline * bl)^0)
10443     end
10444
10445     local NoteBlock
10446         = parsers.check_trail_no_rem * RawNoteRef * parsers.colon
10447         * parsers.spnlc * indented_blocks(chunk)
10448         / register_note
10449
10450     local Reference = NoteBlock + parsers.Reference
10451
10452     self.update_rule("Reference", Reference)
10453     self.insert_pattern("Inline before LinkAndEmph",
10454         NoteRef, "NoteRef")
10455 end
10456
10457 self.add_special_character("^")
10458 end
10459 }
10460 end

```

### 3.1.6.14 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.

```

10461 M.extensions.pipe_tables = function(table_captions, table_attributes)
10462
10463     local function make_pipe_table_rectangular(rows)
10464         local num_columns = #rows[2]
10465         local rectangular_rows = {}
10466         for i = 1, #rows do
10467             local row = rows[i]
10468             local rectangular_row = {}
10469             for j = 1, num_columns do
10470                 rectangular_row[j] = row[j] or ""
10471             end
10472             table.insert(rectangular_rows, rectangular_row)
10473         end
10474         return rectangular_rows
10475     end
10476
10477     local function pipe_table_row(allow_empty_first_column

```

```

10478             , nonempty_column
10479             , column_separator
10480             , column)
10481     local row_beginning
10482     if allow_empty_first_column then
10483         row_beginning = -- empty first column
10484             #(parsers.spacechar^4
10485             * column_separator)
10486             * parsers.optionalspace
10487             * column
10488             * parsers.optionalspace
10489         -- non-empty first column
10490         + parsers.nonindentspace
10491         * nonempty_column^-1
10492         * parsers.optionalspace
10493     else
10494         row_beginning = parsers.nonindentspace
10495             * nonempty_column^-1
10496             * parsers.optionalspace
10497     end
10498
10499     return Ct(row_beginning
10500         * (-- single column with no leading pipes
10501             #(column_separator
10502             * parsers.optionalspace
10503             * parsers.newline)
10504             * column_separator
10505             * parsers.optionalspace
10506             -- single column with leading pipes or
10507             -- more than a single column
10508             + (column_separator
10509             * parsers.optionalspace
10510             * column
10511             * parsers.optionalspace)^1
10512             * (column_separator
10513             * parsers.optionalspace)^-1))
10514 end
10515
10516 return {
10517     name = "built-in pipe_tables syntax extension",
10518     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.

```

10519         function self.table(rows, caption, attributes)
10520             if not self.is_writing then return "" end

```

```

10521     local buffer = {}
10522     if attributes ~= nil then
10523         table.insert(buffer,
10524             "\\markdownRendererTableAttributeContextBegin\n")
10525         table.insert(buffer, self.attributes(attributes))
10526     end
10527     table.insert(buffer,
10528         {"\\markdownRendererTable{",
10529             caption or "", "}{" , #rows - 1, "}{" ,
10530             #rows[1], "}"})
10531     local temp = rows[2] -- put alignments on the first row
10532     rows[2] = rows[1]
10533     rows[1] = temp
10534     for i, row in ipairs(rows) do
10535         table.insert(buffer, "{")
10536         for _, column in ipairs(row) do
10537             if i > 1 then -- do not use braces for alignments
10538                 table.insert(buffer, "{")
10539             end
10540             table.insert(buffer, column)
10541             if i > 1 then
10542                 table.insert(buffer, "}")
10543             end
10544         end
10545         table.insert(buffer, "}")
10546     end
10547     if attributes ~= nil then
10548         table.insert(buffer,
10549             "\\markdownRendererTableAttributeContextEnd{")
10550     end
10551     return buffer
10552 end
10553 end, extend_reader = function(self)
10554     local parsers = self.parsers
10555     local writer = self.writer
10556
10557     local table_hline_separator = parsers.pipe + parsers.plus
10558
10559     local table_hline_column = (parsers.dash
10560         - #(parsers.dash
10561             * (parsers.spacechar
10562                 + table_hline_separator
10563                 + parsers.newline)))^1
10564         * (parsers.colon * Cc("r")
10565             + parsers.dash * Cc("d"))
10566         + parsers.colon
10567         * (parsers.dash

```

```

10568         - #(parsers.dash
10569           * (parsers.spacechar
10570             + table_hline_separator
10571             + parsers.newline)))^1
10572         * (parsers.colon * Cc("c")
10573           + parsers.dash * Cc("l"))
10574
10575     local table_hline = pipe_table_row(false
10576                                       , table_hline_column
10577                                       , table_hline_separator
10578                                       , table_hline_column)
10579
10580     local table_caption_beginning = (parsers.check_minimal_blank_indent_and_any_tra
10581                                     * parsers.optionalspace * parsers.newline)^0
10582                                     * parsers.check_minimal_indent_and_trail
10583                                     * (P("Table")^-1 * parsers.colon)
10584                                     * parsers.optionalspace
10585
10586     local function strip_trailing_spaces(s)
10587         return s:gsub("%s*$","")
10588     end
10589
10590     local table_row = pipe_table_row(true
10591                                     , (C((parsers.linechar - parsers.pipe)^1)
10592                                       / strip_trailing_spaces
10593                                       / self.parser_functions.parse_inlines)
10594                                     , parsers.pipe
10595                                     , (C((parsers.linechar - parsers.pipe)^0)
10596                                       / strip_trailing_spaces
10597                                       / self.parser_functions.parse_inlines))
10598
10599     local table_caption
10600     if table_captions then
10601         table_caption = #table_caption_beginning
10602                       * table_caption_beginning
10603         if table_attributes then
10604             table_caption = table_caption
10605                           * (C(((( parsers.linechar
10606                                - (parsers.attributes
10607                                  * parsers.optionalspace
10608                                  * parsers.newline
10609                                  * -( parsers.optionalspace
10610                                    * parsers.linechar))))
10611                             + ( parsers.newline
10612                               * #( parsers.optionalspace
10613                                 * parsers.linechar)
10614                               * C(parsers.optionalspace) / writer.space))

```

```

10615             * (parsers.linechar
10616               - parsers.lbrace)^0)^1)
10617             / self.parser_functions.parse_inlines)
10618         * (parsers.newline
10619         + ( Ct(parsers.attributes)
10620           * parsers.optionalspace
10621           * parsers.newline))
10622     else
10623         table_caption = table_caption
10624         * C(( parsers.linechar
10625           + ( parsers.newline
10626             * #( parsers.optionalspace
10627               * parsers.linechar)
10628             * C(parsers.optionalspace) / writer.space))^1)
10629         / self.parser_functions.parse_inlines
10630         * parsers.newline
10631     end
10632 else
10633     table_caption = parsers.fail
10634 end
10635
10636 local PipeTable = Ct(table_row * parsers.newline * (parsers.check_minimal_indent
10637               * table_hline * parsers.newline
10638               * ((parsers.check_minimal_indent / {}) * table_row * parsers.
10639               / make_pipe_table_rectangular
10640               * table_caption^-1
10641               / writer.table
10642
10643     self.insert_pattern("Block after Blockquote",
10644                       PipeTable, "PipeTable")
10645 end
10646 }
10647 end

```

### 3.1.6.15 Raw Attributes

The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```

10648 M.extensions.raw_inline = function()
10649     return {
10650         name = "built-in raw_inline syntax extension",
10651         extend_writer = function(self)
10652             local options = self.options
10653

```

Define `writer->rawInline` as a function that will transform an input inline raw span `s` with the raw attribute `attr` to the output format.

```

10654     function self.rawInline(s, attr)
10655         if not self.is_writing then return "" end
10656         if self.flatten_inlines then return s end
10657         local name = util.cache_verbatim(options.cacheDir, s)
10658         return {"\\markdownRendererInputRawInline{" ,
10659             name,"}{" , self.string(attr),"}" }
10660     end
10661 end, extend_reader = function(self)
10662     local writer = self.writer
10663
10664     local RawInline = parsers.inticks
10665                     * parsers.raw_attribute
10666                     / writer.rawInline
10667
10668     self.insert_pattern("Inline before Code",
10669                         RawInline, "RawInline")
10670 end
10671 }
10672 end

```

### 3.1.6.16 Strike-Through

The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```

10673 M.extensions.strike_through = function()
10674     return {
10675         name = "built-in strike_through syntax extension",
10676         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.

```

10677         function self.strike_through(s)
10678             if self.flatten_inlines then return s end
10679             return {"\\markdownRendererStrikeThrough{" ,s,"}" }
10680         end
10681     end, extend_reader = function(self)
10682         local parsers = self.parsers
10683         local writer = self.writer
10684
10685         local StrikeThrough = (
10686             parsers.between(parsers.Inline, parsers.doubletildes,
10687                             parsers.doubletildes)
10688         ) / writer.strike_through
10689
10690     self.insert_pattern("Inline after LinkAndEmph",
10691                         StrikeThrough, "StrikeThrough")
10692
10693     self.add_special_character("~")

```



```

10694     end
10695   }
10696 end

```

### 3.1.6.17 Subscripts

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

```

10697 M.extensions.subscripts = function()
10698   return {
10699     name = "built-in subscripts syntax extension",
10700     extend_writer = function(self)

```

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

```

10701       function self.subscript(s)
10702         if self.flatten_inlines then return s end
10703         return {"\\markdownRendererSubscript{",s,""}
10704       end
10705     end, extend_reader = function(self)
10706       local parsers = self.parsers
10707       local writer = self.writer
10708
10709       local Subscript = (
10710         parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
10711       ) / writer.subscript
10712
10713       self.insert_pattern("Inline after LinkAndEmph",
10714                           Subscript, "Subscript")
10715
10716       self.add_special_character("~")
10717     end
10718   }
10719 end

```

### 3.1.6.18 Superscripts

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

```

10720 M.extensions.superscripts = function()
10721   return {
10722     name = "built-in superscripts syntax extension",
10723     extend_writer = function(self)

```

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

```

10724       function self.superscript(s)
10725         if self.flatten_inlines then return s end

```

```

10726         return {"\\markdownRendererSuperscript{" ,s,""}
10727     end
10728 end, extend_reader = function(self)
10729     local parsers = self.parsers
10730     local writer = self.writer
10731
10732     local Superscript = (
10733         parsers.between(parsers.Str, parsers.circumflex, parsers.circumflex)
10734     ) / writer.superscript
10735
10736     self.insert_pattern("Inline after LinkAndEmph",
10737         Superscript, "Superscript")
10738
10739     self.add_special_character("^")
10740 end
10741 }
10742 end

```

### 3.1.6.19 T<sub>E</sub>X Math

The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```

10743 M.extensions.tex_math = function(tex_math_dollars,
10744                                     tex_math_single_backslash,
10745                                     tex_math_double_backslash)
10746     return {
10747         name = "built-in tex_math syntax extension",
10748         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.

```

10749         function self.display_math(s)
10750             if self.flatten_inlines then return s end
10751             return {"\\markdownRendererDisplayMath{" ,self.math(s),"}"}
10752         end

```

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

```

10753         function self.inline_math(s)
10754             if self.flatten_inlines then return s end
10755             return {"\\markdownRendererInlineMath{" ,self.math(s),"}"}
10756         end
10757     end, extend_reader = function(self)
10758         local parsers = self.parsers
10759         local writer = self.writer
10760
10761         local function between(p, starter, ender)
10762             return (starter * Cs(p * (p - ender)^0) * ender)

```

```

10763     end
10764
10765     local function strip_preceding_spaces(str)
10766         return str:gsub("^%s*(.-)$", "%1")
10767     end
10768
10769     local allowed_before_closing = B( parsers.backslash * parsers.any
10770                                     + parsers.any * (parsers.any - parsers.backslash)
10771
10772     local allowed_before_closing_no_space = B( parsers.backslash * parsers.any
10773                                               + parsers.any * (parsers.nonspacechar
10774

```

The following patterns implement the Pandoc dollar math syntax extension.

```

10775     local dollar_math_content = (parsers.newline * (parsers.check_optional_indent /
10776                                     + parsers.backslash^-1
10777                                     * parsers.linechar)
10778     - parsers.blankline^2
10779     - parsers.dollar
10780
10781     local inline_math_opening_dollars = parsers.dollar
10782                                     * #(parsers.nonspacechar)
10783
10784     local inline_math_closing_dollars = allowed_before_closing_no_space
10785                                     * parsers.dollar
10786                                     * -#(parsers.digit)
10787
10788     local inline_math_dollars = between(Cs( dollar_math_content),
10789                                     inline_math_opening_dollars,
10790                                     inline_math_closing_dollars)
10791
10792     local display_math_opening_dollars = parsers.dollar
10793                                     * parsers.dollar
10794
10795     local display_math_closing_dollars = parsers.dollar
10796                                     * parsers.dollar
10797
10798     local display_math_dollars = between(Cs( dollar_math_content),
10799                                     display_math_opening_dollars,
10800                                     display_math_closing_dollars)

```

The following patterns implement the Pandoc single and double backslash math syntax extensions.

```

10801     local backslash_math_content = (parsers.newline * (parsers.check_optional_indent /
10802                                     + parsers.linechar)
10803     - parsers.blankline^2

```

The following patterns implement the Pandoc double backslash math syntax extension.

```

10804     local inline_math_opening_double = parsers.backslash
10805                                           * parsers.backslash
10806                                           * parsers.lparent
10807
10808     local inline_math_closing_double = allowed_before_closing
10809                                           * parsers.spacechar^0
10810                                           * parsers.backslash
10811                                           * parsers.backslash
10812                                           * parsers.rparent
10813
10814     local inline_math_double = between(Cs( backslash_math_content),
10815                                           inline_math_opening_double,
10816                                           inline_math_closing_double)
10817                                           / strip_preceding_whitespaces
10818
10819     local display_math_opening_double = parsers.backslash
10820                                           * parsers.backslash
10821                                           * parsers.lbracket
10822
10823     local display_math_closing_double = allowed_before_closing
10824                                           * parsers.spacechar^0
10825                                           * parsers.backslash
10826                                           * parsers.backslash
10827                                           * parsers.rbracket
10828
10829     local display_math_double = between(Cs( backslash_math_content),
10830                                           display_math_opening_double,
10831                                           display_math_closing_double)
10832                                           / strip_preceding_whitespaces

```

The following patterns implement the Pandoc single backslash math syntax extension.

```

10833     local inline_math_opening_single = parsers.backslash
10834                                           * parsers.lparent
10835
10836     local inline_math_closing_single = allowed_before_closing
10837                                           * parsers.spacechar^0
10838                                           * parsers.backslash
10839                                           * parsers.rparent
10840
10841     local inline_math_single = between(Cs( backslash_math_content),
10842                                           inline_math_opening_single,
10843                                           inline_math_closing_single)
10844                                           / strip_preceding_whitespaces
10845
10846     local display_math_opening_single = parsers.backslash

```

```

10847             * parsers.lbracket
10848
10849     local display_math_closing_single = allowed_before_closing
10850             * parsers.spacechar^0
10851             * parsers.backslash
10852             * parsers.rbracket
10853
10854     local display_math_single = between(Cs( backslash_math_content),
10855             display_math_opening_single,
10856             display_math_closing_single)
10857             / strip_preceding_whitespace
10858
10859     local display_math = parsers.fail
10860
10861     local inline_math = parsers.fail
10862
10863     if tex_math_dollars then
10864         display_math = display_math + display_math_dollars
10865         inline_math = inline_math + inline_math_dollars
10866     end
10867
10868     if tex_math_double_backslash then
10869         display_math = display_math + display_math_double
10870         inline_math = inline_math + inline_math_double
10871     end
10872
10873     if tex_math_single_backslash then
10874         display_math = display_math + display_math_single
10875         inline_math = inline_math + inline_math_single
10876     end
10877
10878     local TexMath = display_math / writer.display_math
10879             + inline_math / writer.inline_math
10880
10881     self.insert_pattern("Inline after LinkAndEmph",
10882             TexMath, "TexMath")
10883
10884     if tex_math_dollars then
10885         self.add_special_character("$")
10886     end
10887
10888     if tex_math_single_backslash or tex_math_double_backslash then
10889         self.add_special_character("\\")
10890         self.add_special_character("[")
10891         self.add_special_character("]")
10892         self.add_special_character("(")
10893         self.add_special_character("(")

```

```

10894     end
10895   end
10896 }
10897 end

```

### 3.1.6.20 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.

