

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

Vít Starý Novotný  
witiko@mail.muni.cz

Version 3.6.1-0-g5a45a922  
2024-06-20

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>	<b>3</b>	<b>Implementation</b>	<b>139</b>
1.1	Requirements . . . . .	2	3.1	Lua Implementation . . . . .	140
1.2	Feedback . . . . .	6	3.2	Plain T <sub>E</sub> X Implementation	338
1.3	Acknowledgements . . . . .	6	3.3	L <sup>A</sup> T <sub>E</sub> X Implementation . . . . .	360
<b>2</b>	<b>Interfaces</b>	<b>6</b>	3.4	ConT <sub>E</sub> Xt Implementation	391
2.1	Lua Interface . . . . .	7			
2.2	Plain T <sub>E</sub> X Interface . . . . .	50			
2.3	L <sup>A</sup> T <sub>E</sub> X Interface . . . . .	129			
2.4	ConT <sub>E</sub> Xt Interface . . . . .	136			
				<b>References</b>	<b>399</b>
				<b>Index</b>	<b>400</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 . . . . .	46
3	A sequence diagram of typesetting a document using the Lua CLI . . . . .	47
4	Various formats of mathematical formulae . . . . .	134
5	The banner of the Markdown package . . . . .	135

## 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 Lua<sub>TEX</sub> engine (though not necessarily in the LuaMeta<sub>TEX</sub> engine).

**LPeg  $\geq$  0.10** A pattern-matching library for the writing of recursive descent parsers via the Parsing Expression Grammars (PEGs). It is used by the Lunamark library to parse the markdown input. LPeg  $\geq$  0.10 is included in Lua<sub>TEX</sub>  $\geq$  0.72.0 (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 Lua<sub>TEX</sub> (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 Lua<sub>TEX</sub> (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 <sub>TEX</sub> 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 Lua $\TeX$  has not yet been initialized, configure Kpathsea on top of loading it. Since Con $\TeX$ t 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 Lua $\TeX$  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 $\text{\AA}$  $\TeX$ 3 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 3.3.4) or L<sup>A</sup>TeX themes (see Section 2.3.3) and will not be loaded if the option `plain` has been enabled (see Section 2.2.2.3):

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

**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  $2006 \leq$  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 3.3.4.8, 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.

<sup>35</sup> `\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-~~L~~A<sub>T</sub>E<sub>X</sub> 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 L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>, the minted package by Geoffrey M. Poore, which likewise tackles the issue of interfacing with an external interpreter from T<sub>E</sub>X, the filecontents package by Scott Pakin and others.

## 2 Interfaces

This part of the documentation describes the interfaces exposed by the package along with usage notes and examples. It is aimed at the user of the package.

Since neither T<sub>E</sub>X nor Lua provide interfaces as a language construct, the separation to interfaces and implementations is a *gentlemen's agreement*. It serves as a means of

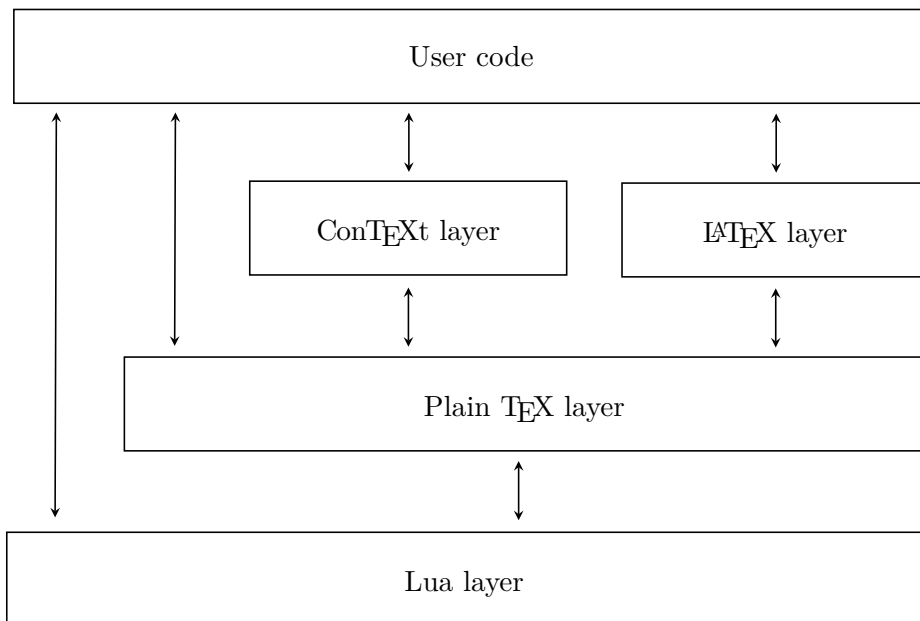
---

<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  $\text{T}_{\text{E}}\text{X}$  *token renderers* is exposed by the Lua layer. The plain  $\text{T}_{\text{E}}\text{X}$  layer exposes the conversion capabilities of Lua as  $\text{T}_{\text{E}}\text{X}$  macros. The  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  and  $\text{ConT}_{\text{E}}\text{Xt}$  layers provide syntactic sugar on top of plain  $\text{T}_{\text{E}}\text{X}$  macros. The user can interface with any and all layers.



**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  $\text{T}_{\text{E}}\text{X}$ . This interface is used by the plain  $\text{T}_{\text{E}}\text{X}$  implementation (see Section 3.2) and will be of interest to the developers of other packages and Lua modules.

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

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

### 2.1.1 Conversion from Markdown to Plain $\text{T}_{\text{E}}\text{X}$

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

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

The following example Lua code converts the markdown string `Hello *world*!` to a  $\text{\TeX}$  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:nmmn
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:nmmn
105   { Vmmn }
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:n` 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:mnN
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:mnN
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>.



`unicodeNormalization=true, false`

default: true

- true**      Markdown documents will be normalized using one of the four Unicode normalization forms<sup>7</sup> before conversion. The Unicode normalization norm used is determined by option `unicodeNormalizationForm`.
- false**     Markdown documents will not be Unicode-normalized before conversion.

```
359 \@@_add_lua_option:nnn
360 { unicodeNormalization }
361 { boolean }
362 { true }

363 defaultOptions.unicodeNormalization = true
```

`unicodeNormalizationForm=nfc, nfd, nfkc, nfkd`

default: nfc

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

```
364 \@@_add_lua_option:nnn
365 { unicodeNormalizationForm }
366 { string }
367 { nfc }

368 defaultOptions.unicodeNormalizationForm = "nfc"
```

## 2.1.5 File and Directory Names

---

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

`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.

```
369 \@@_add_lua_option:nnn
370   { cacheDir }
371   { path }
372   { \markdownOptionOutputDir / _markdown_\jobname }
373 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.

```
374 \@@_add_lua_option:nnn
375   { contentBlocksLanguageMap }
376   { path }
377   { markdown-languages.json }
378 defaultOptions.contentBlocksLanguageMap = "markdown-languages.json"
```

`debugExtensionsFileName`=*<filename>* default: `debug-extensions.json`

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

```
379 \@@_add_lua_option:nnn
380   { debugExtensionsFileName }
381   { path }
382   { \markdownOptionOutputDir / \jobname .debug-extensions.json }
383 defaultOptions.debugExtensionsFileName = "debug-extensions.json"
```

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

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

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

```
384 \@@_add_lua_option:nnn
385   { frozenCacheFileName }
386   { path }
387   { \markdownOptionCacheDir / frozenCache.tex }
388 defaultOptions.frozenCacheFileName = "frozenCache.tex"
```

### 2.1.6 Parser Options

`autoIdentifiers`=true, false default: false

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

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

```
# 123 Sesame Street
```

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

See also the option `gfmAutoIdentifiers`.

```
389 \@@_add_lua_option:nnn
390   { autoIdentifiers }
391   { boolean }
392   { false }
393 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>8</sup>See [https://pandoc.org/MANUAL.html#extension-auto\\_identifiers](https://pandoc.org/MANUAL.html#extension-auto_identifiers).

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

398 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.

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

403 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.

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

408 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.

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

413 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.

```
414 \@@_add_lua_option:nnn
415   { blankBeforeList }
416   { boolean }
417   { false }

418 defaultOptions.blankBeforeList = false
```

`bracketedSpans=true, false` default: false

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

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

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

```
419 \@@_add_lua_option:nnn
420   { bracketedSpans }
421   { boolean }
422   { false }

423 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.

```
424 \@@_add_lua_option:nnn
425   { breakableBlockquotes }
426   { boolean }
427   { true }

428 defaultOptions.breakableBlockquotes = true
```

---

<sup>9</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.

```
429 \@@_add_lua_option:nnn
430 { citationNbsps }
431 { boolean }
432 { true }

433 defaultOptions.citationNbsps = true
```

`citations=true, false`

default: `false`

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

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

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

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

- `false` Disable the Pandoc citation syntax extension.

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

438 defaultOptions.citations = false
```

---

<sup>10</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.``
```

```
439 \@@_add_lua_option:nnn  
440 { codeSpans }  
441 { boolean }  
442 { true }  
  
443 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.

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

```
449 \@@_add_lua_option:nnn
450   { contentLevel }
451   { string }
452   { block }
453 defaultOptions.contentLevel = "block"
```

`debugExtensions=true, false`

default: `false`

**true** Produce a JSON file that will contain the extensible subset of the PEG grammar of markdown (see the `walkable_syntax` hash table) after built-in syntax extensions (see Section 3.1.7) and user-defined syntax extensions (see Section 2.1.2) have been applied. This helps you to see how the different extensions interact. The name of the produced JSON file is controlled by the `debugExtensionsFileName` option.

**false** Do not produce a JSON file with the PEG grammar of markdown.

```
454 \@@_add_lua_option:nnn
455   { debugExtensions }
456   { boolean }
457   { false }
458 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.

```

459 \@@_add_lua_option:nnn
460   { definitionLists }
461   { boolean }
462   { false }

463 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}
```

```
464 \@@_add_lua_option:nnn
465   { expectJekyllData }
466   { boolean }
467   { false }
468 defaultOptions.expectJekyllData = false
```

`extensions=<filenames>`

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

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

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

```

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

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

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

return strike_through

```

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

```

469 metadata.user_extension_api_version = 2
470 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.

```

471 \cs_generate_variant:Nn
472 \@@_add_lua_option:nnn
473 { nnV }
474 \@@_add_lua_option:nnV
475 { extensions }
476 { clist }
477 \c_empty_clist
478 defaultOptions.extensions = {}

```

`fancyLists=true, false`

default: false

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

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

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

```
479 \@@_add_lua_option:nnn
480 { fancyLists }
481 { boolean }
482 { false }
483 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.

```
484 \@@_add_lua_option:nnn
485 { fencedCode }
486 { boolean }
487 { true }
488 defaultOptions.fencedCode = true
```

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

`fencedCodeAttributes=true, false`

default: false

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

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

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

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

493 defaultOptions.fencedCodeAttributes = false
```

`fencedDivs=true, false`

default: false

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

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

And another.
:::::
```

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

```
494 \@@_add_lua_option:nnn
495 { fencedDivs }
496 { boolean }
497 { false }

498 defaultOptions.fencedDivs = false
```

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

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

`finalizeCache=true, false`

default: `false`

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

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

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

503 defaultOptions.finalizeCache = false
```

`frozenCacheCounter=<number>`

default: `0`

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

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

```
504 \@@_add_lua_option:nnn
505   { frozenCacheCounter }
506   { counter }
507   { 0 }

508 defaultOptions.frozenCacheCounter = 0
```

`gfmAutoIdentifiers=true, false`

default: `false`

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

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

# 123 Sesame Street
```

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

---

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

See also the option [autoIdentifiers](#).

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

513 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.

```
514 \@@_add_lua_option:nnn
515   { hashEnumerators }
516   { boolean }
517   { false }

518 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.

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

523 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.

```
524 \@@_add_lua_option:nnn
525   { html }
526   { boolean }
527   { true }

528 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.

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

533 defaultOptions.hybrid = false
```

`inlineCodeAttributes=true, false` default: false

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

``<$>`{.haskell}`

---

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



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

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

538 defaultOptions.inlineCodeAttributes = false
```

`inlineNotes=true, false` default: false

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

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

`false` Disable the Pandoc inline note syntax extension.

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

543 defaultOptions.inlineNotes = false
```

`jeekyllData=true, false` default: false

`true` Enable the Pandoc YAML metadata block syntax extension<sup>17</sup> for entering metadata in YAML:

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

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

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

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

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

```
544 \@@_add_lua_option:nnn
545 { jekyllData }
546 { boolean }
547 { false }
548 defaultOptions.jekyllData = false
```

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

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

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

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

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

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

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

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

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

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

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

---

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

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

`mark=true, false` default: false

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

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

`false` Disable the Pandoc mark syntax extension.

```
559 \@@_add_lua_option:nnn
560 { mark }
561 { boolean }
562 { false }
563 defaultOptions.mark = false
```

`notes=true, false` default: false

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

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

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

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

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

```
    { some.code }
```

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

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

`false` Disable the Pandoc note syntax extension.

```
564 \@@_add_lua_option:nnn
565 { notes }
566 { boolean }
567 { false }
568 defaultOptions.notes = false
```

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

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

`pipeTables=true, false`

default: false

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

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

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

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

573 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.

```
574 \@@_add_lua_option:nnn
575 { preserveTabs }
576 { boolean }
577 { true }

578 defaultOptions.preserveTabs = true
```

`rawAttribute=true, false`

default: false

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

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

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

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

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

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

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

`false`      Disable the Pandoc raw attribute syntax extension.

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

583 defaultOptions.rawAttribute = false
```

`relativeReferences=true, false`

default: `false`

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

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

Conclusion {#conclusion}
=====

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

`false`      Disable relative references in autolinks.

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

588 defaultOptions.relativeReferences = false
```

---

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

`shiftHeadings`=*<shift amount>*

default: 0

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

```
589 \@@_add_lua_option:nnn
590   { shiftHeadings }
591   { number }
592   { 0 }

593 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.

```
594 \@@_add_lua_option:nnn
595   { slice }
596   { slice }
597   { ^-$ }

598 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.