```

10898 M.extensions.jekyll_data = function(expect_jekyll_data)
10899   return {
10900     name = "built-in jekyll_data syntax extension",
10901     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`.

```

10902     function self.jekyllData(d, t, p)
10903       if not self.is_writing then return "" end
10904
10905       local buf = {}
10906
10907       local keys = {}
10908       for k, _ in pairs(d) do
10909         table.insert(keys, k)
10910       end
10911       table.sort(keys)
10912
10913       if not p then
10914         table.insert(buf, "\\markdownRendererJekyllDataBegin")
10915       end
10916
10917       if #d > 0 then
10918         table.insert(buf, "\\markdownRendererJekyllDataSequenceBegin{")
10919         table.insert(buf, self.identifier(p or "null"))
10920         table.insert(buf, "{")
10921         table.insert(buf, #keys)
10922         table.insert(buf, ")")
10923       else
10924         table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
10925         table.insert(buf, self.identifier(p or "null"))
10926         table.insert(buf, "{")
10927         table.insert(buf, #keys)

```

```

10928         table.insert(buf, "}")
10929     end
10930
10931     for _, k in ipairs(keys) do
10932         local v = d[k]
10933         local typ = type(v)
10934         k = tostring(k or "null")
10935         if typ == "table" and next(v) ~= nil then
10936             table.insert(
10937                 buf,
10938                 self.jekyllData(v, t, k)
10939             )
10940         else
10941             k = self.identifier(k)
10942             v = tostring(v)
10943             if typ == "boolean" then
10944                 table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
10945                 table.insert(buf, k)
10946                 table.insert(buf, "{")
10947                 table.insert(buf, v)
10948                 table.insert(buf, "}")
10949             elseif typ == "number" then
10950                 table.insert(buf, "\\markdownRendererJekyllDataNumber{")
10951                 table.insert(buf, k)
10952                 table.insert(buf, "{")
10953                 table.insert(buf, v)
10954                 table.insert(buf, "}")
10955             elseif typ == "string" then
10956                 table.insert(buf, "\\markdownRendererJekyllDataString{")
10957                 table.insert(buf, k)
10958                 table.insert(buf, "{")
10959                 table.insert(buf, t(v))
10960                 table.insert(buf, "}")
10961             elseif typ == "table" then
10962                 table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
10963                 table.insert(buf, k)
10964                 table.insert(buf, "}")
10965             else
10966                 error(format("Unexpected type %s for value of " ..
10967                     "YAML key %s", typ, k))
10968             end
10969         end
10970     end
10971
10972     if #d > 0 then
10973         table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
10974     else

```

```

10975         table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
10976     end
10977
10978     if not p then
10979         table.insert(buf, "\\markdownRendererJekyllDataEnd")
10980     end
10981
10982     return buf
10983 end
10984 end, extend_reader = function(self)
10985     local parsers = self.parsers
10986     local writer = self.writer
10987
10988     local JekyllData
10989         = Cmt( C((parsers.line - P("----") - P("..."))^0)
10990             , function(s, i, text) -- luacheck: ignore s i
10991                 local data
10992                 local ran_ok, _ = pcall(function()
10993                     -- TODO: Replace with `require("tinyyaml")` in TeX Live
10994                     local tinyyaml = require("markdown-tinyyaml")
10995                     data = tinyyaml.parse(text, {timestamps=false})
10996                 end)
10997                 if ran_ok and data ~= nil then
10998                     return true, writer.jekyllData(data, function(s)
10999                         return self.parser_functions.parse_blocks_nested(s)
11000                     end, nil)
11001                 else
11002                     return false
11003                 end
11004             end
11005         )
11006
11007     local UnexpectedJekyllData
11008         = P("----")
11009         * parsers.blankline / 0
11010         * #(-parsers.blankline) -- if followed by blank, it's thematic b
11011         * JekyllData
11012         * (P("----") + P("..."))
11013
11014     local ExpectedJekyllData
11015         = ( P("----")
11016             * parsers.blankline / 0
11017             * #(-parsers.blankline) -- if followed by blank, it's thematic
11018             )^-1
11019         * JekyllData
11020         * (P("----") + P("..."))^-1
11021

```



```

11022     self.insert_pattern("Block before Blockquote",
11023                          UnexpectedJekyllData, "UnexpectedJekyllData")
11024     if expect_jekyll_data then
11025         self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
11026     end
11027 end
11028 }
11029 end

```

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

The `new` function 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.

```

11030 function M.new(options)

```

Make the `options` table inherit from the `defaultOptions` table.

```

11031     options = options or {}
11032     setmetatable(options, { __index = function (_, key)
11033         return defaultOptions[key] end })

```

If the singleton cache contains a conversion function for the same `options`, reuse it.

```

11034     if options.singletonCache and singletonCache.convert then
11035         for k, v in pairs(defaultOptions) do
11036             if type(v) == "table" then
11037                 for i = 1, math.max(#singletonCache.options[k], #options[k]) do
11038                     if singletonCache.options[k][i] ~= options[k][i] then
11039                         goto miss
11040                     end
11041                 end
11042                 elseif singletonCache.options[k] ~= options[k] then
11043                     goto miss
11044                 end
11045             end
11046             return singletonCache.convert
11047         end
11048         ::miss::

```

Apply built-in syntax extensions based on `options`.

```

11049     local extensions = {}
11050
11051     if options.bracketedSpans then
11052         local bracketed_spans_extension = M.extensions.bracketed_spans()
11053         table.insert(extensions, bracketed_spans_extension)
11054     end
11055
11056     if options.contentBlocks then
11057         local content_blocks_extension = M.extensions.content_blocks(

```

```

11058         options.contentBlocksLanguageMap)
11059     table.insert(extensions, content_blocks_extension)
11060 end
11061
11062 if options.definitionLists then
11063     local definition_lists_extension = M.extensions.definition_lists(
11064         options.tightLists)
11065     table.insert(extensions, definition_lists_extension)
11066 end
11067
11068 if options.fencedCode then
11069     local fenced_code_extension = M.extensions.fenced_code(
11070         options.blankBeforeCodeFence,
11071         options.fencedCodeAttributes,
11072         options.rawAttribute)
11073     table.insert(extensions, fenced_code_extension)
11074 end
11075
11076 if options.fencedDivs then
11077     local fenced_div_extension = M.extensions.fenced_divs(
11078         options.blankBeforeDivFence)
11079     table.insert(extensions, fenced_div_extension)
11080 end
11081
11082 if options.headerAttributes then
11083     local header_attributes_extension = M.extensions.header_attributes()
11084     table.insert(extensions, header_attributes_extension)
11085 end
11086
11087 if options.inlineCodeAttributes then
11088     local inline_code_attributes_extension =
11089         M.extensions.inline_code_attributes()
11090     table.insert(extensions, inline_code_attributes_extension)
11091 end
11092
11093 if options.jekyllData then
11094     local jekyll_data_extension = M.extensions.jekyll_data(
11095         options.expectJekyllData)
11096     table.insert(extensions, jekyll_data_extension)
11097 end
11098
11099 if options.linkAttributes then
11100     local link_attributes_extension =
11101         M.extensions.link_attributes()
11102     table.insert(extensions, link_attributes_extension)
11103 end
11104

```

```

11105 if options.lineBlocks then
11106     local line_block_extension = M.extensions.line_blocks()
11107     table.insert(extensions, line_block_extension)
11108 end
11109
11110 if options.mark then
11111     local mark_extension = M.extensions.mark()
11112     table.insert(extensions, mark_extension)
11113 end
11114
11115 if options.pipeTables then
11116     local pipe_tables_extension = M.extensions.pipe_tables(
11117         options.tableCaptions, options.tableAttributes)
11118     table.insert(extensions, pipe_tables_extension)
11119 end
11120
11121 if options.rawAttribute then
11122     local raw_inline_extension = M.extensions.raw_inline()
11123     table.insert(extensions, raw_inline_extension)
11124 end
11125
11126 if options.strikeThrough then
11127     local strike_through_extension = M.extensions.strike_through()
11128     table.insert(extensions, strike_through_extension)
11129 end
11130
11131 if options.subscripts then
11132     local subscript_extension = M.extensions.subscripts()
11133     table.insert(extensions, subscript_extension)
11134 end
11135
11136 if options.superscripts then
11137     local superscript_extension = M.extensions.superscripts()
11138     table.insert(extensions, superscript_extension)
11139 end
11140
11141 if options.texMathDollars or
11142     options.texMathSingleBackslash or
11143     options.texMathDoubleBackslash then
11144     local tex_math_extension = M.extensions.tex_math(
11145         options.texMathDollars,
11146         options.texMathSingleBackslash,
11147         options.texMathDoubleBackslash)
11148     table.insert(extensions, tex_math_extension)
11149 end
11150
11151 if options.notes or options.inlineNotes then

```

```

11152     local notes_extension = M.extensions.notes(
11153         options.notes, options.inlineNotes)
11154     table.insert(extensions, notes_extension)
11155 end
11156
11157 if options.citations then
11158     local citations_extension = M.extensions.citations(options.citationNbsps)
11159     table.insert(extensions, citations_extension)
11160 end
11161
11162 if options.fancyLists then
11163     local fancy_lists_extension = M.extensions.fancy_lists()
11164     table.insert(extensions, fancy_lists_extension)
11165 end

```

Apply user-defined syntax extensions based on `options.extensions`.

```

11166 for _, user_extension_filename in ipairs(options.extensions) do
11167     local user_extension = (function(filename)

```

First, load and compile the contents of the user-defined syntax extension.

```

11168         local pathname = util.lookup_files(filename)
11169         local input_file = assert(io.open(pathname, "r"),
11170             [[Could not open user-defined syntax extension "]]
11171             .. pathname .. [[ for reading]])
11172         local input = assert(input_file:read("*a"))
11173         assert(input_file:close())
11174         local user_extension, err = load([[
11175             local sandbox = {}
11176             setmetatable(sandbox, {__index = _G})
11177             _ENV = sandbox
11178         ]] .. input)()
11179         assert(user_extension,
11180             [[Failed to compile user-defined syntax extension "]]
11181             .. pathname .. [[: ]] .. (err or []))

```

Then, validate the user-defined syntax extension.

```

11182         assert(user_extension.api_version ~= nil,
11183             [[User-defined syntax extension "]] .. pathname
11184             .. [[ does not specify mandatory field "api_version"]])
11185         assert(type(user_extension.api_version) == "number",
11186             [[User-defined syntax extension "]] .. pathname
11187             .. [[ specifies field "api_version" of type "]]
11188             .. type(user_extension.api_version)
11189             .. [[ but "number" was expected]])
11190         assert(user_extension.api_version > 0
11191             and user_extension.api_version <= metadata.user_extension_api_version,
11192             [[User-defined syntax extension "]] .. pathname
11193             .. [[ uses syntax extension API version "]]

```

```

11194     .. user_extension.api_version .. [[ but markdown.lua ]]
11195     .. metadata.version .. [[ uses API version ]]
11196     .. metadata.user_extension_api_version
11197     .. [[, which is incompatible]])
11198
11199     assert(user_extension.grammar_version ~= nil,
11200           [[User-defined syntax extension "]] .. pathname
11201           .. [[ " does not specify mandatory field "grammar_version"]])
11202     assert(type(user_extension.grammar_version) == "number",
11203           [[User-defined syntax extension "]] .. pathname
11204           .. [[ " specifies field "grammar_version" of type "]]
11205           .. type(user_extension.grammar_version)
11206           .. [[ " but "number" was expected]])
11207     assert(user_extension.grammar_version == metadata.grammar_version,
11208           [[User-defined syntax extension "]] .. pathname
11209           .. [[ " uses grammar version "]] .. user_extension.grammar_version
11210           .. [[ but markdown.lua ]] .. metadata.version
11211           .. [[ uses grammar version ]] .. metadata.grammar_version
11212           .. [[, which is incompatible]])
11213
11214     assert(user_extension.finalize_grammar ~= nil,
11215           [[User-defined syntax extension "]] .. pathname
11216           .. [[ " does not specify mandatory "finalize_grammar" field]])
11217     assert(type(user_extension.finalize_grammar) == "function",
11218           [[User-defined syntax extension "]] .. pathname
11219           .. [[ " specifies field "finalize_grammar" of type "]]
11220           .. type(user_extension.finalize_grammar)
11221           .. [[ " 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 3.1.6.)

```

11222     local extension = {
11223       name = [[user-defined "]] .. pathname .. [[ " syntax extension]],
11224       extend_reader = user_extension.finalize_grammar,
11225       extend_writer = function() end,
11226     }
11227     return extension
11228   end)(user_extension_filename)
11229   table.insert(extensions, user_extension)
11230 end

```

Produce a conversion function from markdown to plain TeX.

```

11231 local writer = M.writer.new(options)
11232 local reader = M.reader.new(writer, options)
11233 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`.

```

11234 collectgarbage("collect")
      Update the singleton cache.
11235 if options.singletonCache then
11236     local singletonCacheOptions = {}
11237     for k, v in pairs(options) do
11238         singletonCacheOptions[k] = v
11239     end
11240     setmetatable(singletonCacheOptions,
11241         { __index = function (_, key)
11242             return defaultOptions[key] end })
11243     singletonCache.options = singletonCacheOptions
11244     singletonCache.convert = convert
11245 end
      Return the conversion function from markdown to plain TEX.
11246 return convert
11247 end
11248
11249 return M

```

### 3.1.8 Command-Line Implementation

The command-line implementation provides the actual conversion routine for the command-line interface described in Section 2.1.7.

```

11250
11251 local input
11252 if input_filename then
11253     local input_file = assert(io.open(input_filename, "r"),
11254         [[Could not open file ]] .. input_filename .. [[ for reading]])
11255     input = assert(input_file:read("*a"))
11256     assert(input_file:close())
11257 else
11258     input = assert(io.read("*a"))
11259 end
11260

```

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

```

11261 local lfs = require("lfs")
11262 if options.cacheDir and not lfs.isdir(options.cacheDir) then
11263     assert(lfs.mkdir(options["cacheDir"]))
11264 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.

```

11265 local kpse
11266 (function()
11267     local should_initialize = package.loaded.kpse == nil

```

```

11268             or tex.initialize ~= nil
11269     kpse = require("kpse")
11270     if should_initialize then
11271         kpse.set_program_name("luatex")
11272     end
11273 end)()
11274 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.

```

11275 if metadata.version ~= md.metadata.version then
11276     warn("markdown-cli.lua " .. metadata.version .. " used with " ..
11277         "markdown.lua " .. md.metadata.version .. ".")
11278 end
11279 local convert = md.new(options)
11280 local output = convert(input)
11281
11282 if output_filename then
11283     local output_file = assert(io.open(output_filename, "w"),
11284         [[Could not open file ]] .. output_filename .. [[ for writing]])
11285     assert(output_file:write(output))
11286     assert(output_file:close())
11287 else
11288     assert(io.write(output))
11289 end

```

Remove the `options.cacheDir` directory if it is empty.

```

11290 if options.cacheDir then
11291     lfs.rmdir(options["cacheDir"])
11292 end

```

## 3.2 Plain T<sub>E</sub>X Implementation

The plain T<sub>E</sub>X implementation provides macros for the interfacing between T<sub>E</sub>X and Lua and for the buffering of input text. These macros are then used to implement the macros for the conversion from markdown to plain T<sub>E</sub>X exposed by the plain T<sub>E</sub>X interface (see Section 2.2).

### 3.2.1 Logging Facilities

```

11293 \ExplSyntaxOn
11294 \cs_if_free:NT
11295     \markdownInfo
11296     {
11297         \cs_new:Npn
11298             \markdownInfo #1
11299         {

```

```

11300         \msg_info:nne
11301         { markdown }
11302         { generic-message }
11303         { #1 }
11304     }
11305 }
11306 \cs_if_free:NT
11307     \markdownWarning
11308     {
11309         \cs_new:Npn
11310             \markdownWarning #1
11311             {
11312                 \msg_warning:nne
11313                 { markdown }
11314                 { generic-message }
11315                 { #1 }
11316             }
11317     }
11318 \cs_if_free:NT
11319     \markdownError
11320     {
11321         \cs_new:Npn
11322             \markdownError #1 #2
11323             {
11324                 \msg_error:nnee
11325                 { markdown }
11326                 { generic-message-with-help-text }
11327                 { #1 }
11328                 { #2 }
11329             }
11330     }
11331 \msg_new:nnn
11332     { markdown }
11333     { generic-message }
11334     { #1 }
11335 \msg_new:nnnn
11336     { markdown }
11337     { generic-message-with-help-text }
11338     { #1 }
11339     { #2 }
11340 \cs_generate_variant:Nn
11341     \msg_info:nnn
11342     { nne }
11343 \cs_generate_variant:Nn
11344     \msg_warning:nnn
11345     { nne }
11346 \cs_generate_variant:Nn

```



```

11347 \msg_error:nnnn
11348 { nnee }
11349 \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 TeX themes provided with the Markdown package.

```

11350 \ExplSyntaxOn
11351 \prop_new:N \g_@@_plain_tex_loaded_themes_linenos_prop
11352 \cs_new:Nn
11353   \@@_plain_tex_load_theme:nn
11354   {
11355     \prop_get:NnNTF
11356       \g_@@_plain_tex_loaded_themes_linenos_prop
11357       { #1 }
11358       \l_tmpa_tl
11359       {
11360         \msg_warning:nnnV
11361           { markdown }
11362           { repeatedly-loaded-plain-tex-theme }
11363           { #1 }
11364           \l_tmpa_tl
11365       }
11366       {
11367         \msg_info:nnn
11368           { markdown }
11369           { loading-plain-tex-theme }
11370           { #1 }
11371         \prop_gput:Nnx
11372           \g_@@_plain_tex_loaded_themes_linenos_prop
11373           { #1 }
11374           { \tex_the:D \tex_inputlineno:D }
11375         \file_input:n
11376           { markdown theme #2 }
11377       }
11378   }
11379 \msg_new:nnn
11380   { markdown }
11381   { loading-plain-tex-theme }
11382   { Loading~plain~TeX~Markdown~theme~#1 }
11383 \msg_new:nnn
11384   { markdown }
11385   { repeatedly-loaded-plain-tex-theme }
11386   {
11387     Plain~TeX~Markdown~theme~#1~was~previously~

```

```

11388     loaded~on~line~#2,~not~loading~it~again
11389   }
11390   \cs_generate_variant:Nn
11391     \prop_gput:Nnn
11392     { Nnx }
11393   \cs_gset_eq:NN
11394     \@@_load_theme:nn
11395     \@@_plain_tex_load_theme:nn
11396   \cs_generate_variant:Nn
11397     \@@_load_theme:nn
11398     { nV }

```

Developers can use the `\markdownLoadPlainTeXTheme` macro to load a corresponding plain TeX theme from within themes for higher-level TeX formats such as L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt.

```

11399   \cs_new:Npn
11400     \markdownLoadPlainTeXTheme
11401     {

```

First, we extract the name of the current theme from the `\g_@@_current_theme_tl` macro.

```

11402     \tl_set:NV
11403       \l_tmpa_tl
11404       \g_@@_current_theme_tl
11405     \tl_reverse:N
11406       \l_tmpa_tl
11407     \tl_set:Ne
11408       \l_tmpb_tl
11409       {
11410         \tl_tail:V
11411         \l_tmpa_tl
11412       }
11413     \tl_reverse:N
11414       \l_tmpb_tl

```

Next, we munge the theme name.

```

11415     \str_set:NV
11416       \l_tmpa_str
11417       \l_tmpb_tl
11418     \str_replace_all:Nnn
11419       \l_tmpa_str
11420       { / }
11421       { _ }

```

Finally, we load the plain TeX theme.

```

11422     \@@_plain_tex_load_theme:VV
11423       \l_tmpb_tl
11424       \l_tmpa_str

```

```

11425 }
11426 \cs_generate_variant:Nn
11427 \tl_set:Nn
11428 { Ne }
11429 \cs_generate_variant:Nn
11430 \@@_plain_tex_load_theme:nn
11431 { VV }
11432 \ExplSyntaxOff

```

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

```

11433 \markdownSetup {
11434   rendererPrototypes = {
11435     tilde = {~},
11436   },
11437 }

```

The [witiko/markdown/defaults](#) plain  $\TeX$  theme provides default definitions for token renderer prototypes. See Section 3.2.3 for the actual definitions.

### 3.2.3 Token Renderer Prototypes

The following definitions should be considered placeholder.

```

11438 \def\markdownRendererInterblockSeparatorPrototype{\par}%
11439 \def\markdownRendererParagraphSeparatorPrototype{%
11440   \markdownRendererInterblockSeparator}%
11441 \def\markdownRendererHardLineBreakPrototype{\hfil\break}%
11442 \def\markdownRendererSoftLineBreakPrototype{ }%
11443 \let\markdownRendererEllipsisPrototype\dots
11444 \def\markdownRendererNbspPrototype{~}%
11445 \def\markdownRendererLeftBracePrototype{\char`\{}%
11446 \def\markdownRendererRightBracePrototype{\char`\}%
11447 \def\markdownRendererDollarSignPrototype{\char`\$}%
11448 \def\markdownRendererPercentSignPrototype{\char`\}%
11449 \def\markdownRendererAmpersandPrototype{\&}%
11450 \def\markdownRendererUnderscorePrototype{\char`\_%
11451 \def\markdownRendererHashPrototype{\char`\#}%
11452 \def\markdownRendererCircumflexPrototype{\char`\^}%
11453 \def\markdownRendererBackslashPrototype{\char`\}%
11454 \def\markdownRendererTildePrototype{\char`\~}%
11455 \def\markdownRendererPipePrototype{|}%
11456 \def\markdownRendererCodeSpanPrototype#1{{\tt#1}}%
11457 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
11458 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
11459   \markdownInput{#3}}%
11460 \def\markdownRendererContentBlockOnlineImagePrototype{%
11461   \markdownRendererImage}%
11462 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%

```

```

11463 \markdownRendererInputFencedCode{#3}{#2}{#2}}%
11464 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
11465 \def\markdownRendererUlBeginPrototype{}%
11466 \def\markdownRendererUlBeginTightPrototype{}%
11467 \def\markdownRendererUlItemPrototype{}%
11468 \def\markdownRendererUlItemEndPrototype{}%
11469 \def\markdownRendererUlEndPrototype{}%
11470 \def\markdownRendererUlEndTightPrototype{}%
11471 \def\markdownRendererOlBeginPrototype{}%
11472 \def\markdownRendererOlBeginTightPrototype{}%
11473 \def\markdownRendererFancyOlBeginPrototype#1#2{\markdownRendererOlBegin}%
11474 \def\markdownRendererFancyOlBeginTightPrototype#1#2{\markdownRendererOlBeginTight}%
11475 \def\markdownRendererOlItemPrototype{}%
11476 \def\markdownRendererOlItemWithNumberPrototype#1{}%
11477 \def\markdownRendererOlItemEndPrototype{}%
11478 \def\markdownRendererFancyOlItemPrototype{\markdownRendererOlItem}%
11479 \def\markdownRendererFancyOlItemWithNumberPrototype{\markdownRendererOlItemWithNumber}%
11480 \def\markdownRendererFancyOlItemEndPrototype{}%
11481 \def\markdownRendererOlEndPrototype{}%
11482 \def\markdownRendererOlEndTightPrototype{}%
11483 \def\markdownRendererFancyOlEndPrototype{\markdownRendererOlEnd}%
11484 \def\markdownRendererFancyOlEndTightPrototype{\markdownRendererOlEndTight}%
11485 \def\markdownRendererDlBeginPrototype{}%
11486 \def\markdownRendererDlBeginTightPrototype{}%
11487 \def\markdownRendererDlItemPrototype#1{#1}%
11488 \def\markdownRendererDlItemEndPrototype{}%
11489 \def\markdownRendererDlDefinitionBeginPrototype{}%
11490 \def\markdownRendererDlDefinitionEndPrototype{\par}%
11491 \def\markdownRendererDlEndPrototype{}%
11492 \def\markdownRendererDlEndTightPrototype{}%
11493 \def\markdownRendererEmphasisPrototype#1{{\it#1}}%
11494 \def\markdownRendererStrongEmphasisPrototype#1{{\bf#1}}%
11495 \def\markdownRendererBlockQuoteBeginPrototype{\begingroup\it}%
11496 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
11497 \def\markdownRendererLineBlockBeginPrototype{\begingroup\parindent=0pt}%
11498 \def\markdownRendererLineBlockEndPrototype{\endgroup}%
11499 \def\markdownRendererInputVerbatimPrototype#1{%
11500 \par{\tt\input#1\relax{}}\par}%
11501 \def\markdownRendererInputFencedCodePrototype#1#2#3{%
11502 \markdownRendererInputVerbatim{#1}}%
11503 \def\markdownRendererHeadingOnePrototype#1{#1}%
11504 \def\markdownRendererHeadingTwoPrototype#1{#1}%
11505 \def\markdownRendererHeadingThreePrototype#1{#1}%
11506 \def\markdownRendererHeadingFourPrototype#1{#1}%
11507 \def\markdownRendererHeadingFivePrototype#1{#1}%
11508 \def\markdownRendererHeadingSixPrototype#1{#1}%
11509 \def\markdownRendererThematicBreakPrototype{}%

```

```

11510 \def\markdownRendererNotePrototype#1{#1}%
11511 \def\markdownRendererCitePrototype#1{}%
11512 \def\markdownRendererTextCitePrototype#1{}%
11513 \def\markdownRendererTickedBoxPrototype{[X]}%
11514 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
11515 \def\markdownRendererUntickedBoxPrototype{[ ]}%
11516 \def\markdownRendererStrikeThroughPrototype#1{#1}%
11517 \def\markdownRendererSuperscriptPrototype#1{#1}%
11518 \def\markdownRendererSubscriptPrototype#1{#1}%
11519 \def\markdownRendererDisplayMathPrototype#1{$$#1$$}%
11520 \def\markdownRendererInlineMathPrototype#1{$#1$}%
11521 \ExplSyntaxOn
11522 \cs_gset:Npn
11523   \markdownRendererHeaderAttributeContextBeginPrototype
11524   {
11525     \group_begin:
11526     \color_group_begin:
11527   }
11528 \cs_gset:Npn
11529   \markdownRendererHeaderAttributeContextEndPrototype
11530   {
11531     \color_group_end:
11532     \group_end:
11533   }
11534 \cs_gset_eq:NN
11535   \markdownRendererBracketedSpanAttributeContextBeginPrototype
11536   \markdownRendererHeaderAttributeContextBeginPrototype
11537 \cs_gset_eq:NN
11538   \markdownRendererBracketedSpanAttributeContextEndPrototype
11539   \markdownRendererHeaderAttributeContextEndPrototype
11540 \cs_gset_eq:NN
11541   \markdownRendererFencedDivAttributeContextBeginPrototype
11542   \markdownRendererHeaderAttributeContextBeginPrototype
11543 \cs_gset_eq:NN
11544   \markdownRendererFencedDivAttributeContextEndPrototype
11545   \markdownRendererHeaderAttributeContextEndPrototype
11546 \cs_gset_eq:NN
11547   \markdownRendererFencedCodeAttributeContextBeginPrototype
11548   \markdownRendererHeaderAttributeContextBeginPrototype
11549 \cs_gset_eq:NN
11550   \markdownRendererFencedCodeAttributeContextEndPrototype
11551   \markdownRendererHeaderAttributeContextEndPrototype
11552 \cs_gset:Npn
11553   \markdownRendererReplacementCharacterPrototype
11554   { \codepoint_str_generate:n { fffd } }
11555 \ExplSyntaxOff
11556 \def\markdownRendererSectionBeginPrototype{}%

```

```
11557 \def\markdownRendererSectionEndPrototype{}%
```

### 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.

```
11558 \ExplSyntaxOn
11559 \cs_new:Nn
11560   \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
11561   {
11562     \str_case:nn
11563       { #2 }
11564       {
11565         { md } { \markdownInput{#1} }
11566         { tex } { \markdownEscape{#1} \unskip }
11567       }
11568   }
11569 \cs_new:Nn
11570   \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
11571   {
11572     \str_case:nn
11573       { #2 }
11574       {
11575         { md } { \markdownInput{#1} }
11576         { tex } { \markdownEscape{#1} }
11577       }
11578   }
11579 \cs_gset:Npn
11580   \markdownRendererInputRawInlinePrototype#1#2
11581   {
11582     \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
11583       { #1 }
11584       { #2 }
11585   }
11586 \cs_gset:Npn
11587   \markdownRendererInputRawBlockPrototype#1#2
11588   {
11589     \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
11590       { #1 }
11591       { #2 }
11592   }
11593 \ExplSyntaxOff
```

### 3.2.3.2 YAML Metadata Renderer Prototypes

To keep track of the current type of structure we inhabit when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_datatypes_seq` stack. At every step of the traversal, the stack will contain one of the following constants at any position  $p$ :

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

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

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

```
11594 \ExplSyntaxOn
11595 \seq_new:N \g_@@_jekyll_data_datatypes_seq
11596 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
11597 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
11598 \tl_const:Nn \c_@@_jekyll_data_scalar_tl { scalar }
```

To keep track of our current place when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_wildcard_absolute_address_seq` stack of keys using the `\markdown_jekyll_data_push_address_segment:n` macro.

```
11599 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
11600 \cs_new:Nn \markdown_jekyll_data_push_address_segment:n
11601 {
11602   \seq_if_empty:NF
11603     \g_@@_jekyll_data_datatypes_seq
11604     {
11605       \seq_get_right:NN
11606       \g_@@_jekyll_data_datatypes_seq
11607       \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.

```
11608   \str_if_eq:NNTF
11609     \l_tmpa_tl
11610     \c_@@_jekyll_data_sequence_tl
11611     {
11612       \seq_put_right:Nn
11613       \g_@@_jekyll_data_wildcard_absolute_address_seq
11614       { * }
11615     }
```

```

11616     {
11617         \seq_put_right:Nn
11618         \g_@@_jekyll_data_wildcard_absolute_address_seq
11619         { #1 }
11620     }
11621 }
11622 }

```

Out of `\g_@@_jekyll_data_wildcard_absolute_address_seq`, we will construct the following two token lists:

`\g_@@_jekyll_data_wildcard_absolute_address_tl` An *absolute wildcard*: The wildcard from the root of the document prefixed with a slash (/) with individual keys and asterisks also delimited by slashes. Allows the users to react to complex context-sensitive structures with ease.

For example, the `name` key in the following YAML document would correspond to the `/*/person/name` absolute wildcard:

```
[{person: {name: Elon, surname: Musk}}]
```

`\g_@@_jekyll_data_wildcard_relative_address_tl` A *relative wildcard*: The rightmost segment of the wildcard. Allows the users to react to simple context-free structures.

For example, the `name` key in the following YAML document would correspond to the `name` relative wildcard:

```
[{person: {name: Elon, surname: Musk}}]
```

We will construct `\g_@@_jekyll_data_wildcard_absolute_address_tl` using the `\markdown_jekyll_data_concatenate_address:NN` macro and we will construct both token lists using the `\markdown_jekyll_data_update_address_tls:` macro.

```

11623 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
11624 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
11625 \cs_new:Nn \markdown_jekyll_data_concatenate_address:NN
11626 {
11627     \seq_pop_left:NN #1 \l_tmpa_tl
11628     \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
11629     \seq_put_left:NV #1 \l_tmpa_tl
11630 }
11631 \cs_new:Nn \markdown_jekyll_data_update_address_tls:
11632 {
11633     \markdown_jekyll_data_concatenate_address:NN
11634     \g_@@_jekyll_data_wildcard_absolute_address_seq

```



```

11635     \g_@@_jekyll_data_wildcard_absolute_address_tl
11636     \seq_get_right:NN
11637     \g_@@_jekyll_data_wildcard_absolute_address_seq
11638     \g_@@_jekyll_data_wildcard_relative_address_tl
11639 }

```

To make sure that the stacks and token lists stay in sync, we will use the `\markdown_jekyll_data_push:nN` and `\markdown_jekyll_data_pop:` macros.

```

11640 \cs_new:Nn \markdown_jekyll_data_push:nN
11641 {
11642     \markdown_jekyll_data_push_address_segment:n
11643     { #1 }
11644     \seq_put_right:NV
11645     \g_@@_jekyll_data_datatypes_seq
11646     #2
11647     \markdown_jekyll_data_update_address_tls:
11648 }
11649 \cs_new:Nn \markdown_jekyll_data_pop:
11650 {
11651     \seq_pop_right:NN
11652     \g_@@_jekyll_data_wildcard_absolute_address_seq
11653     \l_tmpa_tl
11654     \seq_pop_right:NN
11655     \g_@@_jekyll_data_datatypes_seq
11656     \l_tmpa_tl
11657     \markdown_jekyll_data_update_address_tls:
11658 }

```

To set a single key–value, we will use the `\markdown_jekyll_data_set_keyval:Nn` macro, ignoring unknown keys. To set key–values for both absolute and relative wildcards, we will use the `\markdown_jekyll_data_set_keyvals:nn` macro.

```

11659 \cs_new:Nn \markdown_jekyll_data_set_keyval:nn
11660 {
11661     \keys_set_known:nn
11662     { markdown/jekyllData }
11663     { { #1 } = { #2 } }
11664 }
11665 \cs_generate_variant:Nn
11666     \markdown_jekyll_data_set_keyval:nn
11667     { Vn }
11668 \cs_new:Nn \markdown_jekyll_data_set_keyvals:nn
11669 {
11670     \markdown_jekyll_data_push:nN
11671     { #1 }
11672     \c_@@_jekyll_data_scalar_tl
11673     \markdown_jekyll_data_set_keyval:Vn
11674     \g_@@_jekyll_data_wildcard_absolute_address_tl
11675     { #2 }

```

```

11676 \markdown_jekyll_data_set_keyval:Vn
11677 \g_@@_jekyll_data_wildcard_relative_address_tl
11678 { #2 }
11679 \markdown_jekyll_data_pop:
11680 }

```

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

```

11681 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
11682 \markdown_jekyll_data_push:nN
11683 { #1 }
11684 \c_@@_jekyll_data_sequence_tl
11685 }
11686 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
11687 \markdown_jekyll_data_push:nN
11688 { #1 }
11689 \c_@@_jekyll_data_mapping_tl
11690 }
11691 \def\markdownRendererJekyllDataSequenceEndPrototype{
11692 \markdown_jekyll_data_pop:
11693 }
11694 \def\markdownRendererJekyllDataMappingEndPrototype{
11695 \markdown_jekyll_data_pop:
11696 }
11697 \def\markdownRendererJekyllDataBooleanPrototype#1#2{
11698 \markdown_jekyll_data_set_keyvals:nn
11699 { #1 }
11700 { #2 }
11701 }
11702 \def\markdownRendererJekyllDataEmptyPrototype#1{}
11703 \def\markdownRendererJekyllDataNumberPrototype#1#2{
11704 \markdown_jekyll_data_set_keyvals:nn
11705 { #1 }
11706 { #2 }
11707 }
11708 \def\markdownRendererJekyllDataStringPrototype#1#2{
11709 \markdown_jekyll_data_set_keyvals:nn
11710 { #1 }
11711 { #2 }
11712 }
11713 \ExplSyntaxOff

```

If plain  $\text{\TeX}$  is the top layer, we load the [witiko/markdown/defaults](#) plain  $\text{\TeX}$  theme with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```

11714 \ExplSyntaxOn
11715 \str_if_eq:VVT
11716 \c_@@_top_layer_tl

```

```

11717 \c_@@_option_layer_plain_tex_tl
11718 {
11719   \ExplSyntaxOff
11720   \@@_if_option:nF
11721     { noDefaults }
11722     {
11723       \@@_setup:n
11724         {theme = witiko/markdown/defaults}
11725     }
11726   \ExplSyntaxOn
11727 }
11728 \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).

```

11729 \ExplSyntaxOn
11730 \tl_new:N \g_@@_formatted_lua_options_tl
11731 \cs_new:Nn \@@_format_lua_options:
11732 {
11733   \tl_gclear:N
11734     \g_@@_formatted_lua_options_tl
11735   \seq_map_function:NN
11736     \g_@@_lua_options_seq
11737     \@@_format_lua_option:n
11738 }
11739 \cs_new:Nn \@@_format_lua_option:n
11740 {
11741   \@@_typecheck_option:n
11742     { #1 }
11743   \@@_get_option_type:nN
11744     { #1 }
11745   \l_tmpa_tl
11746   \bool_case_true:nF
11747     {
11748       {
11749         \str_if_eq_p:VV
11750           \l_tmpa_tl
11751             \c_@@_option_type_boolean_tl ||
11752         \str_if_eq_p:VV
11753           \l_tmpa_tl
11754             \c_@@_option_type_number_tl ||
11755         \str_if_eq_p:VV
11756           \l_tmpa_tl

```

```

11757         \c_@@_option_type_counter_tl
11758     }
11759     {
11760         \@@_get_option_value:nN
11761         { #1 }
11762         \l_tmpa_tl
11763         \tl_gput_right:Nx
11764         \g_@@_formatted_lua_options_tl
11765         { #1~== \l_tmpa_tl ,~ }
11766     }
11767     {
11768         \str_if_eq_p:VV
11769         \l_tmpa_tl
11770         \c_@@_option_type_clist_tl
11771     }
11772     {
11773         \@@_get_option_value:nN
11774         { #1 }
11775         \l_tmpa_tl
11776         \tl_gput_right:Nx
11777         \g_@@_formatted_lua_options_tl
11778         { #1~==\c_left_brace_str }
11779         \clist_map_inline:Vn
11780         \l_tmpa_tl
11781         {
11782             \tl_gput_right:Nx
11783             \g_@@_formatted_lua_options_tl
11784             { "##1" ,~ }
11785         }
11786         \tl_gput_right:Nx
11787         \g_@@_formatted_lua_options_tl
11788         { \c_right_brace_str ,~ }
11789     }
11790 }
11791 {
11792     \@@_get_option_value:nN
11793     { #1 }
11794     \l_tmpa_tl
11795     \tl_gput_right:Nx
11796     \g_@@_formatted_lua_options_tl
11797     { #1~== " \l_tmpa_tl " ,~ }
11798 }
11799 }
11800 \cs_generate_variant:Nn
11801 \clist_map_inline:nn
11802 { Vn }
11803 \let\markdownPrepareLuaOptions=\@@_format_lua_options:

```

```

11804 \def\markdownLuaOptions{{ \g_@@_formatted_lua_options_tl }}
11805 \ExplSyntaxOff

```

The `\markdownPrepare` macro contains the Lua code that is executed prior to any conversion from markdown to plain T<sub>E</sub>X. It exposes the `convert` function for the use by any further Lua code.

```

11806 \def\markdownPrepare{%
First, ensure that the cacheDir directory exists.
11807   local lfs = require("lfs")
11808   local cacheDir = "\markdownOptionCacheDir"
11809   if not lfs.isdir(cacheDir) then
11810     assert(lfs.mkdir(cacheDir))
11811   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.