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

603 defaultOptions.smartEllipses = false
```

`startNumber=true, false` default: true

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

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

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

608 defaultOptions.startNumber = true
```

`strikeThrough=true, false` default: false

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

This ~~is deleted text.~~

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

```
609 \@@_add_lua_option:nnn
610 { strikeThrough }
611 { boolean }
612 { false }

613 defaultOptions.strikeThrough = false
```

---

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

`stripIndent=true, false`

default: `false`

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

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

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

```
614 \@@_add_lua_option:nmn
615   { stripIndent }
616   { boolean }
617   { false }
618 defaultOptions.stripIndent = false
```

`subscripts=true, false`

default: `false`

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

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

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

```
619 \@@_add_lua_option:nmn
620   { subscripts }
621   { boolean }
622   { false }
623 defaultOptions.subscripts = false
```

---

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



`superscripts=true, false`

default: `false`

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

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

`false` Disable the Pandoc superscript syntax extension.

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

628 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.

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

633 defaultOptions.tableAttributes = false
```

---

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

`tableCaptions=true, false`

default: `false`

`true`

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

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

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

`false` Disable the Pandoc table caption syntax extension.

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

638 defaultOptions.tableCaptions = false
```

`taskLists=true, false`

default: `false`

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

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

`false` Disable the Pandoc task list syntax extension.

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

643 defaultOptions.taskLists = false
```

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

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

`texComments=true, false`

default: false

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

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

Always enabled when `hybrid` is enabled.

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

```
644 \@@_add_lua_option:nnn
645   { texComments }
646   { boolean }
647   { false }
648 defaultOptions.texComments = false
```

`texMathDollars=true, false`

default: false

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

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

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

```
649 \@@_add_lua_option:nnn
650   { texMathDollars }
651   { boolean }
652   { false }
653 defaultOptions.texMathDollars = false
```

---

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

`texMathDoubleBackslash=true, false` default: false

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

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

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

```
654 \@@_add_lua_option:nnn
655   { texMathDoubleBackslash }
656   { boolean }
657   { false }

658 defaultOptions.texMathDoubleBackslash = false
```

`texMathSingleBackslash=true, false` default: false

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

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

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

```
659 \@@_add_lua_option:nnn
660   { texMathSingleBackslash }
661   { boolean }
662   { false }

663 defaultOptions.texMathSingleBackslash = false
```

`tightLists=true, false` default: true

**true** Unordered and ordered lists whose items do not consist of multiple paragraphs will be considered *tight*. Tight lists will produce tight renderers that may produce different output than lists that are not tight:

---

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

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

```

- This is
- a tight
- unordered list.

- This is

  not a tight

- unordered list.

```

**false** Unordered and ordered lists whose items consist of multiple paragraphs will be treated the same way as lists that consist of multiple paragraphs.

```

664 \@@_add_lua_option:nmn
665   { tightLists }
666   { boolean }
667   { true }

668 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.

```

669 \@@_add_lua_option:nmn
670   { underscores }
671   { boolean }
672   { true }
673 \ExplSyntaxOff

674 defaultOptions.underscores = true

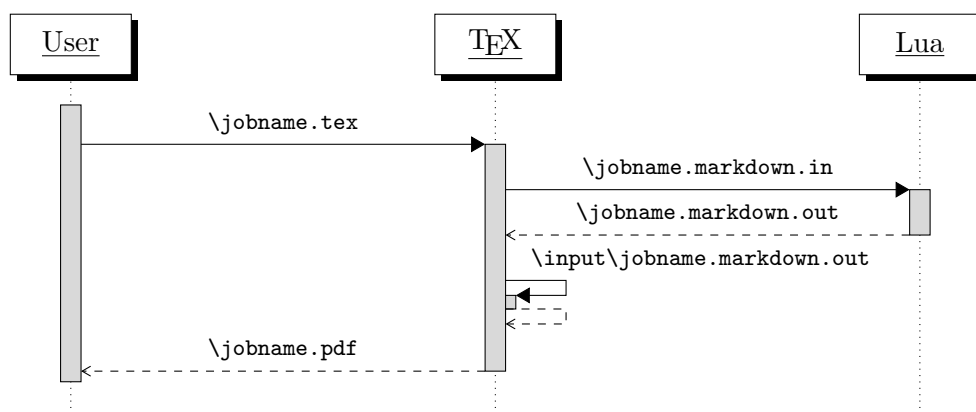
```

### 2.1.7 Command-Line Interface

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

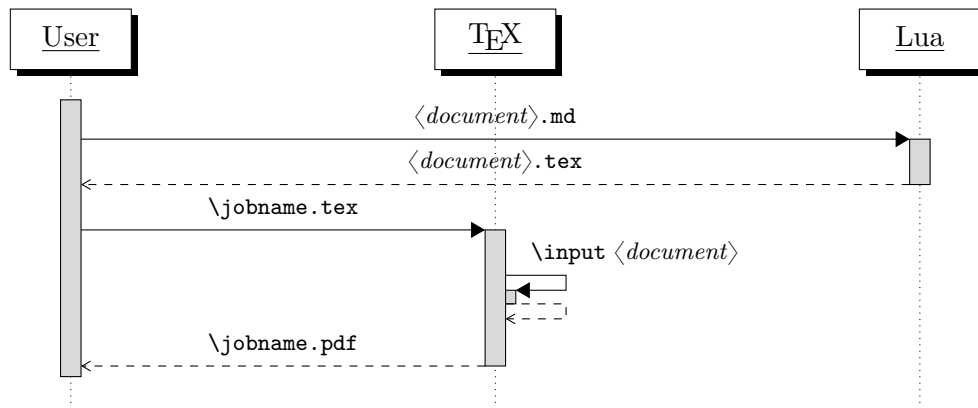
This procedure has the advantage of being fully automated. However, it also has several important disadvantages: The converted  $\text{T}_{\text{E}}\text{X}$  documents are cached on the file system, taking up increasing amount of space. Unless the  $\text{T}_{\text{E}}\text{X}$  engine includes a Lua 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  $\text{T}_{\text{E}}\text{X}$  is also provided, see Figure 3.



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

```
675
676 local HELP_STRING = [[
677 Usage: texlua ]] .. arg[0] .. [[ [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
678 where OPTIONS are documented in the Lua interface section of the
679 technical Markdown package documentation.
680
681 When OUTPUT_FILE is unspecified, the result of the conversion will be
682 written to the standard output. When INPUT_FILE is also unspecified, the
683 result of the conversion will be read from the standard input.
684
685 Report bugs to: witiko@mail.muni.cz
686 Markdown package home page: <https://github.com/witiko/markdown>]]
687
```



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

```

688 local VERSION_STRING = [[
689 markdown-cli.lua (Markdown) ]] .. metadata.version .. [[
690
691 Copyright (C) ]] .. table.concat(metadata.copyright,
692                                "\nCopyright (C) ") .. [[
693
694 License: ]] .. metadata.license
695
696 local function warn(s)
697   io.stderr:write("Warning: " .. s .. "\n") end
698
699 local function error(s)
700   io.stderr:write("Error: " .. s .. "\n")
701   os.exit(1)
702 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.