```

11812   local md = require("markdown")
11813   local convert = md.new(\markdownLuaOptions)
11814 }%

```

The `\markdownCleanup` macro contains the Lua code that is executed after any conversion from markdown to plain T<sub>E</sub>X.

```

11815 \def\markdownCleanup{%
Remove the options.cacheDir directory if it is empty.
11816   lfs.rmdir(cacheDir)
11817 }%

```

### 3.2.5 Buffering 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.

```

11818 \csname newread\endcsname\markdownInputFileStream
11819 \csname newwrite\endcsname\markdownOutputFileStream

```

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

```

11820 \begingroup
11821   \catcode\^^I=12%
11822   \gdef\markdownReadAndConvertTab{^^I}%
11823 \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.

```

11824 \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.

```

11825 \catcode\^M=13%
11826 \catcode\^I=13%
11827 \catcode|=0%
11828 \catcode\|=12%
11829 |catcode@=14%
11830 |catcode|%=12@
11831 |gdef|markdownReadAndConvert#1#2{@
11832 |begingroup@

```

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

```

11833 |markdownIfOption{frozenCache}{@
11834 |immediate|openout|markdownOutputFileStream@
11835 |markdownOptionInputTempFileName|relax@
11836 |markdownInfo{Buffering markdown input into the temporary @
11837 |input file "|markdownOptionInputTempFileName" and scanning @
11838 |for the closing token sequence "#1"}@
11839 }@

```

Locally change the category of the special plain TeX characters to *other* in order to prevent unwanted interpretation of the input. Change also the category of the space character, so that we can retrieve it unaltered.

```

11840 |def|do##1{|catcode`##1=12}|dospecials@
11841 |catcode`|=12@
11842 |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.

```

11843 |def|markdownReadAndConvertStripPercentSign##1{@
11844 |markdownIfOption{stripPercentSigns}{@
11845 |if##1%@
11846 |expandafter|expandafter|expandafter@
11847 |markdownReadAndConvertProcessLine@
11848 |else@
11849 |expandafter|expandafter|expandafter@
11850 |markdownReadAndConvertProcessLine@
11851 |expandafter|expandafter|expandafter##1@
11852 |fi@
11853 }{@
11854 |expandafter@
11855 |markdownReadAndConvertProcessLine@
11856 |expandafter##1@

```

```

11857     }@
11858     }@

```

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.

```

11859     |def|markdownReadAndConvertProcessLine##1#1##2#1##3|relax{@

```

If we are not reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, store the line in the `inputTempFileName` file. If we are reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, gobble the line.

```

11860         |ifx|relax##3|relax@
11861         |markdownIfOption{frozenCache}{@}{@
11862         |immediate|write|markdownOutputFileStream{##1}@
11863         }@
11864         |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.

```

11865         |def^^M{@
11866         |markdownInfo{The ending token sequence was found}@
11867         |markdownIfOption{frozenCache}{@}{@
11868         |immediate|closeout|markdownOutputFileStream@
11869         }@
11870         |endgroup@
11871         |markdownInput{@
11872         |markdownOptionOutputDir@
11873         /|markdownOptionInputTempFileName@
11874         }@
11875         #2}@
11876         |fi@

```

Repeat with the next line.

```

11877         ^^M}@

```

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

```

11878         |catcode`|^I=13@
11879         |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.

```

11880         |catcode`|^M=13@
11881         |def^^M##1^^M{@

```

```

11882      |def^^M###1^^M{@
11883      |markdownReadAndConvertStripPercentSign###1#1#1|relax}@
11884      ^^M}@
11885      ^^M}@

```

Reset the character categories back to the former state.

```

11886 |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.

```

11887 \ExplSyntaxOn
11888 \cs_new:Npn
11889   \markdownLuaExecute
11890   #1
11891   {
11892     \int_compare:nNt
11893       { \g_luabridge_method_int }
11894       =
11895       { \c_luabridge_method_shell_int }
11896       {
11897         \sys_if_shell_unrestricted:F
11898         {
11899           \sys_if_shell:TF
11900           {
11901             \msg_error:nn
11902               { markdown }
11903               { restricted-shell-access }
11904           }
11905           {
11906             \msg_error:nn
11907               { markdown }
11908               { disabled-shell-access }
11909           }
11910         }
11911       }
11912     \luabridge_now:e
11913     { #1 }
11914   }
11915 \cs_generate_variant:Nn
11916   \msg_new:nnnn
11917   { nnnV }
11918 \tl_set:Nn
11919   \l_tmpa_tl
11920   {
11921     You~may~need~to~run~TeX~with~the~--shell-escape~or~the~
11922     --enable-write18~flag,~or~write~shell_escape=t~in~the~
11923     texmf.cnf~file.
11924   }

```



```

11925 \msg_new:nnnV
11926   { markdown }
11927   { restricted-shell-access }
11928   { Shell~escape~is~restricted }
11929   \l_tmpa_tl
11930 \msg_new:nnnV
11931   { markdown }
11932   { disabled-shell-access }
11933   { Shell~escape~is~disabled }
11934   \l_tmpa_tl
11935 \ExplSyntaxOff

```

### 3.2.6 Typesetting Markdown

The `\markdownInput` macro uses an implementation of the `\markdownLuaExecute` macro to convert the contents of the file whose filename it has received as its single argument from markdown to plain TeX.

```

11936 \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.

```

11937   \catcode`\|=0%
11938   \catcode`\|=12%
11939   \catcode`\&=6%
11940   |gdef|markdownInput#1{%

```

Change the category code of the percent sign (%) to other, so that a user of the `hybrid` Lua option or a malevolent actor can't produce TeX comments in the plain TeX output of the Markdown package.

```

11941     |begingroup
11942     |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.

```

11943     |catcode`|#=12

```

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

```

11944     |markdownIfOption{frozenCache}{%
11945       |ifnum|markdownOptionFrozenCacheCounter=0|relax
11946         |markdownInfo{Reading frozen cache from
11947           "|markdownOptionFrozenCacheFileName"}%
11948         |input|markdownOptionFrozenCacheFileName|relax
11949       |fi
11950       |markdownInfo{Including markdown document number

```

```

11951      "|the|markdownOptionFrozenCacheCounter" from frozen cache}%
11952      |csname markdownFrozenCache|the|markdownOptionFrozenCacheCounter|endcsname
11953      |global|advance|markdownOptionFrozenCacheCounter by 1|relax
11954    }{%
11955      |markdownInfo{Including markdown document "&1"}%

```

Attempt to open the markdown document to record it in the `.log` and `.fls` files. This allows external programs such as L<sup>A</sup>T<sub>E</sub>X<sub>M</sub>k to track changes to the markdown document.

```

11956      |openin|markdownInputFileStream&1
11957      |closein|markdownInputFileStream
11958      |markdownPrepareLuaOptions
11959      |markdownLuaExecute{%
11960        |markdownPrepare
11961        local file = assert(io.open("&1", "r"),
11962          [[Could not open file "&1" for reading]])
11963        local input = assert(file:read("*a"))
11964        assert(file:close())
11965        print(convert(input))
11966      |markdownCleanup}%

```

If we are finalizing the frozen cache, increment `frozenCacheCounter`.

```

11967      |markdownIfOption{finalizeCache}{%
11968        |global|advance|markdownOptionFrozenCacheCounter by 1|relax}{}%
11969    }%
11970    |endgroup
11971  }%
11972 |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 T<sub>E</sub>X to execute a T<sub>E</sub>X document in the middle of a markdown document fragment.

```

11973 \gdef\markdownEscape#1{%
11974   \catcode`\%=14\relax
11975   \catcode`\#=6\relax
11976   \input #1\relax
11977   \catcode`\%=12\relax
11978   \catcode`\#=12\relax
11979 }%

```

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

The L<sup>A</sup>T<sub>E</sub>X implemenation makes use of the fact that, apart from some subtle differences, L<sup>A</sup>T<sub>E</sub>X implements the majority of the plain T<sub>E</sub>X format [11, Section 9]. As a consequence, we can directly reuse the existing plain T<sub>E</sub>X implementation.

```

11980 \def\markdownVersionSpace{ }%

```

```

11981 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
11982 \markdownVersion\markdownVersionSpace markdown renderer]%

```

### 3.3.1 Logging Facilities

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

### 3.3.2 Typesetting Markdown

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

```

11983 \let\markdownInputPlainTeX\markdownInput
11984 \renewcommand\markdownInput[2][]{%
11985   \begingroup
11986     \markdownSetup{#1}%
11987     \markdownInputPlainTeX{#2}%
11988   \endgroup}%

```

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

```

11989 \ExplSyntaxOn
11990 \renewenvironment
11991   { markdown }
11992   {

```

In our implementation of the `markdown` L<sup>A</sup>T<sub>E</sub>X environment, we want to distinguish between the following two cases:

|                                               |                               |
|-----------------------------------------------|-------------------------------|
| <code>\begin{markdown} [smartEllipses]</code> | <code>\begin{markdown}</code> |
| <code>% This is an optional argument ^</code> | <code>[smartEllipses]</code>  |
| <code>% ...</code>                            | <code>% ^ This is link</code> |
| <code>\end{markdown}</code>                   | <code>\end{markdown}</code>   |

Therefore, we cannot use the built-in L<sup>A</sup>T<sub>E</sub>X support for environments with optional arguments or packages such as `xparse`. Instead, we must read the optional argument manually and prevent reading past the end of a line.

To prevent reading past the end of a line when looking for the optional argument of the `markdown` L<sup>A</sup>T<sub>E</sub>X environment and accidentally tokenizing markdown text, we change the category code of carriage return (`\r`, ASCII character 13 in decimal) from 5 (end of line).

While any category code other than 5 (end of line) would work, we switch to the category 13 (active), which is also used by the `\markdownReadAndConvert` macro.

This is necessary if we read until the end of a line, because then the carriage return character will be produced by  $\text{\TeX}$  via the `\endlinechar` plain  $\text{\TeX}$  macro and it needs to have the correct category code, so that `\markdownReadAndConvert` processes it correctly.

```
11993 \group_begin:
11994 \char_set_catcode_active:n { 13 }
```

To prevent doubling the hash signs (`#`, ASCII code 35 in decimal), we switch its category from 6 (parameter) to 12 (letter).

```
11995 \char_set_catcode_letter:n { 35 }
```

After we have matched the opening `[` that begins the optional argument, we accept carriage returns as well.

```
11996 \peek_regex_replace_once:nnF
11997 { \ *\[\r*([^\r]*)\[^\r]* }
11998 {
```

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.

```
11999 \c { group_end: }
12000 \c { tl_set_rescan:Nnn } \c { l_tmpa_tl } { } { \1 }
```

Then, we pass the retokenized content to the `\markdownSetup` macro.

```
12001 \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.

```
12002 \c { markdownReadAndConvert@markdown } { }
12003 }
12004 {
12005 \group_end:
12006 \markdownReadAndConvert@markdown { }
12007 }
12008 }
12009 { \markdownEnd }
12010 \renewenvironment
12011 { markdown* }
12012 [ 1 ]
12013 {
12014 \msg_warning:nnn
12015 { markdown }
12016 { latex-markdown-star-deprecated }
12017 { #1 }
```

```

12018 \@@_setup:n
12019 { #1 }
12020 \markdownReadAndConvert@markdown *
12021 }
12022 { \markdownEnd }
12023 \msg_new:nnn
12024 { markdown }
12025 { latex-markdown-star-deprecated }
12026 {
12027   The~markdown*~LaTeX~environment~has~been~deprecated~and~will~
12028   be~removed~in~the~next~major~version~of~the~Markdown~package.
12029 }
12030 \ExplSyntaxOff
12031 \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*.

```

12032 \catcode`\|=0\catcode`\<=1\catcode`\>=2%
12033 \catcode`\|=12\catcode`\{=12\catcode`\}=12%
12034 |gdef|markdownReadAndConvert@markdown#1<%
12035   |markdownReadAndConvert<\end{markdown#1}>%
12036   <|end<markdown#1>>>%
12037 |endgroup

```

### 3.3.3 Options

The supplied package options are processed using the `\markdownSetup` macro.

```

12038 \DeclareOption*{%
12039   \expandafter\markdownSetup\expandafter{\CurrentOption}}%
12040 \ProcessOptions\relax

```

### 3.3.4 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  $\text{\LaTeX}$  themes provided with the Markdown package.

```

12041 \ExplSyntaxOn
12042 \cs_gset:Nn
12043 \@@_load_theme:nn
12044 {

```

If the Markdown package has already been loaded, determine whether a file named `markdowntheme<munged theme name>.sty` exists and whether we are still in the preamble.

```

12045     \ifmarkdownLaTeXLoaded
12046     \ifx\@onlypreamble\@notprerr

```

If both conditions are true does, end with an error, since we cannot load  $\text{\LaTeX}$  themes after the preamble. Otherwise, try loading a plain  $\text{\TeX}$  theme instead.

```

12047         \file_if_exist:nTF
12048         { markdown theme #2.sty }
12049         {
12050             \msg_error:nnn
12051             { markdown }
12052             { latex-theme-after-preamble }
12053             { #1 }
12054         }
12055         {
12056             \@@_plain_tex_load_theme:nn
12057             { #1 }
12058             { #2 }
12059         }
12060     \else

```

If the Markdown package has already been loaded but we are still in the preamble, load a  $\text{\LaTeX}$  theme if it exists or load a plain  $\text{\TeX}$  theme otherwise.

```

12061         \file_if_exist:nTF
12062         { markdown theme #2.sty }
12063         {
12064             \msg_info:nnn
12065             { markdown }
12066             { loading-latex-theme }
12067             { #1 }
12068             \RequirePackage
12069             { markdown theme #2 }
12070         }
12071         {
12072             \@@_plain_tex_load_theme:nn
12073             { #1 }
12074             { #2 }
12075         }
12076     \fi
12077 \else

```

If the Markdown package has not yet been loaded, postpone the loading until the Markdown package has finished loading.

```

12078     \msg_info:nnn
12079     { markdown }
12080     { theme-loading-postponed }
12081     { #1 }
12082 \AtEndOfPackage
12083 {

```

```

12084         \@@_load_theme:nn
12085         { #1 }
12086         { #2 }
12087     }
12088     \fi
12089 }
12090 \msg_new:nnn
12091 { markdown }
12092 { theme-loading-postponed }
12093 {
12094     Postponing~loading~Markdown~theme~#1~until~
12095     Markdown~package~has~finished~loading
12096 }
12097 \msg_new:nnn
12098 { markdown }
12099 { loading-latex-theme }
12100 { Loading~LaTeX~Markdown~theme~#1 }
12101 \cs_generate_variant:Nn
12102 \msg_new:nnnn
12103 { nnVV }
12104 \tl_set:Nn
12105 \l_tmpa_tl
12106 { Cannot~load~LaTeX~Markdown~theme~#1~after~ }
12107 \tl_put_right:NV
12108 \l_tmpa_tl
12109 \c_backslash_str
12110 \tl_put_right:Nn
12111 \l_tmpa_tl
12112 { begin{document} }
12113 \tl_set:Nn
12114 \l_tmpb_tl
12115 { Load~Markdown~theme~#1~before~ }
12116 \tl_put_right:NV
12117 \l_tmpb_tl
12118 \c_backslash_str
12119 \tl_put_right:Nn
12120 \l_tmpb_tl
12121 { begin{document} }
12122 \msg_new:nnVV
12123 { markdown }
12124 { latex-theme-after-preamble }
12125 \l_tmpa_tl
12126 \l_tmpb_tl
12127 \ExplSyntaxOff

```

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

```

12128 \markdownSetup{fencedCode}%

```

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

```
12129 \RequirePackage{ifthen,grffile}
```

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

```
12130 \let\markdown@witiko@dot@oldRendererInputFencedCodePrototype
```

```
12131 \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:

```
12132 \renewcommand\markdownRendererInputFencedCodePrototype[3]{%
12133   \def\next##1 ##2\relax{%
12134     \ifthenelse{\equal{##1}{dot}}{%
12135       \markdownIfOption{frozenCache}{}{%
12136         \immediate\write18{%
12137           if ! test -e #1.pdf.source || ! diff #1 #1.pdf.source;
12138         then
12139           dot -Tpdf -o #1.pdf #1;
12140           cp #1 #1.pdf.source;
12141         fi}}%
12142     \next##1 ##2\relax}%
12143 }
```

We include the typeset image using the image token renderer:

```
12142 \markdownRendererImage{Graphviz image}{#1.pdf}{#1.pdf}{##2}%
12143 }
```

If the infostring does not start with `dot ...`, we use the previous definition of the fenced code token renderer prototype:

```
12144 \markdown@witiko@dot@oldRendererInputFencedCodePrototype{#1}{#2}{#3}%
12145 }%
12146 }%
12147 \next#2 \relax}%
12148 }
```

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

```
12148 \let\markdown@witiko@graphicx@http@oldRendererImagePrototype
```

```
12149 \markdownRendererImagePrototype
```

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

```
12150 \RequirePackage{catchfile,grffile}
```

We define the `\markdown@witiko@graphicx@http@counter` counter to enumerate the images for caching and the `\markdown@witiko@graphicx@http@filename` command, which will store the pathname of the file containing the pathname of the downloaded image file.

```
12151 \newcount\markdown@witiko@graphicx@http@counter
```

```
12152 \markdown@witiko@graphicx@http@counter=0
```

```
12153 \newcommand\markdown@witiko@graphicx@http@filename{%
```

```
12154   \markdownOptionCacheDir/witiko_graphicx_http%
```



```
12155 .\the\markdown@witiko@graphicx@http@counter}%
```

We define the `\markdown@witiko@graphicx@http@download` command, which will receive two arguments that correspond to the URL of the online image and to the pathname, where the online image should be downloaded. The command will produce a shell command that tries to download the online image to the pathname.

```
12156 \newcommand\markdown@witiko@graphicx@http@download[2]{%
12157   wget -O #2 #1 || curl --location -o #2 #1 || rm -f #2}
```

We locally swap the category code of the percentage sign with the line feed control character, so that we can use percentage signs in the shell code:

```
12158 \begingroup
12159 \catcode`\%=12
12160 \catcode`\^^A=14
```

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

```
12161 \global\def\markdownRendererImagePrototype#1#2#3#4{^^A
12162   \begingroup
12163   \edef\filename{\markdown@witiko@graphicx@http@filename}^^A
```

The image will be downloaded only if the image URL has the http or https protocols and the `frozenCache` plain TeX option is disabled:

```
12164   \markdownIfOption{frozenCache}{-}{^^A
12165     \immediate\write18{^^A
12166       mkdir -p "\markdownOptionCacheDir";
12167       if printf '%s' "#3" | grep -q -E '^https?:';
12168       then
```

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

```
12169       OUTPUT_PREFIX="\markdownOptionCacheDir";
12170       OUTPUT_BODY="$(printf '%s' '#3' | md5sum | cut -d' ' -f1)";
12171       OUTPUT_SUFFIX="$(printf '%s' '#3' | sed 's/.*[.]//')";
12172       OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";
```

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

```
12173       if ! [ -e "$OUTPUT" ];
12174       then
12175         \markdown@witiko@graphicx@http@download{'#3'}{"$OUTPUT"};
12176         printf '%s' "$OUTPUT" > "\filename";
12177       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:

```
12178       else
12179         printf '%s' '#3' > "\filename";
12180       fi}}^^A
```

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

```

12181 \CatchFileDef{\filename}{\filename}{\endlinechar=-1}^^A
12182 \markdown@witiko@graphicx@http@oldRendererImagePrototype^^A
12183 {#1}{#2}{\filename}{#4}^^A
12184 \endgroup
12185 \global\advance\markdown@witiko@graphicx@http@counter by 1\relax}^^A
12186 \endgroup

```

The `witiko/markdown/defaults` L<sup>A</sup>T<sub>E</sub>X theme provides default definitions for token renderer prototypes. First, the L<sup>A</sup>T<sub>E</sub>X theme loads the plain T<sub>E</sub>X theme with the default definitions for plain T<sub>E</sub>X:

```

12187 \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.

### 3.3.5 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.

```

12188 \markdownIfOption{plain}{\iffalse}{\iftrue}

```

If either the `tightLists` or the `fancyLists` Lua option is enabled and the current document class is not beamer, then load the paralist package.

```

12189 \@ifclassloaded{beamer}{}{}%
12190 \markdownIfOption{tightLists}{\RequirePackage{paralist}}{}%
12191 \markdownIfOption{fancyLists}{\RequirePackage{paralist}}{}%
12192 }

```

If we loaded the paralist package, define the respective renderer prototypes to make use of the capabilities of the package. Otherwise, define the renderer prototypes to fall back on the corresponding renderers for the non-tight lists.

```

12193 \ExplSyntaxOn
12194 \@ifpackageloaded{paralist}{
12195   \tl_new:N
12196     \l_@@_latex_fancy_list_item_label_number_style_tl
12197   \tl_new:N
12198     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
12199   \cs_new:Nn
12200     \@@_latex_fancy_list_item_label_number:nn
12201   {
12202     \str_case:nn
12203       { #1 }
12204       {
12205         { Decimal } { #2 }
12206         { LowerRoman } { \int_to_roman:n { #2 } }

```

```

12207         { UpperRoman } { \int_to_Roman:n { #2 } }
12208         { LowerAlpha } { \int_to_alph:n { #2 } }
12209         { UpperAlpha } { \int_to_Alph:n { #2 } }
12210     }
12211 }
12212 \cs_new:Nn
12213   \@@_latex_fancy_list_item_label_delimiter:n
12214 {
12215   \str_case:nn
12216     { #1 }
12217     {
12218       { Default } { . }
12219       { OneParen } { ) }
12220       { Period } { . }
12221     }
12222 }
12223 \cs_new:Nn
12224   \@@_latex_fancy_list_item_label:nnn
12225 {
12226   \@@_latex_fancy_list_item_label_number:nn
12227     { #1 }
12228     { #3 }
12229   \@@_latex_fancy_list_item_label_delimiter:n
12230     { #2 }
12231 }
12232 \cs_new:Nn
12233   \@@_latex_paralist_style:nn
12234 {
12235   \str_case:nn
12236     { #1 }
12237     {
12238       { Decimal } { 1 }
12239       { LowerRoman } { i }
12240       { UpperRoman } { I }
12241       { LowerAlpha } { a }
12242       { UpperAlpha } { A }
12243     }
12244   \@@_latex_fancy_list_item_label_delimiter:n
12245     { #2 }
12246 }
12247 \markdownSetup{rendererPrototypes={

```

Make tight bullet lists a little less compact by adding extra vertical space above and below them.

```

12248   ulBeginTight = {%
12249     \group_begin:
12250     \pltopsep=\topsep

```

```

12251     \plpartopsep=\partopsep
12252     \begin{compactitem}
12253 },
12254 ulEndTight = {
12255     \end{compactitem}
12256     \group_end:
12257 },
12258 fancyOlBegin = {
12259     \group_begin:
12260     \tl_set:Nn
12261       \l_@@_latex_fancy_list_item_label_number_style_tl
12262       { #1 }
12263     \tl_set:Nn
12264       \l_@@_latex_fancy_list_item_label_delimiter_style_tl
12265       { #2 }
12266     \@@_if_option:nTF
12267       { startNumber }
12268       {
12269         \tl_set:Nn
12270           \l_tmpa_tl
12271           { \begin{enumerate} }
12272       }
12273       {
12274         \tl_set:Nn
12275           \l_tmpa_tl
12276           { \begin{enumerate}[ ] }
12277         \tl_put_right:Nx
12278           \l_tmpa_tl
12279           { \@@_latex_paralist_style:nn { #1 } { #2 } }
12280         \tl_put_right:Nn
12281           \l_tmpa_tl
12282           { ] }
12283       }
12284     \tl_use:N
12285     \l_tmpa_tl
12286 },
12287 fancyOlEnd = {
12288     \end{enumerate}
12289     \group_end:
12290 },

```

Make tight ordered lists a little less compact by adding extra vertical space above and below them.

```

12291 olBeginTight = {%
12292     \group_begin:
12293     \plpartopsep=\partopsep
12294     \pltopsep=\topsep

```

```

12295     \begin{compactenum}
12296 },
12297 olEndTight = {
12298     \end{compactenum}
12299     \group_end:
12300 },
12301 fancyOlBeginTight = {
12302     \group_begin:
12303     \tl_set:Nn
12304         \l_@@_latex_fancy_list_item_label_number_style_tl
12305         { #1 }
12306     \tl_set:Nn
12307         \l_@@_latex_fancy_list_item_label_delimiter_style_tl
12308         { #2 }
12309     \tl_set:Nn
12310         \l_tmpa_tl
12311         {
12312             \plpartopsep=\partopsep
12313             \pltopsep=\topsep
12314         }
12315     \@@_if_option:nTF
12316         { startNumber }
12317         {
12318             \tl_put_right:Nn
12319                 \l_tmpa_tl
12320                 { \begin{compactenum} }
12321         }
12322         {
12323             \tl_put_right:Nn
12324                 \l_tmpa_tl
12325                 { \begin{compactenum}[ ] }
12326             \tl_put_right:Nx
12327                 \l_tmpa_tl
12328                 { \@@_latex_paralist_style:nn { #1 } { #2 } }
12329             \tl_put_right:Nn
12330                 \l_tmpa_tl
12331                 { ] }
12332         }
12333     \tl_use:N
12334         \l_tmpa_tl
12335 },
12336 fancyOlEndTight = {
12337     \end{compactenum}
12338     \group_end:
12339 },
12340 fancyOlItemWithNumber = {
12341     \item

```

```

12342     [
12343         \@@_latex_fancy_list_item_label:VVn
12344         \l_@@_latex_fancy_list_item_label_number_style_tl
12345         \l_@@_latex_fancy_list_item_label_delimiter_style_tl
12346         { #1 }
12347     ]
12348 },

```

Make tight definition lists a little less compact by adding extra vertical space above and below them.

```

12349     dlBeginTight = {
12350         \group_begin:
12351         \plpartopsep=\partopsep
12352         \pltopsep=\topsep
12353         \begin{compactdesc}
12354     },
12355     dlEndTight = {
12356         \end{compactdesc}
12357         \group_end:
12358     }}}
12359 \cs_generate_variant:Nn
12360 \@@_latex_fancy_list_item_label:nnn
12361 { VVn }
12362 }{
12363 \markdownSetup{rendererPrototypes={
12364     ulBeginTight = {\markdownRendererUlBegin},
12365     ulEndTight = {\markdownRendererUlEnd},
12366     fancyOlBegin = {\markdownRendererOlBegin},
12367     fancyOlEnd = {\markdownRendererOlEnd},
12368     olBeginTight = {\markdownRendererOlBegin},
12369     olEndTight = {\markdownRendererOlEnd},
12370     fancyOlBeginTight = {\markdownRendererOlBegin},
12371     fancyOlEndTight = {\markdownRendererOlEnd},
12372     dlBeginTight = {\markdownRendererDlBegin},
12373     dlEndTight = {\markdownRendererDlEnd}}}
12374 }
12375 \ExplSyntaxOff
12376 \RequirePackage{amsmath}

```

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

```

12377 \@ifpackageloaded{unicode-math}{
12378     \markdownSetup{rendererPrototypes={
12379         untickedBox = {\$ \mdlgwhtsquare$},
12380     }}
12381 }{
12382     \RequirePackage{amssymb}
12383     \markdownSetup{rendererPrototypes={

```

```

12384     untickedBox = {\square$},
12385   }}
12386 }
12387 \RequirePackage{csvsimple}
12388 \RequirePackage{fancyvrb}
12389 \RequirePackage{graphicx}
12390 \markdownSetup{rendererPrototypes={
12391   hardLineBreak = {\},
12392   leftBrace = {\textbraceleft},
12393   rightBrace = {\textbraceright},
12394   dollarSign = {\textdollar},
12395   underscore = {\textunderscore},
12396   circumflex = {\textasciicircum},
12397   backslash = {\textbackslash},
12398   tilde = {\textasciitilde},
12399   pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by T<sub>E</sub>X during the typesetting. Therefore, even if we don't know whether a span of text is part of math formula or not when we are parsing markdown,<sup>33</sup> we can reliably detect math mode inside the renderer.

Here, we will redefine the code span renderer prototype to typeset upright text in math formulae and typewriter text outside math formulae.

```

12400   codeSpan = {%
12401     \ifmmode
12402       \text{#1}%
12403     \else
12404       \texttt{#1}%
12405     \fi
12406   }}
12407 \ExplSyntaxOn
12408 \markdownSetup{
12409   rendererPrototypes = {
12410     contentBlock = {
12411       \str_case:nnF
12412         { #1 }
12413         {
12414           { csv }
12415           {
12416             \begin{table}
12417               \begin{center}
12418                 \csvautotabular{#3}
12419                 \end{center}

```

---

<sup>33</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.

```

12420             \tl_if_empty:nF
12421             { #4 }
12422             { \caption{#4} }
12423         \end{table}
12424     }
12425     { tex } { \markdownEscape{#3} }
12426 }
12427 { \markdownInput{#3} }
12428 },
12429 },
12430 }
12431 \ExplSyntaxOff
12432 \markdownSetup{rendererPrototypes={
12433     image = {%
12434         \begin{figure}%
12435         \begin{center}%
12436             \includegraphics[alt={#1}]{#3}%
12437         \end{center}%
12438         \ifx\empty#4\empty\else
12439             \caption{#4}%
12440         \fi
12441     \end{figure}},
12442     ulBegin = {\begin{itemize}},
12443     ulEnd = {\end{itemize}},
12444     olBegin = {\begin{enumerate}},
12445     olItem = {\item{}},
12446     olItemWithNumber = {\item[#1.]},
12447     olEnd = {\end{enumerate}},
12448     dlBegin = {\begin{description}},
12449     dlItem = {\item[#1]},
12450     dlEnd = {\end{description}},
12451     emphasis = {\emph{#1}},
12452     tickedTextBox = {\$\boxtimes$},
12453     halfTickedTextBox = {\$\boxdot$}}

```

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

```

12454 \ExplSyntaxOn
12455 \seq_new:N \l_@@_header_identifiers_seq
12456 \markdownSetup{
12457     rendererPrototypes = {
12458         headerAttributeContextBegin = {
12459             \seq_clear:N \l_@@_header_identifiers_seq
12460             \markdownSetup
12461             {
12462                 renderers = {
12463                     attributeIdentifier = {

```



```

12464         \seq_put_right:Nn
12465         \l_@@_header_identifiers_seq
12466         { ##1 }
12467     },
12468 },
12469 }
12470 },
12471 headerAttributeContextEnd = {
12472     \seq_map_inline:Nn
12473     \l_@@_header_identifiers_seq
12474     { \label { ##1 } }
12475 },
12476 },
12477 }
12478 \ExplSyntaxOff
12479 \markdownSetup{rendererPrototypes={
12480     superscript = {\textsuperscript{#1}},
12481     subscript = {\textsubscript{#1}},
12482     blockQuoteBegin = {\begin{quotation}},
12483     blockQuoteEnd = {\end{quotation}},
12484     inputVerbatim = {\VerbatimInput{#1}},
12485     thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
12486     note = {\footnote{#1}}}}

```

### 3.3.5.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

12487 \RequirePackage{ltxcmds}
12488 \ExplSyntaxOn
12489 \cs_gset:Npn
12490     \markdownRendererInputFencedCodePrototype#1#2#3
12491     {
12492         \tl_if_empty:nTF
12493             { #2 }
12494             { \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.

```

12495     {
12496         \regex_extract_once:nnN
12497             { \w* }
12498             { #2 }
12499             \l_tmpa_seq
12500         \seq_pop_left:NN
12501             \l_tmpa_seq
12502             \l_tmpa_tl

```

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

```

12503     \ltx@ifpackageloaded
12504     { minted }
12505     {
12506         \catcode`\#=6\relax
12507         \exp_args:NV
12508         \inputminted
12509         \l_tmpa_tl
12510         { #1 }
12511         \catcode`\#=12\relax
12512     }
12513     {

```

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

```

12514     \ltx@ifpackageloaded
12515     { listings }
12516     { \lstinputlisting[language=\l_tmpa_tl]{#1} }

```

When neither the listings package nor the minted package is loaded, act as though no infostring were given.

```

12517         { \markdownRendererInputFencedCode{#1}{}} }
12518     }
12519 }
12520 }
12521 \ExplSyntaxOff

```

Support the nesting of strong emphasis.

```

12522 \ExplSyntaxOn
12523 \def\markdownLATEXStrongEmphasis#1{%
12524     \str_if_in:NnTF
12525     \f@series
12526     { b }
12527     { \textnormal{#1} }
12528     { \textbf{#1} }
12529 }
12530 \ExplSyntaxOff
12531 \markdownSetup{rendererPrototypes={strongEmphasis={%
12532     \protect\markdownLATEXStrongEmphasis{#1}}}}

```

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

```

12533 \@ifundefined{chapter}{%
12534     \markdownSetup{rendererPrototypes = {
12535         headingOne = {\section{#1}},
12536         headingTwo = {\subsection{#1}},
12537         headingThree = {\subsubsection{#1}},
12538         headingFour = {\paragraph{#1}},
12539         headingFive = {\subparagraph{#1}}}}
12540 }{%
12541     \markdownSetup{rendererPrototypes = {
12542         headingOne = {\chapter{#1}},

```

```

12543     headingTwo = {\section{#1}},
12544     headingThree = {\subsection{#1}},
12545     headingFour = {\subsubsection{#1}},
12546     headingFive = {\paragraph{#1}},
12547     headingSix = {\subparagraph{#1}}}}
12548 }%

```

### 3.3.5.2 Tickboxes

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

```

12549 \markdownSetup{
12550   rendererPrototypes = {
12551     ulItem = {%
12552       \futurelet\markdownLaTeXCheckbox\markdownLaTeXUItem
12553     },
12554   },
12555 }
12556 \def\markdownLaTeXUItem{%
12557   \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
12558     \item[\markdownLaTeXCheckbox]%
12559     \expandafter\@gobble
12560   \else
12561     \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
12562       \item[\markdownLaTeXCheckbox]%
12563       \expandafter\expandafter\expandafter\@gobble
12564     \else
12565       \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
12566         \item[\markdownLaTeXCheckbox]%
12567         \expandafter\expandafter\expandafter\expandafter
12568         \expandafter\expandafter\expandafter\@gobble
12569       \else
12570         \item{}%
12571       \fi
12572     \fi
12573   \fi
12574 }

```

### 3.3.5.3 HTML elements

If the `html` option is enabled and we are using  $\text{\TeX}4\text{ht}$ <sup>34</sup>, we will pass HTML elements to the output HTML document unchanged.

```

12575 \@ifundefined{HCode}{}{
12576   \markdownSetup{
12577     rendererPrototypes = {
12578       inlineHtmlTag = {%

```

---

<sup>34</sup>See <https://tug.org/tex4ht/>.

```

12579     \ifvmode
12580     \IgnorePar
12581     \EndP
12582     \fi
12583     \HCode{#1}%
12584   },
12585   inputBlockHtmlElement = {%
12586     \ifvmode
12587     \IgnorePar
12588     \fi
12589     \EndP
12590     \special{t4ht*<#1}%
12591     \par
12592     \ShowPar
12593   },
12594 },
12595 }
12596 }

```

### 3.3.5.4 Citations

Here is a basic implementation for citations that uses the L<sup>A</sup>T<sub>E</sub>X `\cite` macro. There are also implementations that use the natbib `\citep`, and `\citet` macros, and the BibL<sup>A</sup>T<sub>E</sub>X `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```

12597 \newcount\markdownLaTeXCitationsCounter
12598
12599 % Basic implementation
12600 \RequirePackage{gobble}
12601 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
12602   \advance\markdownLaTeXCitationsCounter by 1\relax
12603   \ifx\relax#4\relax
12604     \ifx\relax#5\relax
12605       \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
12606         \cite{#1#2#6}% Without prenotes and postnotes, just accumulate cites
12607         \expandafter\expandafter\expandafter
12608         \expandafter\expandafter\expandafter\expandafter
12609         \@gobblethree
12610       \fi
12611     \else% Before a postnote (#5), dump the accumulator
12612       \ifx\relax#1\relax\else
12613         \cite{#1}%
12614       \fi
12615       \cite[#5]{#6}%
12616     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
12617     \else
12618       \expandafter\expandafter\expandafter

```