```

703 local function camel_case(option_name)
704   local cased_option_name = option_name:gsub("_(%1)", function(match)
705     return match:sub(2, 2):upper()
706   end)
707   return cased_option_name
708 end
709
710 local function snake_case(option_name)
711   local cased_option_name = option_name:gsub("%1%u", function(match)
712     return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()
  
```

```

713 end)
714 return cased_option_name
715 end
716
717 local cases = {camel_case, snake_case}
718 local various_case_options = {}
719 for option_name, _ in pairs(defaultOptions) do
720   for _, case in ipairs(cases) do
721     various_case_options[case(option_name)] = option_name
722   end
723 end
724
725 local process_options = true
726 local options = {}
727 local input_filename
728 local output_filename
729 for i = 1, #arg do
730   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.

```

731   if arg[i] == "--" then
732     process_options = false
733     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.

```

734   elseif arg[i]:match("=") then
735     local key, value = arg[i]:match("(.)=(.*)")
736     if defaultOptions[key] == nil and
737        various_case_options[key] ~= nil then
738       key = various_case_options[key]
739     end

```

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

```

740     local default_type = type(defaultOptions[key])
741     if default_type == "boolean" then
742       options[key] = (value == "true")
743     elseif default_type == "number" then
744       options[key] = tonumber(value)
745     elseif default_type == "table" then
746       options[key] = {}
747       for item in value:gmatch("[^,]+") do
748         table.insert(options[key], item)

```



```

749     end
750   else
751     if default_type ~= "string" then
752       if default_type == "nil" then
753         warn('Option "' .. key .. '" not recognized.')
754       else
755         warn('Option "' .. key .. '" type not recognized, please file ' ..
756             'a report to the package maintainer.')
757       end
758       warn('Parsing the ' .. 'value "' .. value ..'" of option "' ..
759           key .. '" as a string.')
760     end
761     options[key] = value
762   end
763   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.

```

764   elseif arg[i] == "--help" or arg[i] == "-h" then
765     print(HELP_STRING)
766     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.

```

767   elseif arg[i] == "--version" or arg[i] == "-v" then
768     print(VERSION_STRING)
769     os.exit()
770   end
771 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.

```

772 if input_filename == nil then
773   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.

```

774 elseif output_filename == nil then
775   output_filename = arg[i]
776 else
777   error('Unexpected argument: "' .. arg[i] .. "'.')
778 end
779 ::continue::
780 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 TeX document `hello.tex`. After the Markdown package for our TeX format has been loaded, the converted document can be typeset as follows:

```
\input hello
```

## 2.2 Plain TeX Interface

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

```
781 \def\markdownLastModified{((LASTMODIFIED))}%  
782 \def\markdownVersion{((VERSION))}%
```

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

```
\input markdown
```

It is expected that the special plain TeX characters have the expected category codes, when `\inputting` the file.

### 2.2.1 Typesetting Markdown

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

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

```
783 \let\markdownBegin\relax  
784 \let\markdownEnd\relax
```

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

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

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

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

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

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

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

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

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

```
785 \let\markinline\relax
```

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

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

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

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

```
\markdownEnd
\bye
```

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

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

```
786 \let\markdownInput\relax
```

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

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

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

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

```
787 \let\markdownEscape\relax
```

## 2.2.2 Options

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

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

```
788 \ExplSyntaxOn
789 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
790 \cs_generate_variant:Nn
791   \tl_const:Nn
792   { NV }
793 \tl_if_exist:NF
794   \c_@@_top_layer_tl
795   {
```

```

796     \tl_const:NV
797     \c_@@_top_layer_tl
798     \c_@@_option_layer_plain_tex_tl
799   }

```

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

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

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

```

801 \prop_new:N \g_@@_plain_tex_option_types_prop
802 \prop_new:N \g_@@_default_plain_tex_options_prop
803 \seq_gput_right:NV \g_@@_option_layers_seq \c_@@_option_layer_plain_tex_tl
804 \cs_new:Nn
805   \@@_add_plain_tex_option:nnn
806   {
807     \@@_add_option:Vnnn
808     \c_@@_option_layer_plain_tex_tl
809     { #1 }
810     { #2 }
811     { #3 }
812   }

```

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.

```

813 \cs_new:Nn
814   \@@_setup:n
815   {
816     \keys_set:nn
817       { markdown/options }
818       { #1 }
819   }
820 \cs_gset_eq:NN
821   \markdownSetup
822   \@@_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.

```

823 \prg_new_conditional:Nnn
824   \@@_if_option:n
825   { TF, T, F }
826   {

```

```

827 \@@_get_option_type:nN
828   { #1 }
829   \l_tmpa_tl
830 \str_if_eq:NNF
831   \l_tmpa_tl
832   \c_@@_option_type_boolean_tl
833   {
834     \msg_error:nxxx
835     { markdown }
836     { expected-boolean-option }
837     { #1 }
838     { \l_tmpa_tl }
839   }
840 \@@_get_option_value:nN
841   { #1 }
842   \l_tmpa_tl
843 \str_if_eq:NNTF
844   \l_tmpa_tl
845   \c_@@_option_value_true_tl
846   { \prg_return_true: }
847   { \prg_return_false: }
848 }
849 \msg_new:nnn
850 { markdown }
851 { expected-boolean-option }
852 {
853   Option~#1~has~type~#2,~
854   but~a~boolean~was~expected.
855 }
856 \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`.

```

857 \@@_add_plain_tex_option:nnn
858 { frozenCache }
859 { boolean }
860 { false }

```

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

1. Remove the `cacheDir` cache directory with stale auxiliary cache files.
2. Enable the `finalizeCache` option.
4. Typeset the plain `TeX` document to populate and finalize the cache.
5. Enable the `frozenCache` option.
6. Publish the source code of the plain `TeX` document and the `cacheDir` directory.

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

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

```
861 \@@_add_plain_tex_option:nnn
862   { inputTempFileName }
863   { path }
864   { \jobname.markdown.in }
```

The `\markdownOptionOutputDir` macro sets the path to the directory that will contain the auxiliary cache files produced by the Lua implementation and also the auxiliary files produced by the plain `TeX` implementation. The option defaults to `.` or, since `TeX Live 2024`, to the value of the `-output-directory` option of your `TeX` engine.

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

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

```
865 \cs_generate_variant:Nn
866   \@@_add_plain_tex_option:nnn
867   { 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`.

```
868 \ExplSyntaxOff
869 \input lt3luabridge.tex
870 \ExplSyntaxOn
871 \bool_if:nTF
872   {
873     \cs_if_exist_p:N
874       \luabridge_tl_set:Nn &&
```

```

875     (
876     \int_compare_p:nNn
877     { \g_luabridge_method_int }
878     =
879     { \c_luabridge_method_directlua_int } ||
880     \sys_if_shell_unrestricted_p:
881     )
882 }
883 {
884 \luabridge_tl_set:Nn
885 \l_tmpa_tl
886 { print(os.getenv("TEXMF_OUTPUT_DIRECTORY") or ".") }
887 }
888 {
889 \tl_set:Nn
890 \l_tmpa_tl
891 { . }
892 }
893 \@@_add_plain_tex_option:nnV
894 { outputDir }
895 { path }
896 \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.

```
897 \@@_add_plain_tex_option:nnn
898   { plain }
899   { boolean }
900   { 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  $\LaTeX$  document:

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

Here is how you would enable the macro in a ConTeXt document:

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

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

```
901 \@@_add_plain_tex_option:nnn
902   { noDefaults }
903   { boolean }
904   { false }
```

#### 2.2.2.4 Miscellaneous Options

The `\markdownOptionStripPercentSigns` macro controls whether a percent sign (%) at the beginning of a line will be discarded when buffering Markdown input (see sections 3.2.5 and 3.2.6) or not. Notably, this enables the use of markdown when writing  $\TeX$  package documentation using the Doc  $\LaTeX$  package [7] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```
905 \seq_gput_right:Nn
906   \g_@@_plain_tex_options_seq
907   { stripPercentSigns }
908 \prop_gput:Nnn
909   \g_@@_plain_tex_option_types_prop
```

```

910 { stripPercentSigns }
911 { boolean }
912 \prop_gput:Nnx
913 \g_@@_default_plain_tex_options_prop
914 { stripPercentSigns }
915 { 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.

```

916 \cs_new:Nn
917 \@@_define_option_commands_and_keyvals:
918 {
919   \seq_map_inline:Nn
920     \g_@@_option_layers_seq
921     {
922       \seq_map_inline:cn
923         { g_@@_ ##1 _options_seq }
924         {
925           \@@_define_option_command:n
926             { #####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`.

```

927         \@@_with_various_cases:nn
928         { #####1 }
929         {
930           \@@_define_option_keyval:nnn
931             { ##1 }
932             { #####1 }
933             { #####1 }
934         }
935     }

```

```

936     }
937 }
938 \cs_new:Nn
939   \@@_define_option_command:n
940   {

```

Do not override options defined before loading the package.

```

941   \@@_option_tl_to_csname:nN
942   { #1 }
943   \l_tmpa_tl
944   \cs_if_exist:cF
945   { \l_tmpa_tl }
946   {
947     \@@_get_default_option_value:nN
948     { #1 }
949     \l_tmpa_tl
950     \@@_set_option_value:nV
951     { #1 }
952     \l_tmpa_tl
953   }
954 }
955 \cs_new:Nn
956   \@@_set_option_value:nn
957   {
958     \@@_define_option:n
959     { #1 }
960     \@@_get_option_type:nN
961     { #1 }
962     \l_tmpa_tl
963     \str_if_eq:NNTF
964     \c_@@_option_type_counter_tl
965     \l_tmpa_tl
966     {
967       \@@_option_tl_to_csname:nN
968       { #1 }
969       \l_tmpa_tl
970       \int_gset:cn
971       { \l_tmpa_tl }
972       { #2 }
973     }
974     {
975       \@@_option_tl_to_csname:nN
976       { #1 }
977       \l_tmpa_tl
978       \cs_set:cpn
979       { \l_tmpa_tl }
980       { #2 }
981     }

```

```

982 }
983 \cs_generate_variant:Nn
984 \@@_set_option_value:nn
985 { nV }
986 \cs_new:Nn
987 \@@_define_option:n
988 {
989   \@@_option_tl_to_csname:nN
990   { #1 }
991   \l_tmpa_tl
992   \cs_if_free:cT
993   { \l_tmpa_tl }
994   {
995     \@@_get_option_type:nN
996     { #1 }
997     \l_tmpb_tl
998     \str_if_eq:NNT
999     \c_@@_option_type_counter_tl
1000     \l_tmpb_tl
1001     {
1002       \@@_option_tl_to_csname:nN
1003       { #1 }
1004       \l_tmpa_tl
1005       \int_new:c
1006       { \l_tmpa_tl }
1007     }
1008   }
1009 }
1010 \cs_new:Nn
1011 \@@_define_option_keyval:nnn
1012 {
1013   \prop_get:cnN
1014   { g_@@_ #1 _option_types_prop }
1015   { #2 }
1016   \l_tmpa_tl
1017   \str_if_eq:VVTF
1018   \l_tmpa_tl
1019   \c_@@_option_type_boolean_tl
1020   {
1021     \keys_define:nn
1022     { markdown/options }
1023     {

```

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

```

1024         #3 .code:n = {
1025             \tl_set:Nx

```

```

1026         \l_tmpa_tl
1027         {
1028             \str_case:nnF
1029             { ##1 }
1030             {
1031                 { yes } { true }
1032                 { no } { false }
1033             }
1034             { ##1 }
1035         }
1036         \@@_set_option_value:nV
1037         { #2 }
1038         \l_tmpa_tl
1039     },
1040     #3 .default:n = { true },
1041 }
1042 }
1043 {
1044     \keys_define:nn
1045     { markdown/options }
1046     {
1047         #3 .code:n = {
1048             \@@_set_option_value:nn
1049             { #2 }
1050             { ##1 }
1051         },
1052     }
1053 }

```

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.

```

1054     \str_if_eq:VVT
1055     \l_tmpa_tl
1056     \c_@@_option_type_clist_tl
1057     {
1058         \tl_set:Nn
1059         \l_tmpa_tl
1060         { #3 }
1061         \tl_reverse:N
1062         \l_tmpa_tl
1063         \str_if_eq:enF
1064         {
1065             \tl_head:V
1066             \l_tmpa_tl

```

```

1067     }
1068     { s }
1069     {
1070         \msg_error:nnn
1071         { markdown }
1072         { malformed-name-for-clist-option }
1073         { #3 }
1074     }
1075     \tl_set:Nx
1076     \l_tmpa_tl
1077     {
1078         \tl_tail:V
1079         \l_tmpa_tl
1080     }
1081     \tl_reverse:N
1082     \l_tmpa_tl
1083     \tl_put_right:Nn
1084     \l_tmpa_tl
1085     {
1086         .code:n = {
1087             \@@_get_option_value:nN
1088             { #2 }
1089             \l_tmpa_tl
1090             \clist_set:NV
1091             \l_tmpa_clist
1092             { \l_tmpa_tl, { ##1 } }
1093             \@@_set_option_value:nV
1094             { #2 }
1095             \l_tmpa_clist
1096         }
1097     }
1098     \keys_define:nV
1099     { markdown/options }
1100     \l_tmpa_tl
1101 }
1102 }
1103 \cs_generate_variant:Nn
1104 \clist_set:Nn
1105 { NV }
1106 \cs_generate_variant:Nn
1107 \keys_define:nn
1108 { nV }
1109 \cs_generate_variant:Nn
1110 \@@_set_option_value:nn
1111 { nV }
1112 \prg_generate_conditional_variant:Nnn
1113 \str_if_eq:nn

```

```

1114 { en }
1115 { F }
1116 \msg_new:nnn
1117 { markdown }
1118 { malformed-name-for-clist-option }
1119 {
1120   Clist-option-name~#1~does~not~end~with~-s.
1121 }

```

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

```

1122 \str_if_eq:VVT
1123 \c_@@_top_layer_tl
1124 \c_@@_option_layer_plain_tex_tl
1125 {
1126   \@@_define_option_commands_and_keyvals:
1127 }
1128 \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 TeX document (further referred to as *a theme*) named `markdowntheme<munged theme name>.tex`, where the *munged theme name* is the *theme name* after the substitution of all forward slashes (/) for an underscore (\_). The theme name 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 TeX directory structure. For example, loading a theme named `witiko/beamer/MU` would load a TeX document package named `markdownthemewitiko_beamer_MU.tex`.