```

12619     \expandafter\expandafter\expandafter\expandafter
12620     \expandafter\expandafter\expandafter
12621     \expandafter\expandafter\expandafter\expandafter
12622     \markdownLaTeXBasicCitations
12623     \fi
12624     \expandafter\expandafter\expandafter
12625     \expandafter\expandafter\expandafter\expandafter{%
12626     \expandafter\expandafter\expandafter
12627     \expandafter\expandafter\expandafter\expandafter}%
12628     \expandafter\expandafter\expandafter
12629     \expandafter\expandafter\expandafter\expandafter{%
12630     \expandafter\expandafter\expandafter
12631     \expandafter\expandafter\expandafter\expandafter}%
12632     \expandafter\expandafter\expandafter
12633     \@gobblethree
12634     \fi
12635 \else% Before a prenote (#4), dump the accumulator
12636     \ifx\relax#1\relax\else
12637         \cite{#1}%
12638     \fi
12639     \ifnum\markdownLaTeXCitationsCounter>1\relax
12640         \space % Insert a space before the prenote in later citations
12641     \fi
12642     #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
12643     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
12644     \else
12645         \expandafter\expandafter\expandafter
12646         \expandafter\expandafter\expandafter\expandafter
12647         \markdownLaTeXBasicCitations
12648     \fi
12649     \expandafter\expandafter\expandafter{%
12650     \expandafter\expandafter\expandafter}%
12651     \expandafter\expandafter\expandafter{%
12652     \expandafter\expandafter\expandafter}%
12653     \expandafter
12654     \@gobblethree
12655     \fi\markdownLaTeXBasicCitations{#1#2#6},}
12656 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
12657
12658 % Natbib implementation
12659 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
12660     \advance\markdownLaTeXCitationsCounter by 1\relax
12661     \ifx\relax#3\relax
12662         \ifx\relax#4\relax
12663             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
12664                 \citep{#1,#5}% Without prenotes and postnotes, just accumulate cites
12665             \expandafter\expandafter\expandafter

```

```

12666         \expandafter\expandafter\expandafter\expandafter
12667         \@gobbletwo
12668     \fi
12669 \else% Before a postnote (#4), dump the accumulator
12670     \ifx\relax#1\relax\else
12671         \citep{#1}%
12672     \fi
12673     \citep[] [#4]{#5}%
12674     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
12675     \else
12676         \expandafter\expandafter\expandafter
12677         \expandafter\expandafter\expandafter\expandafter
12678         \expandafter\expandafter\expandafter
12679         \expandafter\expandafter\expandafter\expandafter
12680         \markdownLaTeXNatbibCitations
12681     \fi
12682     \expandafter\expandafter\expandafter
12683     \expandafter\expandafter\expandafter\expandafter{%
12684     \expandafter\expandafter\expandafter
12685     \expandafter\expandafter\expandafter\expandafter}%
12686     \expandafter\expandafter\expandafter
12687     \@gobbletwo
12688 \fi
12689 \else% Before a prenote (#3), dump the accumulator
12690     \ifx\relax#1\relax\relax\else
12691         \citep{#1}%
12692     \fi
12693     \citep[#3] [#4]{#5}%
12694     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
12695     \else
12696         \expandafter\expandafter\expandafter
12697         \expandafter\expandafter\expandafter\expandafter
12698         \markdownLaTeXNatbibCitations
12699     \fi
12700     \expandafter\expandafter\expandafter{%
12701     \expandafter\expandafter\expandafter}%
12702     \expandafter
12703     \@gobbletwo
12704 \fi\markdownLaTeXNatbibCitations{#1,#5}}
12705 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
12706     \advance\markdownLaTeXCitationsCounter by 1\relax
12707     \ifx\relax#3\relax
12708     \ifx\relax#4\relax
12709         \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
12710             \citet{#1,#5}% Without prenotes and postnotes, just accumulate cites
12711             \expandafter\expandafter\expandafter
12712             \expandafter\expandafter\expandafter\expandafter

```

```

12713         \@gobbletwo
12714     \fi
12715 \else% After a prenote or a postnote, dump the accumulator
12716 \ifx\relax#1\relax\else
12717     \citet{#1}%
12718 \fi
12719 , \citet[#3][#4]{#5}%
12720 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
12721     ,
12722 \else
12723     \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
12724     ,
12725 \fi
12726 \fi
12727 \expandafter\expandafter\expandafter
12728 \expandafter\expandafter\expandafter\expandafter
12729 \markdownLaTeXNatbibTextCitations
12730 \expandafter\expandafter\expandafter
12731 \expandafter\expandafter\expandafter\expandafter{%
12732 \expandafter\expandafter\expandafter
12733 \expandafter\expandafter\expandafter\expandafter}%
12734 \expandafter\expandafter\expandafter
12735 \@gobbletwo
12736 \fi
12737 \else% After a prenote or a postnote, dump the accumulator
12738 \ifx\relax#1\relax\relax\else
12739     \citet{#1}%
12740 \fi
12741 , \citet[#3][#4]{#5}%
12742 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
12743     ,
12744 \else
12745     \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
12746     ,
12747 \fi
12748 \fi
12749 \expandafter\expandafter\expandafter
12750 \markdownLaTeXNatbibTextCitations
12751 \expandafter\expandafter\expandafter{%
12752 \expandafter\expandafter\expandafter}%
12753 \expandafter
12754 \@gobbletwo
12755 \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
12756
12757 % BibLaTeX implementation
12758 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
12759     \advance\markdownLaTeXCitationsCounter by 1\relax

```

```

12760 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
12761 \autocites#1[#3][#4]{#5}%
12762 \expandafter\@gobbletwo
12763 \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}
12764 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
12765 \advance\markdownLaTeXCitationsCounter by 1\relax
12766 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
12767 \textcites#1[#3][#4]{#5}%
12768 \expandafter\@gobbletwo
12769 \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}
12770
12771 \markdownSetup{rendererPrototypes = {
12772 cite = {%
12773 \markdownLaTeXCitationsCounter=1%
12774 \def\markdownLaTeXCitationsTotal{#1}%
12775 \@ifundefined{autocites}{%
12776 \@ifundefined{citep}{%
12777 \expandafter\expandafter\expandafter
12778 \markdownLaTeXBasicCitations
12779 \expandafter\expandafter\expandafter{%
12780 \expandafter\expandafter\expandafter}%
12781 \expandafter\expandafter\expandafter{%
12782 \expandafter\expandafter\expandafter}%
12783 }{%
12784 \expandafter\expandafter\expandafter
12785 \markdownLaTeXNatbibCitations
12786 \expandafter\expandafter\expandafter{%
12787 \expandafter\expandafter\expandafter}%
12788 }%
12789 }{%
12790 \expandafter\expandafter\expandafter
12791 \markdownLaTeXBibLaTeXCitations
12792 \expandafter{\expandafter}%
12793 }},
12794 textCite = {%
12795 \markdownLaTeXCitationsCounter=1%
12796 \def\markdownLaTeXCitationsTotal{#1}%
12797 \@ifundefined{autocites}{%
12798 \@ifundefined{citep}{%
12799 \expandafter\expandafter\expandafter
12800 \markdownLaTeXBasicTextCitations
12801 \expandafter\expandafter\expandafter{%
12802 \expandafter\expandafter\expandafter}%
12803 \expandafter\expandafter\expandafter{%
12804 \expandafter\expandafter\expandafter}%
12805 }{%
12806 \expandafter\expandafter\expandafter

```



```

12807         \markdownLaTeXNatbibTextCitations
12808         \expandafter\expandafter\expandafter{%
12809         \expandafter\expandafter\expandafter}%
12810     }%
12811 }{%
12812     \expandafter\expandafter\expandafter
12813     \markdownLaTeXBibLaTeXTextCitations
12814     \expandafter{\expandafter}%
12815 }}}}
```

### 3.3.5.5 Links

Here is an implementation for hypertext links and relative references.

```

12816 \RequirePackage{url}
12817 \RequirePackage{expl3}
12818 \ExplSyntaxOn
12819 \def\markdownRendererLinkPrototype#1#2#3#4{
12820     \tl_set:Nn \l_tmpa_tl { #1 }
12821     \tl_set:Nn \l_tmpb_tl { #2 }
12822     \bool_set:Nn
12823         \l_tmpa_bool
12824     {
12825         \tl_if_eq_p:NN
12826             \l_tmpa_tl
12827             \l_tmpb_tl
12828     }
12829     \tl_set:Nn \l_tmpa_tl { #4 }
12830     \bool_set:Nn
12831         \l_tmpb_bool
12832     {
12833         \tl_if_empty_p:N
12834             \l_tmpa_tl
12835     }
```

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.

```

12836     \bool_if:nTF
12837     {
12838         \l_tmpa_bool && \l_tmpb_bool
12839     }
12840     {
12841         \markdownLaTeXRendererAutolink { #2 } { #3 }
12842     }{
12843         \markdownLaTeXRendererDirectOrIndirectLink { #1 } { #2 } { #3 } { #4 }
12844     }
12845 }
12846 \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.

```

12847 \tl_set:Nn
12848   \l_tmpa_tl
12849   { #2 }
12850 \tl_trim_spaces:N
12851   \l_tmpa_tl
12852 \tl_set:Nx
12853   \l_tmpb_tl
12854   {
12855     \tl_range:Nnn
12856       \l_tmpa_tl
12857       { 1 }
12858       { 1 }
12859   }
12860 \str_if_eq:NNTF
12861   \l_tmpb_tl
12862   \c_hash_str
12863   {
12864     \tl_set:Nx
12865       \l_tmpb_tl
12866       {
12867         \tl_range:Nnn
12868           \l_tmpa_tl
12869           { 2 }
12870           { -1 }
12871       }
12872     \exp_args:NV
12873       \ref
12874       \l_tmpb_tl
12875   }{
12876     \url { #2 }
12877   }
12878 }
12879 \ExplSyntaxOff
12880 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
12881   #1\footnote{\ifx\empty#4\empty\else#4: \fi\url{#3}}}
```

### 3.3.5.6 Tables

Here is a basic implementation of tables. If the booktabs package is loaded, then it is used to produce horizontal lines.

```

12882 \newcount\markdownLaTeXRowCount
12883 \newcount\markdownLaTeXRowTotal
12884 \newcount\markdownLaTeXColumnCounter
12885 \newcount\markdownLaTeXColumnTotal
12886 \newtoks\markdownLaTeXTable
```

```

12887 \newtoks\markdownLaTeXTableAlignment
12888 \newtoks\markdownLaTeXTableEnd
12889 \AtBeginDocument{%
12890   \@ifpackageloaded{booktabs}{%
12891     \def\markdownLaTeXTopRule{\toprule}%
12892     \def\markdownLaTeXMidRule{\midrule}%
12893     \def\markdownLaTeXBottomRule{\bottomrule}%
12894   }{%
12895     \def\markdownLaTeXTopRule{\hline}%
12896     \def\markdownLaTeXMidRule{\hline}%
12897     \def\markdownLaTeXBottomRule{\hline}%
12898   }%
12899 }
12900 \markdownSetup{rendererPrototypes={
12901   table = {%
12902     \markdownLaTeXTable={}%
12903     \markdownLaTeXTableAlignment={}%
12904     \markdownLaTeXTableEnd={%
12905       \markdownLaTeXBottomRule
12906       \end{tabular}}%
12907     \ifx\empty#1\empty\else
12908       \addto@hook\markdownLaTeXTable{%
12909         \begin{table}
12910         \centering}%
12911       \addto@hook\markdownLaTeXTableEnd{%
12912         \caption{#1}
12913         \end{table}}%
12914     \fi
12915     \addto@hook\markdownLaTeXTable{\begin{tabular}}%
12916     \markdownLaTeXRowCounter=0%
12917     \markdownLaTeXRowTotal=#2%
12918     \markdownLaTeXColumnTotal=#3%
12919     \markdownLaTeXRenderTableRow
12920   }
12921 }}
12922 \def\markdownLaTeXRenderTableRow#1{%
12923   \markdownLaTeXColumnCounter=0%
12924   \ifnum\markdownLaTeXRowCounter=0\relax
12925     \markdownLaTeXReadAlignments#1%
12926     \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
12927       \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
12928         \the\markdownLaTeXTableAlignment}}%
12929     \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
12930   \else
12931     \markdownLaTeXRenderTableCell#1%
12932   \fi
12933   \ifnum\markdownLaTeXRowCounter=1\relax

```

```

12934 \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
12935 \fi
12936 \advance\markdownLaTeXRowCount by 1\relax
12937 \ifnum\markdownLaTeXRowCount>\markdownLaTeXRowTotal\relax
12938 \the\markdownLaTeXTable
12939 \the\markdownLaTeXTableEnd
12940 \expandafter\@gobble
12941 \fi\markdownLaTeXRenderTableRow}
12942 \def\markdownLaTeXReadAlignments#1{%
12943 \advance\markdownLaTeXColumnCounter by 1\relax
12944 \if#1d%
12945 \addto@hook\markdownLaTeXTableAlignment{1}%
12946 \else
12947 \addto@hook\markdownLaTeXTableAlignment{#1}%
12948 \fi
12949 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
12950 \expandafter\@gobble
12951 \fi\markdownLaTeXReadAlignments}
12952 \def\markdownLaTeXRenderTableCell#1{%
12953 \advance\markdownLaTeXColumnCounter by 1\relax
12954 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
12955 \addto@hook\markdownLaTeXTable{#1&}%
12956 \else
12957 \addto@hook\markdownLaTeXTable{#1\\}%
12958 \expandafter\@gobble
12959 \fi\markdownLaTeXRenderTableCell}

```

### 3.3.5.7 Line Blocks

Here is a basic implementation of line blocks. If the verse package is loaded, then it is used to produce the verses.

```

12960
12961 \markdownIfOption{lineBlocks}{%
12962 \RequirePackage{verse}
12963 \markdownSetup{rendererPrototypes={
12964 lineBlockBegin = {%
12965 \begingroup
12966 \def\markdownRendererHardLineBreak{\\}%
12967 \begin{verse}%
12968 },
12969 lineBlockEnd = {%
12970 \end{verse}%
12971 \endgroup
12972 },
12973 }}
12974 }{}
12975

```

### 3.3.5.8 YAML Metadata

The default setup of YAML metadata will invoke the `\title`, `\author`, and `\date` macros when scalar values for keys that correspond to the `title`, `author`, and `date` relative wildcards are encountered, respectively.

```
12976 \ExplSyntaxOn
12977 \keys_define:nn
12978 { markdown/jekyllData }
12979 {
12980   author .code:n = { \author{#1} },
12981   date   .code:n = { \date{#1}   },
12982   title  .code:n = { \title{#1}  },
12983 }
```

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.

```
12984 \markdownSetup{
12985   rendererPrototypes = {
12986     jekyllDataEnd = {
12987       \AddToHook{begindocument/end}{\maketitle}
12988     },
12989   },
12990 }
12991 \ExplSyntaxOff
```

### 3.3.5.9 Strike-Through

If the `strikeThrough` option is enabled, we will load the `soulutf8` package and use it to implement strike-throughs.

```
12992 \markdownIfOption{strikeThrough}{%
12993   \RequirePackage{soulutf8}%
12994   \markdownSetup{
12995     rendererPrototypes = {
12996       strikeThrough = {%
12997         \st{#1}%
12998       },
12999     }
13000   }
13001 }{}
```

### 3.3.5.10 Marked Text

If the `mark` option is enabled, we will load the `soulutf8` package and use it to implement marked text.

```
13002 \markdownIfOption{mark}{%
13003   \RequirePackage{soulutf8}%
```

```

13004 \markdownSetup{
13005     rendererPrototypes = {
13006         mark = {%
13007             \hl{#1}%
13008         },
13009     }
13010 }
13011 }{}

```

### 3.3.5.11 Image Attributes

If the `linkAttributes` option is enabled, we will load the `graphicx` package. Furthermore, in image attribute contexts, we will make attributes in the form  $\langle key \rangle = \langle value \rangle$  set the corresponding keys of the `graphicx` package to the corresponding values.

```

13012 \ExplSyntaxOn
13013 \@@_if_option:nT
13014 { linkAttributes }
13015 {
13016     \RequirePackage{graphicx}
13017     \markdownSetup{
13018         rendererPrototypes = {
13019             imageAttributeContextBegin = {
13020                 \group_begin:
13021                 \markdownSetup{
13022                     rendererPrototypes = {
13023                         attributeKeyValue = {
13024                             \setkeys
13025                             { Gin }
13026                             { { ##1 } = { ##2 } }
13027                         },
13028                     },
13029                 }
13030             },
13031             imageAttributeContextEnd = {
13032                 \group_end:
13033             },
13034         },
13035     }
13036 }
13037 \ExplSyntaxOff

```

### 3.3.5.12 Raw Attributes

In the raw block and inline raw span renderer prototypes, default to the plain TeX renderer prototypes, translating raw attribute `latex` to `tex`.

```

13038 \ExplSyntaxOn

```

```

13039 \cs_gset:Npn
13040   \markdownRendererInputRawInlinePrototype#1#2
13041   {
13042     \str_case:nnF
13043       { #2 }
13044       {
13045         { latex }
13046         {
13047           \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
13048             { #1 }
13049             { tex }
13050         }
13051       }
13052     {
13053       \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
13054         { #1 }
13055         { #2 }
13056     }
13057   }
13058 \cs_gset:Npn
13059   \markdownRendererInputRawBlockPrototype#1#2
13060   {
13061     \str_case:nnF
13062       { #2 }
13063       {
13064         { latex }
13065         {
13066           \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
13067             { #1 }
13068             { tex }
13069         }
13070       }
13071     {
13072       \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
13073         { #1 }
13074         { #2 }
13075     }
13076   }
13077 \ExplSyntaxOff
13078 \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.

```
13079 \newcommand\markdownMakeOther{%
13080   \count0=128\relax
13081   \loop
13082     \catcode\count0=11\relax
13083     \advance\count0 by 1\relax
13084   \ifnum\count0<256\repeat}%
```

### 3.4 ConTeXt Implementation

The ConTeXt implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV ConTeXt formats *seem* to implement (the documentation is scarce) the majority of the plain TeX format required by the plain TeX implementation. As a consequence, we can directly reuse the existing plain TeX implementation after supplying the missing plain TeX macros.

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `\enableregime` macro. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the `filecontents` L<sup>A</sup>T<sub>E</sub>X package.