```

1129 \ExplSyntaxOn
1130 \keys_define:nn
1131 { markdown/options }

```

```

1132 {
1133   theme .code:n = {
1134     \@@_set_theme:n
1135     { #1 }
1136   },
1137   import .code:n = {
1138     \tl_set:Nn
1139     \l_tmpa_tl
1140     { #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.

```

1141     \tl_replace_all:NnV
1142     \l_tmpa_tl
1143     { / }
1144     \c_backslash_str
1145     \keys_set:nV
1146     { markdown/options/import }
1147     \l_tmpa_tl
1148   },
1149 }

```

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.

```

1150 \seq_new:N
1151 \g_@@_themes_seq
1152 \tl_new:N
1153 \g_@@_current_theme_tl
1154 \tl_gset:Nn
1155 \g_@@_current_theme_tl
1156 { }
1157 \seq_gput_right:NV
1158 \g_@@_themes_seq
1159 \g_@@_current_theme_tl
1160 \cs_new:Nn
1161 \@@_set_theme:n
1162 {

```

First, we validate the theme name.

```

1163 \str_if_in:nnF
1164 { #1 }
1165 { / }
1166 {
1167   \msg_error:nnn

```



```

1168         { markdown }
1169         { unqualified-theme-name }
1170         { #1 }
1171     }
1172 \str_if_in:nnT
1173     { #1 }
1174     { _ }
1175     {
1176         \msg_error:nnn
1177         { markdown }
1178         { underscores-in-theme-name }
1179         { #1 }
1180     }

```

Next, we munge the theme name.

```

1181 \str_set:Nn
1182     \l_tmpa_str
1183     { #1 }
1184 \str_replace_all:Nnn
1185     \l_tmpa_str
1186     { / }
1187     { _ }

```

Finally, we load the theme.

```

1188 \tl_gset:Nn
1189     \g_@@_current_theme_tl
1190     { #1 / }
1191 \seq_gput_right:NV
1192     \g_@@_themes_seq
1193     \g_@@_current_theme_tl
1194 \@@_load_theme:nV
1195     { #1 }
1196     \l_tmpa_str
1197 \seq_gpop_right:NN
1198     \g_@@_themes_seq
1199     \l_tmpa_tl
1200 \seq_get_right:NN
1201     \g_@@_themes_seq
1202     \l_tmpa_tl
1203 \tl_gset:NV
1204     \g_@@_current_theme_tl
1205     \l_tmpa_tl
1206 }
1207 \msg_new:nnnn
1208 { markdown }
1209 { unqualified-theme-name }
1210 { Won't load theme with unqualified name #1 }
1211 { Theme names must contain at least one forward slash }

```

```

1212 \msg_new:nnnn
1213   { markdown }
1214   { underscores-in-theme-name }
1215   { Won't-load-theme-with-an-underscore-in-its-name~#1 }
1216   { Theme-names-must-not-contain-underscores-in-their-names }
1217 \cs_generate_variant:Nn
1218   \tl_replace_all:Nnn
1219   { NnV }
1220 \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.

```

1221 \ExplSyntaxOn
1222 \prop_new:N
1223   \g_@@_snippets_prop
1224 \cs_new:Nn
1225   \@@_setup_snippet:nn
1226   {
1227     \tl_if_empty:nT
1228       { #1 }
1229     {

```

```

1230     \msg_error:nnn
1231     { markdown }
1232     { empty-snippet-name }
1233     { #1 }
1234   }
1235   \tl_set:NV
1236     \l_tmpa_tl
1237     \g_@@_current_theme_tl
1238   \tl_put_right:Nn
1239     \l_tmpa_tl
1240     { #1 }
1241   \@@_if_snippet_exists:nT
1242     { #1 }
1243     {
1244       \msg_warning:nnV
1245       { markdown }
1246       { redefined-snippet }
1247       \l_tmpa_tl
1248     }
1249   \prop_gput:NVn
1250     \g_@@_snippets_prop
1251     \l_tmpa_tl
1252     { #2 }
1253 }
1254 \cs_gset_eq:NN
1255   \markdownSetupSnippet
1256   \@@_setup_snippet:nn
1257   \msg_new:nnnn
1258   { markdown }
1259   { empty-snippet-name }
1260   { Empty-snippet-name~#1 }
1261   { Pick~a~non-empty~name~for~your~snippet }
1262   \msg_new:nnn
1263   { markdown }
1264   { redefined-snippet }
1265   { Redefined~snippet~#1 }

```

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

```

1266 \prg_new_conditional:Nnn
1267   \@@_if_snippet_exists:n
1268   { TF, T, F }
1269   {
1270     \tl_set:NV
1271       \l_tmpa_tl
1272       \g_@@_current_theme_tl
1273     \tl_put_right:Nn
1274       \l_tmpa_tl

```

```

1275     { #1 }
1276     \prop_get:NVNTF
1277     \g_@@_snippets_prop
1278     \l_tmpa_tl
1279     \l_tmpb_tl
1280     { \prg_return_true: }
1281     { \prg_return_false: }
1282   }
1283 \cs_gset_eq:NN
1284 \markdownIfSnippetExists
1285 \@@_if_snippet_exists:nTF

```

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

```

1286 \keys_define:nn
1287   { markdown/options }
1288   {
1289     snippet .code:n = {
1290       \tl_set:NV
1291         \l_tmpa_tl
1292         \g_@@_current_theme_tl
1293       \tl_put_right:Nn
1294         \l_tmpa_tl
1295         { #1 }
1296       \@@_if_snippet_exists:nTF
1297         { #1 }
1298         {
1299           \prop_get:NVN
1300             \g_@@_snippets_prop
1301             \l_tmpa_tl
1302             \l_tmpb_tl
1303           \@@_setup:V
1304             \l_tmpb_tl
1305         }
1306         {
1307           \msg_error:nnV
1308             { markdown }
1309             { undefined-snippet }
1310             \l_tmpa_tl
1311         }
1312     }
1313   }
1314 \msg_new:nnn
1315   { markdown }
1316   { undefined-snippet }
1317   { Can't~invoke~undefined~snippet~#1 }
1318 \cs_generate_variant:Nn
1319   \@@_setup:n
1320   { V }

```

1321 \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 `jdoue/lists` theme, we could import the `jdoue/lists` theme and use the qualified name `jdoue/lists/romanNumerals` to invoke the snippet:

```
\markdownSetup{import=jdoue/lists}
\begin{markdown}[snippet=jdoue/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`  $\LaTeX$  option:

```

\markdownSetup{
  import = {
    jdoe/longpackagename/lists = {
      arabic as arabic1,

```

```

    roman,
    alphabetic,
  },
  jdoe/anotherlongpackagename/lists = {
    arabic as arabic2,
  },
  jdoe/yetanotherlongpackagename,
},
}

```

```

1322 \ExplSyntaxOn
1323 \tl_new:N
1324 \l_@@_import_current_theme_tl
1325 \keys_define:nn
1326 { markdown/options/import }
1327 {

```

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

```

1328     unknown .default:n = {},
1329     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.

```

1330     \tl_set_eq:NN
1331     \l_@@_import_current_theme_tl
1332     \l_keys_key_str
1333     \tl_replace_all:NVN
1334     \l_@@_import_current_theme_tl
1335     \c_backslash_str
1336     { / }

```

Here, we import the snippets.

```

1337     \clist_map_inline:nn
1338     { #1 }
1339     {
1340         \regex_extract_once:nnNTF
1341         { ^(.*)\s+as\s+(.*)$ }
1342         { ##1 }
1343         \l_tmpa_seq
1344         {
1345             \seq_pop:NN
1346             \l_tmpa_seq

```

```

1347         \l_tmpa_tl
1348         \seq_pop:NN
1349         \l_tmpa_seq
1350         \l_tmpa_tl
1351         \seq_pop:NN
1352         \l_tmpa_seq
1353         \l_tmpb_tl
1354     }
1355     {
1356         \tl_set:Nn
1357         \l_tmpa_tl
1358         { ##1 }
1359         \tl_set:Nn
1360         \l_tmpb_tl
1361         { ##1 }
1362     }
1363     \tl_put_left:Nn
1364     \l_tmpa_tl
1365     { / }
1366     \tl_put_left:NV
1367     \l_tmpa_tl
1368     \l_@@_import_current_theme_tl
1369     \@@_setup_snippet:Vx
1370     \l_tmpb_tl
1371     { snippet = { \l_tmpa_tl } }
1372 }

```

Here, we load the theme.

```

1373     \@@_set_theme:V
1374     \l_@@_import_current_theme_tl
1375 },
1376 }
1377 \cs_generate_variant:Nn
1378 \tl_replace_all:Nnn
1379 { NVn }
1380 \cs_generate_variant:Nn
1381 \@@_set_theme:n
1382 { V }
1383 \cs_generate_variant:Nn
1384 \@@_setup_snippet:nn
1385 { Vx }

```

## 2.2.5 Token Renderers

The following  $\TeX$  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.

```
1386 \ExplSyntaxOn
1387 \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.

```
1388 \prop_new:N \g_@@_renderer_arities_prop
1389 \ExplSyntaxOff
```

### 2.2.5.1 Attribute Renderers

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

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

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

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

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

```
1390 \def\markdownRendererAttributeIdentifier{%
1391   \markdownRendererAttributeIdentifierPrototype}%
1392 \ExplSyntaxOn
1393 \seq_gput_right:Nn
1394   \g_@@_renderers_seq
1395   { attributeIdentifier }
1396 \prop_gput:Nnn
1397   \g_@@_renderer_arities_prop
1398   { attributeIdentifier }
1399   { 1 }
1400 \ExplSyntaxOff
1401 \def\markdownRendererAttributeName{%
```

```

1402 \markdownRendererAttributeClassNamePrototype}%
1403 \ExplSyntaxOn
1404 \seq_gput_right:Nn
1405 \g_@@_renderers_seq
1406 { attributeClassName }
1407 \prop_gput:Nnn
1408 \g_@@_renderer_arities_prop
1409 { attributeClassName }
1410 { 1 }
1411 \ExplSyntaxOff
1412 \def\markdownRendererAttributeKeyValue{%
1413 \markdownRendererAttributeKeyValuePrototype}%
1414 \ExplSyntaxOn
1415 \seq_gput_right:Nn
1416 \g_@@_renderers_seq
1417 { attributeKeyValue }
1418 \prop_gput:Nnn
1419 \g_@@_renderer_arities_prop
1420 { attributeKeyValue }
1421 { 2 }
1422 \ExplSyntaxOff

```

### 2.2.5.2 Block Quote Renderers

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

```

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

```

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

```

1434 \def\markdownRendererBlockQuoteEnd{%
1435 \markdownRendererBlockQuoteEndPrototype}%
1436 \ExplSyntaxOn
1437 \seq_gput_right:Nn
1438 \g_@@_renderers_seq
1439 { blockQuoteEnd }
1440 \prop_gput:Nnn

```

```

1441 \g_@@_renderer_arities_prop
1442 { blockQuoteEnd }
1443 { 0 }
1444 \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.

```

1445 \def\markdownRendererBracketedSpanAttributeContextBegin{%
1446   \markdownRendererBracketedSpanAttributeContextBeginPrototype}%
1447 \ExplSyntaxOn
1448 \seq_gput_right:Nn
1449   \g_@@_renderers_seq
1450   { bracketedSpanAttributeContextBegin }
1451 \prop_gput:Nnn
1452   \g_@@_renderer_arities_prop
1453   { bracketedSpanAttributeContextBegin }
1454   { 0 }
1455 \ExplSyntaxOff
1456 \def\markdownRendererBracketedSpanAttributeContextEnd{%
1457   \markdownRendererBracketedSpanAttributeContextEndPrototype}%
1458 \ExplSyntaxOn
1459 \seq_gput_right:Nn
1460   \g_@@_renderers_seq
1461   { bracketedSpanAttributeContextEnd }
1462 \prop_gput:Nnn
1463   \g_@@_renderer_arities_prop
1464   { bracketedSpanAttributeContextEnd }
1465   { 0 }
1466 \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.

```

1467 \def\markdownRendererUlBegin{%
1468   \markdownRendererUlBeginPrototype}%
1469 \ExplSyntaxOn
1470 \seq_gput_right:Nn
1471   \g_@@_renderers_seq
1472   { ulBegin }

```

```

1473 \prop_gput:Nnn
1474   \g_@@_renderer_arities_prop
1475   { ulBegin }
1476   { 0 }
1477 \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.

```

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

```

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

```

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

```

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

```

1500 \def\markdownRendererUlItemEnd{%
1501   \markdownRendererUlItemEndPrototype}%
1502 \ExplSyntaxOn
1503 \seq_gput_right:Nn
1504   \g_@@_renderers_seq
1505   { ulItemEnd }
1506 \prop_gput:Nnn
1507   \g_@@_renderer_arities_prop

```

```

1508 { ulItemEnd }
1509 { 0 }
1510 \ExplSyntaxOff

```

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

```

1511 \def\markdownRendererUEnd{%
1512   \markdownRendererUEndPrototype}%
1513 \ExplSyntaxOn
1514 \seq_gput_right:Nn
1515   \g_@@_renderers_seq
1516   { ulEnd }
1517 \prop_gput:Nnn
1518   \g_@@_renderer_arities_prop
1519   { ulEnd }
1520   { 0 }
1521 \ExplSyntaxOff

```

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

```

1522 \def\markdownRendererUEndTight{%
1523   \markdownRendererUEndTightPrototype}%
1524 \ExplSyntaxOn
1525 \seq_gput_right:Nn
1526   \g_@@_renderers_seq
1527   { ulEndTight }
1528 \prop_gput:Nnn
1529   \g_@@_renderer_arities_prop
1530   { ulEndTight }
1531   { 0 }
1532 \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.

```

1533 \def\markdownRendererCite{%
1534   \markdownRendererCitePrototype}%
1535 \ExplSyntaxOn

```

```

1536 \seq_gput_right:Nn
1537   \g_@@_renderers_seq
1538   { cite }
1539 \prop_gput:Nnn
1540   \g_@@_renderer_arities_prop
1541   { cite }
1542   { 1 }
1543 \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.