```
13085 \def\markdownMakeOther{%
13086   \count0=128\relax
13087   \loop
13088     \catcode\count0=11\relax
13089     \advance\count0 by 1\relax
13090   \ifnum\count0<256\repeat
```

On top of that, make the pipe character (`|`) inactive during the scanning. This is necessary, since the character is active in ConTeXt.

```
13091   \catcode`|=12}%
```

#### 3.4.1 Typesetting Markdown

The `\inputmarkdown` macro is defined to accept an optional argument with options recognized by the ConTeXt interface (see Section 2.4.2).

```
13092 \long\def\inputmarkdown{%
13093   \dosingleempty
13094   \doinputmarkdown}%
13095 \long\def\doinputmarkdown[#1]#2{%
13096   \begingroup
13097     \iffirstargument
13098       \setupmarkdown[#1]%
13099     \fi
13100     \markdownInput{#2}%
13101   \endgroup}%
```



The `\startmarkdown` and `\stopmarkdown` 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. [12, sec. 31]. According to Eijkhout [13, 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.

```

13102 \startluacode
13103   document.markdown_buffering = false
13104   local function preserve_trailing_spaces(line)
13105     if document.markdown_buffering then
13106       line = line:gsub("[ \t][ \t]$", "\t\t")
13107     end
13108     return line
13109   end
13110   resolvers.installinputlinehandler(preserve_trailing_spaces)
13111 \stopluacode
13112 \begingroup
13113   \catcode\|=0%
13114   \catcode\|=12%
13115   |gdef|startmarkdown{%
13116     |ctxlua{document.markdown_buffering = true}%
13117     |markdownReadAndConvert{\stopmarkdown}%
13118                               {|stopmarkdown}}%
13119   |gdef|stopmarkdown{%
13120     |ctxlua{document.markdown_buffering = false}%
13121     |markdownEnd}%
13122 |endgroup

```

### 3.4.2 Themes

This section overrides the plain T<sub>E</sub>X implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in ConT<sub>E</sub>Xt themes provided with the Markdown package.

```

13123 \ExplSyntaxOn
13124 \cs_gset:Nn
13125   \@@_load_theme:nn
13126   {

```

Determine whether a file named `t-markdowntheme<munged theme name>.tex` exists. If it does, load it. Otherwise, try loading a plain T<sub>E</sub>X theme instead.

```

13127   \file_if_exist:nTF
13128     { t - markdown theme #2.tex }
13129     {
13130       \msg_info:nnn

```

```

13131         { markdown }
13132         { loading-context-theme }
13133         { #1 }
13134     \usemodule
13135     [ t ]
13136     [ markdown theme #2 ]
13137 }
13138 {
13139     \@@_plain_tex_load_theme:nn
13140     { #1 }
13141     { #2 }
13142 }
13143 }
13144 \msg_new:nnn
13145 { markdown }
13146 { loading-context-theme }
13147 { Loading~ConTeXt~Markdown~theme~#1 }
13148 \ExplSyntaxOff

```

The `witiko/markdown/defaults` ConTeXt theme provides default definitions for token renderer prototypes. First, the ConTeXt theme loads the plain TeX theme with the default definitions for plain TeX:

```

13149 \markdownLoadPlainTeXTheme

```

Next, the ConTeXt theme overrides some of the plain TeX definitions. See Section 3.4.3 for the actual definitions.

### 3.4.3 Token Renderer Prototypes

The following configuration should be considered placeholder. If the option `plain` has been enabled (see Section 2.2.2.3), none of the definitions will take effect.

```

13150 \markdownIfOption{plain}{\iffalse}{\iftrue}
13151 \def\markdownRendererHardLineBreakPrototype{\blank}%
13152 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
13153 \def\markdownRendererRightBracePrototype{\textbraceright}%
13154 \def\markdownRendererDollarSignPrototype{\textdollar}%
13155 \def\markdownRendererPercentSignPrototype{\percent}%
13156 \def\markdownRendererUnderscorePrototype{\textunderscore}%
13157 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
13158 \def\markdownRendererBackslashPrototype{\textbackslash}%
13159 \def\markdownRendererTildePrototype{\textasciitilde}%
13160 \def\markdownRendererPipePrototype{\char`|}%
13161 \def\markdownRendererLinkPrototype#1#2#3#4{%
13162     \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
13163     \fi\texttt<\hyphenatedurl{#3}>}}%
13164 \usemodule[database]
13165 \defineseparatedlist

```

```

13166 [MarkdownConTeXtCSV]
13167 [separator={,},
13168 before=\bTABLE,after=\eTABLE,
13169 first=\bTR,last=\eTR,
13170 left=\bTD,right=\eTD]
13171 \def\markdownConTeXtCSV{csv}
13172 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
13173 \def\markdownConTeXtCSV@arg{#1}%
13174 \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
13175 \placetable[] [tab:#1]{#4}{%
13176 \processseparatedfile[MarkdownConTeXtCSV][#3]}%
13177 \else
13178 \markdownInput{#3}%
13179 \fi}%
13180 \def\markdownRendererImagePrototype#1#2#3#4{%
13181 \placefigure[] []{#4}{\externalfigure[#3]}}%
13182 \def\markdownRendererUlBeginPrototype{\startitemize}%
13183 \def\markdownRendererUlBeginTightPrototype{\startitemize[packed]}%
13184 \def\markdownRendererUlItemPrototype{\item}%
13185 \def\markdownRendererUlEndPrototype{\stopitemize}%
13186 \def\markdownRendererUlEndTightPrototype{\stopitemize}%
13187 \def\markdownRendererOlBeginPrototype{\startitemize[n]}%
13188 \def\markdownRendererOlBeginTightPrototype{\startitemize[packed,n]}%
13189 \def\markdownRendererOlItemPrototype{\item}%
13190 \def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
13191 \def\markdownRendererOlEndPrototype{\stopitemize}%
13192 \def\markdownRendererOlEndTightPrototype{\stopitemize}%
13193 \definedescription
13194 [MarkdownConTeXtDlItemPrototype]
13195 [location=hanging,
13196 margin=standard,
13197 headstyle=bold]%
13198 \definestartstop
13199 [MarkdownConTeXtDlPrototype]
13200 [before=\blank,
13201 after=\blank]%
13202 \definestartstop
13203 [MarkdownConTeXtDlTightPrototype]
13204 [before=\blank\startpacked,
13205 after=\stoppacked\blank]%
13206 \def\markdownRendererDlBeginPrototype{%
13207 \startMarkdownConTeXtDlPrototype}%
13208 \def\markdownRendererDlBeginTightPrototype{%
13209 \startMarkdownConTeXtDlTightPrototype}%
13210 \def\markdownRendererDlItemPrototype#1{%
13211 \startMarkdownConTeXtDlItemPrototype{#1}}%
13212 \def\markdownRendererDlItemEndPrototype{%

```

```

13213 \stopMarkdownConTeXtDlItemPrototype}%
13214 \def\markdownRendererDlEndPrototype{%
13215 \stopMarkdownConTeXtDlPrototype}%
13216 \def\markdownRendererDlEndTightPrototype{%
13217 \stopMarkdownConTeXtDlTightPrototype}%
13218 \def\markdownRendererEmphasisPrototype#1{\em#1}%
13219 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
13220 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
13221 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
13222 \def\markdownRendererLineBlockBeginPrototype{%
13223 \begingroup
13224 \def\markdownRendererHardLineBreak{
13225 }%
13226 \startlines
13227 }%
13228 \def\markdownRendererLineBlockEndPrototype{%
13229 \stoplines
13230 \endgroup
13231 }%
13232 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%

```

### 3.4.3.1 Fenced Code

When no `infostring` has been specified, default to the indented code block renderer.

```

13233 \ExplSyntaxOn
13234 \cs_gset:Npn
13235 \markdownRendererInputFencedCodePrototype#1#2#3
13236 {
13237 \tl_if_empty:nTF
13238 { #2 }
13239 { \markdownRendererInputVerbatim{#1} }

```

Otherwise, extract the first word of the `infostring` and treat it as the name of the programming language in which the code block is written. This name is then used in the ConTeXt `\definetying` macro, which allows the user to set up code highlighting mapping as follows:

```

\definetying [latex]
\setuptying [latex] [option=TEX]

\starttext
\startmarkdown
~~~ latex
\documentclass{article}
\begin{document}
Hello world!

```

```

\end{document}
~~~
\stopmarkdown
\stoptext

```

```

13240     {
13241         \regex_extract_once:nnN
13242         { \w* }
13243         { #2 }
13244         \l_tmpa_seq
13245         \seq_pop_left:NN
13246         \l_tmpa_seq
13247         \l_tmpa_tl
13248         \typefile[\l_tmpa_tl] []{#1}
13249     }
13250 }
13251 \ExplSyntaxOff
13252 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
13253 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
13254 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
13255 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
13256 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
13257 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
13258 \def\markdownRendererThematicBreakPrototype{%
13259     \blackrule[height=1pt, width=\hsize]}%
13260 \def\markdownRendererNotePrototype#1{\footnote{#1}}%
13261 \def\markdownRendererTickedBoxPrototype{$\boxtimes$}
13262 \def\markdownRendererHalfTickedBoxPrototype{$\boxdot$}
13263 \def\markdownRendererUntickedBoxPrototype{$\square$}
13264 \def\markdownRendererStrikeThroughPrototype#1{\overstrides{#1}}
13265 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
13266 \def\markdownRendererSubscriptPrototype#1{\low{#1}}
13267 \def\markdownRendererDisplayMathPrototype#1{\startformula#1\stopformula}%

```

### 3.4.3.2 Tables

There is a basic implementation of tables.

```

13268 \newcount\markdownConTeXtRowCounter
13269 \newcount\markdownConTeXtRowTotal
13270 \newcount\markdownConTeXtColumnCounter
13271 \newcount\markdownConTeXtColumnTotal
13272 \newtoks\markdownConTeXtTable
13273 \newtoks\markdownConTeXtTableFloat
13274 \def\markdownRendererTablePrototype#1#2#3{%
13275     \markdownConTeXtTable={}%
13276     \ifx\empty#1\empty
13277         \markdownConTeXtTableFloat={%

```

```

13278     \the\markdownConTeXtTable}%
13279 \else
13280     \markdownConTeXtTableFloat={%
13281     \placetable{#1}{\the\markdownConTeXtTable}}%
13282 \fi
13283 \begingroup
13284 \setupTABLE[r][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
13285 \setupTABLE[c][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
13286 \setupTABLE[r][1][topframe=on, bottomframe=on]
13287 \setupTABLE[r][#1][bottomframe=on]
13288 \markdownConTeXtRowCounter=0%
13289 \markdownConTeXtRowTotal=#2%
13290 \markdownConTeXtColumnTotal=#3%
13291 \markdownConTeXtRenderTableRow}
13292 \def\markdownConTeXtRenderTableRow#1{%
13293     \markdownConTeXtColumnCounter=0%
13294     \ifnum\markdownConTeXtRowCounter=0\relax
13295         \markdownConTeXtReadAlignments#1%
13296         \markdownConTeXtTable={\bTABLE}%
13297     \else
13298         \markdownConTeXtTable=\expandafter{%
13299             \the\markdownConTeXtTable\bTR}%
13300         \markdownConTeXtRenderTableCell#1%
13301         \markdownConTeXtTable=\expandafter{%
13302             \the\markdownConTeXtTable\eTR}%
13303     \fi
13304     \advance\markdownConTeXtRowCounter by 1\relax
13305     \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
13306         \markdownConTeXtTable=\expandafter{%
13307             \the\markdownConTeXtTable\eTABLE}%
13308         \the\markdownConTeXtTableFloat
13309     \endgroup
13310     \expandafter\gobbleoneargument
13311     \fi\markdownConTeXtRenderTableRow}
13312 \def\markdownConTeXtReadAlignments#1{%
13313     \advance\markdownConTeXtColumnCounter by 1\relax
13314     \if#1d%
13315         \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
13316     \fi\if#1l%
13317         \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
13318     \fi\if#1c%
13319         \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
13320     \fi\if#1r%
13321         \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
13322     \fi
13323     \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
13324         \expandafter\gobbleoneargument

```

```

13325 \fi\markdownConTeXtReadAlignments}
13326 \def\markdownConTeXtRenderTableCell#1{%
13327 \advance\markdownConTeXtColumnCounter by 1\relax
13328 \markdownConTeXtTable=\expandafter{%
13329 \the\markdownConTeXtTable\bTD#1\eTD}%
13330 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
13331 \expandafter\gobbleoneargument
13332 \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`.

```

13333 \ExplSyntaxOn
13334 \cs_gset:Npn
13335 \markdownRendererInputRawInlinePrototype#1#2
13336 {
13337 \str_case:nnF
13338 { #2 }
13339 {
13340 { latex }
13341 {
13342 \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
13343 { #1 }
13344 { context }
13345 }
13346 }
13347 {
13348 \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
13349 { #1 }
13350 { #2 }
13351 }
13352 }
13353 \cs_gset:Npn
13354 \markdownRendererInputRawBlockPrototype#1#2
13355 {
13356 \str_case:nnF
13357 { #2 }
13358 {
13359 { context }
13360 {
13361 \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
13362 { #1 }
13363 { tex }
13364 }
13365 }
13366 {

```

```

13367      \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
13368      { #1 }
13369      { #2 }
13370    }
13371  }
13372  \cs_gset_eq:NN
13373    \markdownRendererInputRawBlockPrototype
13374    \markdownRendererInputRawInlinePrototype
13375  \fi % Closes ``\markdownIfOption{plain}{\iffalse}{\iftrue}`
13376  \ExplSyntaxOff
13377  \stopmodule
13378  \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).

```

13379  \markdownIfOption{noDefaults}{}{
13380    \setupmarkdown[theme=witiko/markdown/defaults]
13381  }
13382  \stopmodule
13383  \protect

```

## References

- [1] LuaT<sub>E</sub>X development team. *LuaT<sub>E</sub>X reference manual*. Version 1.10 (stable). July 23, 2021. URL: <https://www.pragma-ade.com/general/manuals/luatex.pdf> (visited on 09/30/2022).
- [2] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: <https://www.muni.cz/en/research/projects/32984> (visited on 02/19/2018).
- [3] Anton Sotkov. *File transclusion syntax for Markdown*. Jan. 19, 2017. URL: <https://github.com/iainc/Markdown-Content-Blocks> (visited on 01/08/2018).
- [4] John MacFarlane. *Pandoc. a universal document converter*. 2022. URL: <https://pandoc.org/> (visited on 10/05/2022).
- [5] Bonita Sharif and Jonathan I. Maletic. “An Eye Tracking Study on camelCase and under\_score Identifier Styles.” In: *2010 IEEE 18th International Conference on Program Comprehension*. 2010, pp. 196–205. DOI: [10.1109/ICPC.2010.41](https://doi.org/10.1109/ICPC.2010.41).
- [6] Donald Ervin Knuth. *The T<sub>E</sub>Xbook*. 3rd ed. Vol. A. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. ix, 479. ISBN: 0-201-13447-0.
- [7] Frank Mittelbach. *The doc and shortverb Packages*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/doc.pdf> (visited on 02/19/2018).



- [8] Till Tantau, Joseph Wright, and Vedran Miletic. *The Beamer class*. Feb. 10, 2021. URL: <https://mirrors.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf> (visited on 02/11/2021).
- [9] 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).
- [10] Roberto Ierusalimsky. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.
- [11] 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).
- [12] 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.
- [13] Victor Eijkhout. *T<sub>E</sub>X by Topic. A T<sub>E</sub>Xnician's Reference*. Wokingham, England: Addison-Wesley, Feb. 1, 1992. 307 pp. ISBN: 978-0-201-56882-0.

## Index

|                          |                                   |
|--------------------------|-----------------------------------|
| autoIdentifiers          | 18, 30, 71, 85                    |
| blankBeforeBlockquote    | 18                                |
| blankBeforeCodeFence     | 19                                |
| blankBeforeDivFence      | 19                                |
| blankBeforeHeading       | 19                                |
| blankBeforeList          | 20                                |
| bracketedSpans           | 20, 73                            |
| breakableBlockquotes     | 20                                |
| cacheDir                 | 4, 15, 17, 53, 127, 275, 341, 353 |
| citationNbsps            | 21                                |
| citations                | 21, 76                            |
| codeSpans                | 22                                |
| contentBlocks            | 17, 22                            |
| contentBlocksLanguageMap | 17                                |
| contentLevel             | 23                                |
| debugExtensions          | 9, 17, 23, 272                    |
| debugExtensionsFileName  | 17, 23                            |
| defaultOptions           | 9, 47, 321                        |
| definitionLists          | 23, 80                            |

|                                   |                                     |
|-----------------------------------|-------------------------------------|
| eagerCache                        | 15                                  |
| entities.char_entity              | 185                                 |
| entities.dec_entity               | 185                                 |
| entities.hex_entity               | 185                                 |
| entities.hex_entity_with_x_char   | 185                                 |
| expandtabs                        | 237                                 |
| expectJekyllData                  | 24                                  |
| extensions                        | 25, 134, 277                        |
| extensions.bracketed_spans        | 277                                 |
| extensions.citations              | 278                                 |
| extensions.content_blocks         | 282                                 |
| extensions.definition_lists       | 285                                 |
| extensions.fancy_lists            | 287                                 |
| extensions.fenced_code            | 292                                 |
| extensions.fenced_divs            | 297                                 |
| extensions.header_attributes      | 300                                 |
| extensions.inline_code_attributes | 301                                 |
| extensions.jekyll_data            | 318                                 |
| extensions.line_blocks            | 302                                 |
| extensions.link_attributes        | 303                                 |
| extensions.mark                   | 303                                 |
| extensions.notes                  | 305                                 |
| extensions.pipe_table             | 307                                 |
| extensions.raw_inline             | 311                                 |
| extensions.strike_through         | 312                                 |
| extensions.subscripts             | 313                                 |
| extensions.superscripts           | 313                                 |
| extensions.tex_math               | 314                                 |
| fancyLists                        | 27, 94–99, 354                      |
| fencedCode                        | 27, 35, 77, 83, 99, 351             |
| fencedCodeAttributes              | 28, 71                              |
| fencedDiv                         | 84                                  |
| fencedDivs                        | 28, 37                              |
| finalizeCache                     | 16, 18, 29, 29, 53, 126, 276        |
| frozenCache                       | 18, 29, 53, 126, 129, 130, 352, 353 |
| frozenCacheCounter                | 29, 276, 345, 346                   |
| frozenCacheFileName               | 18, 29, 53, 276                     |
| gfmAutoIdentifiers                | 18, 29, 71, 85                      |
| hashEnumerators                   | 30                                  |
| headerAttributes                  | 30, 37, 71, 85                      |

|                                              |                                                 |
|----------------------------------------------|-------------------------------------------------|
| html                                         | 31, 87, 88, 363                                 |
| hybrid                                       | 31, 36, 42, 44, 56, 64, 101, 127, 190, 238, 345 |
| inlineCodeAttributes                         | 31, 71, 78                                      |
| inlineNotes                                  | 32                                              |
| \input                                       | 50                                              |
| \inputmarkdown                               | 131, 132, 376                                   |
| inputTempFileName                            | 54, 56, 342, 343                                |
| iterlines                                    | 237                                             |
| jekyllData                                   | 3, 24, 25, 32, 107–111                          |
| languages_json                               | 282, 282                                        |
| lineBlocks                                   | 33, 90                                          |
| linkAttributes                               | 33, 71, 89, 92, 255, 374                        |
| mark                                         | 34, 92, 373                                     |
| \markdown                                    | 125                                             |
| markdown                                     | 124, 125, 347                                   |
| markdown*                                    | 124, 125, 126, 347                              |
| \markdown_jekyll_data_concatenate_address:NN | 336                                             |
| \markdown_jekyll_data_pop:                   | 337                                             |
| \markdown_jekyll_data_push:nN                | 337                                             |
| \markdown_jekyll_data_push_address_segment:n | 335                                             |
| \markdown_jekyll_data_set_keyval:NN          | 337                                             |
| \markdown_jekyll_data_set_keyvals:nn         | 337                                             |
| \markdown_jekyll_data_update_address_tls:    | 336                                             |
| \markdownBegin                               | 49, 49, 50, 123, 125, 132                       |
| \markdownCleanup                             | 341                                             |
| \markdownEnd                                 | 49, 49, 50, 123, 125, 132                       |
| \markdownError                               | 123, 123                                        |
| \markdownEscape                              | 49, 51, 346                                     |
| \markdownIfOption                            | 52                                              |
| \markdownIfSnippetExists                     | 66                                              |
| \markdownInfo                                | 123, 123                                        |
| \markdownInput                               | 49, 50, 124–126, 132, 345, 347                  |
| \markdownInputFileStream                     | 341                                             |
| \markdownInputPlainTeX                       | 347                                             |
| \markdownLoadPlainTeXTheme                   | 127, 134, 330                                   |
| \markdownLuaExecute                          | 344, 345                                        |
| \markdownLuaOptions                          | 339, 341                                        |
| \markdownMakeOther                           | 123, 376                                        |
| \markdownOptionFinalizeCache                 | 53                                              |

|                                                     |                         |
|-----------------------------------------------------|-------------------------|
| \markdownOptionFrozenCache                          | 53                      |
| \markdownOptionHybrid                               | 56                      |
| \markdownOptionInputTempFileName                    | 53                      |
| \markdownOptionNoDefaults                           | 55                      |
| \markdownOptionOutputDir                            | 53, 54                  |
| \markdownOptionPlain                                | 55                      |
| \markdownOptionStripPercentSigns                    | 56                      |
| \markdownOutputFileStream                           | 341                     |
| \markdownPrepare                                    | 341                     |
| \markdownPrepareLuaOptions                          | 339                     |
| \markdownReadAndConvert                             | 123, 341, 347, 348, 377 |
| \markdownReadAndConvertProcessLine                  | 343, 343                |
| \markdownReadAndConvertStripPercentSigns            | 342                     |
| \markdownReadAndConvertTab                          | 341                     |
| \markdownRendererAttributeClassName                 | 72                      |
| \markdownRendererAttributeIdentifier                | 71                      |
| \markdownRendererAttributeKeyValue                  | 72                      |
| \markdownRendererBlockQuoteBegin                    | 73                      |
| \markdownRendererBlockQuoteEnd                      | 73                      |
| \markdownRendererBracketedSpanAttributeContextBegin | 73                      |
| \markdownRendererBracketedSpanAttributeContextEnd   | 73                      |
| \markdownRendererCite                               | 76, 76                  |
| \markdownRendererCodeSpan                           | 77                      |
| \markdownRendererCodeSpanAttributeContextBegin      | 78                      |
| \markdownRendererCodeSpanAttributeContextEnd        | 78                      |
| \markdownRendererContentBlock                       | 78, 79                  |
| \markdownRendererContentBlockCode                   | 79                      |
| \markdownRendererContentBlockOnlineImage            | 79                      |
| \markdownRendererDisplayMath                        | 106                     |
| \markdownRendererDlBegin                            | 80                      |
| \markdownRendererDlBeginTight                       | 80                      |
| \markdownRendererDlDefinitionBegin                  | 81                      |
| \markdownRendererDlDefinitionEnd                    | 81                      |
| \markdownRendererDlEnd                              | 82                      |
| \markdownRendererDlEndTight                         | 82                      |
| \markdownRendererDlItem                             | 80                      |
| \markdownRendererDlItemEnd                          | 81                      |
| \markdownRendererDocumentBegin                      | 93                      |
| \markdownRendererDocumentEnd                        | 93                      |
| \markdownRendererEllipsis                           | 38, 82                  |
| \markdownRendererEmphasis                           | 83, 113                 |
| \markdownRendererFancyOlBegin                       | 95, 95                  |

|                                                               |     |
|---------------------------------------------------------------|-----|
| <code>\markdownRendererFancyOlBeginTight</code>               | 95  |
| <code>\markdownRendererFancyOlEnd</code>                      | 98  |
| <code>\markdownRendererFancyOlEndTight</code>                 | 98  |
| <code>\markdownRendererFancyOlItem</code>                     | 96  |
| <code>\markdownRendererFancyOlItemEnd</code>                  | 97  |
| <code>\markdownRendererFancyOlItemWithNumber</code>           | 97  |
| <code>\markdownRendererFencedCodeAttributeContextBegin</code> | 83  |
| <code>\markdownRendererFencedCodeAttributeContextEnd</code>   | 83  |
| <code>\markdownRendererFencedDivAttributeContextBegin</code>  | 84  |
| <code>\markdownRendererFencedDivAttributeContextEnd</code>    | 84  |
| <code>\markdownRendererHalfTickedBox</code>                   | 107 |
| <code>\markdownRendererHardLineBreak</code>                   | 91  |
| <code>\markdownRendererHeaderAttributeContextBegin</code>     | 85  |
| <code>\markdownRendererHeaderAttributeContextEnd</code>       | 85  |
| <code>\markdownRendererHeadingFive</code>                     | 86  |
| <code>\markdownRendererHeadingFour</code>                     | 86  |
| <code>\markdownRendererHeadingOne</code>                      | 85  |
| <code>\markdownRendererHeadingSix</code>                      | 87  |
| <code>\markdownRendererHeadingThree</code>                    | 86  |
| <code>\markdownRendererHeadingTwo</code>                      | 86  |
| <code>\markdownRendererImage</code>                           | 88  |
| <code>\markdownRendererImageAttributeContextBegin</code>      | 89  |
| <code>\markdownRendererImageAttributeContextEnd</code>        | 89  |
| <code>\markdownRendererInlineHtmlComment</code>               | 87  |
| <code>\markdownRendererInlineHtmlTag</code>                   | 87  |
| <code>\markdownRendererInlineMath</code>                      | 106 |
| <code>\markdownRendererInputBlockHtmlElement</code>           | 88  |
| <code>\markdownRendererInputFencedCode</code>                 | 77  |
| <code>\markdownRendererInputRawBlock</code>                   | 99  |
| <code>\markdownRendererInputRawInline</code>                  | 99  |
| <code>\markdownRendererInputVerbatim</code>                   | 77  |
| <code>\markdownRendererInterblockSeparator</code>             | 89  |
| <code>\markdownRendererJekyllDataBegin</code>                 | 107 |
| <code>\markdownRendererJekyllDataBoolean</code>               | 109 |
| <code>\markdownRendererJekyllDataEmpty</code>                 | 111 |
| <code>\markdownRendererJekyllDataEnd</code>                   | 108 |
| <code>\markdownRendererJekyllDataMappingBegin</code>          | 108 |
| <code>\markdownRendererJekyllDataMappingEnd</code>            | 108 |
| <code>\markdownRendererJekyllDataNumber</code>                | 110 |
| <code>\markdownRendererJekyllDataSequenceBegin</code>         | 109 |
| <code>\markdownRendererJekyllDataSequenceEnd</code>           | 109 |
| <code>\markdownRendererJekyllDataString</code>                | 110 |

|                                             |                                |
|---------------------------------------------|--------------------------------|
| \markdownRendererLineBlockBegin             | 90                             |
| \markdownRendererLineBlockEnd               | 90                             |
| \markdownRendererLink                       | 91, 113                        |
| \markdownRendererLinkAttributeContextBegin  | 92                             |
| \markdownRendererLinkAttributeContextEnd    | 92                             |
| \markdownRendererMark                       | 92                             |
| \markdownRendererNbsp                       | 93                             |
| \markdownRendererNote                       | 94                             |
| \markdownRendererOlBegin                    | 94                             |
| \markdownRendererOlBeginTight               | 94                             |
| \markdownRendererOlEnd                      | 97                             |
| \markdownRendererOlEndTight                 | 98                             |
| \markdownRendererOlItem                     | 38, 95                         |
| \markdownRendererOlItemEnd                  | 96                             |
| \markdownRendererOlItemWithNumber           | 38, 96                         |
| \markdownRendererParagraphSeparator         | 90                             |
| \markdownRendererReplacementCharacter       | 100                            |
| \markdownRendererSectionBegin               | 100                            |
| \markdownRendererSectionEnd                 | 100                            |
| \markdownRendererSoftLineBreak              | 91                             |
| \markdownRendererStrikeThrough              | 103                            |
| \markdownRendererStrongEmphasis             | 83                             |
| \markdownRendererSubscript                  | 104                            |
| \markdownRendererSuperscript                | 104                            |
| \markdownRendererTable                      | 105                            |
| \markdownRendererTableAttributeContextBegin | 104                            |
| \markdownRendererTableAttributeContextEnd   | 104                            |
| \markdownRendererTextCite                   | 76                             |
| \markdownRendererThematicBreak              | 106                            |
| \markdownRendererTickedBox                  | 107                            |
| \markdownRendererUlBegin                    | 74                             |
| \markdownRendererUlBeginTight               | 74                             |
| \markdownRendererUlEnd                      | 75                             |
| \markdownRendererUlEndTight                 | 75                             |
| \markdownRendererUlItem                     | 74                             |
| \markdownRendererUlItemEnd                  | 75                             |
| \markdownRendererUntickedBox                | 107                            |
| \markdownSetup                              | 52, 52, 56, 126, 132, 348, 349 |
| \markdownSetupSnippet                       | 65, 65                         |
| \markdownWarning                            | 123, 123                       |
| new                                         | 7, 16, 321                     |

|                                |                           |
|--------------------------------|---------------------------|
| notes                          | 34, 94                    |
| parsers                        | 204, 236, 237             |
| pipeTables                     | 6, 35, 41, 105            |
| preserveTabs                   | 35, 39, 237               |
| rawAttribute                   | 35, 36, 99                |
| reader                         | 8, 26, 134, 204, 236, 277 |
| reader->add_special_character  | 8, 9, 26, 271             |
| reader->auto_link_email        | 261                       |
| reader->auto_link_url          | 261                       |
| reader->create_parser          | 237                       |
| reader->finalize_grammar       | 267, 325                  |
| reader->initialize_named_group | 271                       |
| reader->insert_pattern         | 8, 9, 26, 267, 268, 273   |
| reader->lookup_reference       | 249                       |
| reader->normalize_tag          | 237                       |
| reader->options                | 236                       |
| reader->parser_functions       | 237                       |
| reader->parser_functions.name  | 237                       |
| reader->parsers                | 236, 236, 237             |
| reader->register_link          | 249                       |
| reader->update_rule            | 267, 270, 273             |
| reader->writer                 | 236                       |
| reader.new                     | 236, 236, 325             |
| relativeReferences             | 36                        |
| \setupmarkdown                 | 132                       |
| shiftHeadings                  | 6, 37                     |
| singletonCache                 | 16                        |
| slice                          | 6, 37, 187, 198, 199      |
| smartEllipses                  | 38, 82, 127               |
| \startmarkdown                 | 131, 132, 377             |
| startNumber                    | 38, 95–97                 |
| \stopmarkdown                  | 131, 132, 377             |
| strikeThrough                  | 38, 103, 373              |
| stripIndent                    | 39, 238                   |
| stripPercentSigns              | 342                       |
| subscripts                     | 39, 104                   |
| superscripts                   | 40, 104                   |
| syntax                         | 268, 273                  |
| tableAttributes                | 40, 104                   |

|                               |                                         |
|-------------------------------|-----------------------------------------|
| tableCaptions                 | 6, 40, 41, 104                          |
| taskLists                     | 41, 107, 363                            |
| texComments                   | 42, 238                                 |
| texMathDollars                | 42, 106                                 |
| texMathDoubleBackslash        | 43, 106                                 |
| texMathSingleBackslash        | 43, 106                                 |
| tightLists                    | 43, 74, 75, 80, 82, 94, 95, 98, 99, 354 |
| underscores                   | 44                                      |
| util.cache                    | 135, 135                                |
| util.cache_verbatim           | 135                                     |
| util.encode_json_string       | 136                                     |
| util.err                      | 135                                     |
| util.escaper                  | 138                                     |
| util.expand_tabs_in_line      | 136                                     |
| util.flatten                  | 137                                     |
| util.intersperse              | 138                                     |
| util.lookup_files             | 136                                     |
| util.map                      | 138                                     |
| util.pathname                 | 139                                     |
| util.rope_last                | 137                                     |
| util.rope_to_string           | 137                                     |
| util.table_copy               | 136                                     |
| util.walk                     | 136, 137                                |
| walkable_syntax               | 8, 17, 23, 267, 268, 270–273            |
| writer                        | 134, 134, 186, 277                      |
| writer->active_attributes     | 197, 197, 198                           |
| writer->attribute_type_levels | 197                                     |
| writer->attributes            | 195                                     |
| writer->block_html_element    | 193                                     |
| writer->blockquote            | 194                                     |
| writer->bulletitem            | 192                                     |
| writer->bulletlist            | 192                                     |
| writer->citations             | 278                                     |
| writer->code                  | 190                                     |
| writer->contentblock          | 282                                     |
| writer->defer_call            | 204, 204                                |
| writer->definitionlist        | 285                                     |
| writer->display_math          | 314                                     |
| writer->div_begin             | 297                                     |
| writer->div_end               | 297                                     |



|                                  |               |
|----------------------------------|---------------|
| writer->document                 | 194           |
| writer->ellipsis                 | 188           |
| writer->emphasis                 | 194           |
| writer->escape                   | 190           |
| writer->escape_minimal           | 189           |
| writer->escape_programmatic_text | 189           |
| writer->escape_typographic_text  | 189           |
| writer->escaped_chars            | 189, 189      |
| writer->escaped_minimal_strings  | 189, 189      |
| writer->escaped_strings          | 189           |
| writer->escaped_uri_chars        | 189, 189      |
| writer->fancyitem                | 288           |
| writer->fancylist                | 287           |
| writer->fencedCode               | 293           |
| writer->flatten_inlines          | 186, 186      |
| writer->get_state                | 203           |
| writer->hard_line_break          | 188           |
| writer->heading                  | 202           |
| writer->identifier               | 190           |
| writer->image                    | 191           |
| writer->infostring               | 190           |
| writer->inline_html_comment      | 193           |
| writer->inline_html_tag          | 193           |
| writer->inline_math              | 314           |
| writer->interblocksep            | 188           |
| writer->is_writing               | 187, 187      |
| writer->jekyllData               | 318           |
| writer->lineblock                | 302           |
| writer->link                     | 191           |
| writer->mark                     | 303           |
| writer->math                     | 190           |
| writer->nbsp                     | 187           |
| writer->note                     | 305           |
| writer->options                  | 186           |
| writer->ordereditem              | 193           |
| writer->orderedlist              | 192           |
| writer->pack                     | 188, 276      |
| writer->paragraph                | 188           |
| writer->paragraphsep             | 188           |
| writer->plain                    | 187           |
| writer->pop_attributes           | 197, 198      |
| writer->push_attributes          | 197, 198, 199 |

|                                         |                                                                 |
|-----------------------------------------|-----------------------------------------------------------------|
| <code>writer-&gt;rawBlock</code>        | <a href="#">293</a>                                             |
| <code>writer-&gt;rawInline</code>       | <a href="#">311</a>                                             |
| <code>writer-&gt;set_state</code>       | <a href="#">203</a>                                             |
| <code>writer-&gt;slice_begin</code>     | <a href="#">187</a>                                             |
| <code>writer-&gt;slice_end</code>       | <a href="#">187</a>                                             |
| <code>writer-&gt;soft_line_break</code> | <a href="#">188</a>                                             |
| <code>writer-&gt;space</code>           | <a href="#">187</a>                                             |
| <code>writer-&gt;span</code>            | <a href="#">277</a>                                             |
| <code>writer-&gt;strike_through</code>  | <a href="#">312</a>                                             |
| <code>writer-&gt;string</code>          | <a href="#">190</a>                                             |
| <code>writer-&gt;strong</code>          | <a href="#">194</a>                                             |
| <code>writer-&gt;subscript</code>       | <a href="#">313</a>                                             |
| <code>writer-&gt;suffix</code>          | <a href="#">187</a>                                             |
| <code>writer-&gt;superscript</code>     | <a href="#">313</a>                                             |
| <code>writer-&gt;table</code>           | <a href="#">308</a>                                             |
| <code>writer-&gt;thematic_break</code>  | <a href="#">188</a>                                             |
| <code>writer-&gt;checkbox</code>        | <a href="#">194</a>                                             |
| <code>writer-&gt;uri</code>             | <a href="#">190</a>                                             |
| <code>writer-&gt;verbatim</code>        | <a href="#">194</a>                                             |
| <code>writer.new</code>                 | <a href="#">186</a> , <a href="#">186</a> , <a href="#">325</a> |