```

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

```

1555 \def\markdownRendererInputVerbatim{%
1556   \markdownRendererInputVerbatimPrototype}%
1557 \ExplSyntaxOn
1558 \seq_gput_right:Nn
1559   \g_@@_renderers_seq
1560   { inputVerbatim }
1561 \prop_gput:Nnn
1562   \g_@@_renderer_arities_prop
1563   { inputVerbatim }
1564   { 1 }
1565 \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.

```

1566 \def\markdownRendererInputFencedCode{%
1567   \markdownRendererInputFencedCodePrototype}%
1568 \ExplSyntaxOn
1569 \seq_gput_right:Nn
1570   \g_@@_renderers_seq
1571   { inputFencedCode }
1572 \prop_gput:Nnn
1573   \g_@@_renderer_arities_prop
1574   { inputFencedCode }
1575   { 3 }
1576 \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.

```

1577 \def\markdownRendererCodeSpan{%
1578   \markdownRendererCodeSpanPrototype}%
1579 \ExplSyntaxOn
1580 \seq_gput_right:Nn
1581   \g_@@_renderers_seq
1582   { codeSpan }
1583 \prop_gput:Nnn
1584   \g_@@_renderer_arities_prop
1585   { codeSpan }
1586   { 1 }
1587 \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.

```

1588 \def\markdownRendererCodeSpanAttributeContextBegin{%
1589   \markdownRendererCodeSpanAttributeContextBeginPrototype}%
1590 \ExplSyntaxOn
1591 \seq_gput_right:Nn
1592   \g_@@_renderers_seq
1593   { codeSpanAttributeContextBegin }
1594 \prop_gput:Nnn
1595   \g_@@_renderer_arities_prop
1596   { codeSpanAttributeContextBegin }

```

```

1597 { 0 }
1598 \ExplSyntaxOff
1599 \def\markdownRendererCodeSpanAttributeContextEnd{%
1600 \markdownRendererCodeSpanAttributeContextEndPrototype}%
1601 \ExplSyntaxOn
1602 \seq_gput_right:Nn
1603 \g_@@_renderers_seq
1604 { codeSpanAttributeContextEnd }
1605 \prop_gput:Nnn
1606 \g_@@_renderer_arities_prop
1607 { codeSpanAttributeContextEnd }
1608 { 0 }
1609 \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.

```

1610 \def\markdownRendererContentBlock{%
1611 \markdownRendererContentBlockPrototype}%
1612 \ExplSyntaxOn
1613 \seq_gput_right:Nn
1614 \g_@@_renderers_seq
1615 { contentBlock }
1616 \prop_gput:Nnn
1617 \g_@@_renderer_arities_prop
1618 { contentBlock }
1619 { 4 }
1620 \ExplSyntaxOff

```

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

```

1621 \def\markdownRendererContentBlockOnlineImage{%
1622 \markdownRendererContentBlockOnlineImagePrototype}%
1623 \ExplSyntaxOn
1624 \seq_gput_right:Nn
1625 \g_@@_renderers_seq
1626 { contentBlockOnlineImage }
1627 \prop_gput:Nnn
1628 \g_@@_renderer_arities_prop
1629 { contentBlockOnlineImage }
1630 { 4 }
1631 \ExplSyntaxOff

```



The `\markdownRendererContentBlockCode` macro represents an iA Writer content block that was recognized as a file in a known programming language by its filename extension  $s$ . If any `markdown-languages.json` file found by `kpathsea`<sup>32</sup> contains a record  $(k, v)$ , then a non-online-image content block with the filename extension  $s$ ,  $s:\text{lower}() = k$  is considered to be in a known programming language  $v$ . The macro receives five arguments: the local file name extension  $s$  cast to the lower case, the language  $v$ , the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the content block.

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

```

1632 \def\markdownRendererContentBlockCode{%
1633   \markdownRendererContentBlockCodePrototype}%
1634 \ExplSyntaxOn
1635 \seq_gput_right:Nn
1636   \g_@@_renderers_seq
1637   { contentBlockCode }
1638 \prop_gput:Nnn
1639   \g_@@_renderer_arities_prop
1640   { contentBlockCode }
1641   { 5 }
1642 \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.

```

1643 \def\markdownRendererDlBegin{%
1644   \markdownRendererDlBeginPrototype}%
1645 \ExplSyntaxOn
1646 \seq_gput_right:Nn
1647   \g_@@_renderers_seq
1648   { dlBegin }
1649 \prop_gput:Nnn
1650   \g_@@_renderer_arities_prop
1651   { dlBegin }

```

---

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

```

1652 { 0 }
1653 \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.

```

1654 \def\markdownRendererDlBeginTight{%
1655   \markdownRendererDlBeginTightPrototype}%
1656 \ExplSyntaxOn
1657 \seq_gput_right:Nn
1658   \g_@@_renderers_seq
1659   { dlBeginTight }
1660 \prop_gput:Nnn
1661   \g_@@_renderer_arities_prop
1662   { dlBeginTight }
1663   { 0 }
1664 \ExplSyntaxOff

```

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

```

1665 \def\markdownRendererDlItem{%
1666   \markdownRendererDlItemPrototype}%
1667 \ExplSyntaxOn
1668 \seq_gput_right:Nn
1669   \g_@@_renderers_seq
1670   { dlItem }
1671 \prop_gput:Nnn
1672   \g_@@_renderer_arities_prop
1673   { dlItem }
1674   { 1 }
1675 \ExplSyntaxOff

```

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

```

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

```

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

```
1687 \def\markdownRendererDlDefinitionBegin{%
1688   \markdownRendererDlDefinitionBeginPrototype}%
1689 \ExplSyntaxOn
1690 \seq_gput_right:Nn
1691   \g_@@_renderers_seq
1692   { dlDefinitionBegin }
1693 \prop_gput:Nnn
1694   \g_@@_renderer_arities_prop
1695   { dlDefinitionBegin }
1696   { 0 }
1697 \ExplSyntaxOff
```

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

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

```
1709 \def\markdownRendererDlEnd{%
1710   \markdownRendererDlEndPrototype}%
1711 \ExplSyntaxOn
1712 \seq_gput_right:Nn
1713   \g_@@_renderers_seq
1714   { dlEnd }
1715 \prop_gput:Nnn
1716   \g_@@_renderer_arities_prop
1717   { dlEnd }
1718   { 0 }
1719 \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.

```
1720 \def\markdownRendererDlEndTight{%
1721   \markdownRendererDlEndTightPrototype}%
1722 \ExplSyntaxOn
1723 \seq_gput_right:Nn
1724   \g_@@_renderers_seq
1725   { dlEndTight }
1726 \prop_gput:Nnn
1727   \g_@@_renderer_arities_prop
1728   { dlEndTight }
1729   { 0 }
1730 \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.

```
1731 \def\markdownRendererEllipsis{%
1732   \markdownRendererEllipsisPrototype}%
1733 \ExplSyntaxOn
1734 \seq_gput_right:Nn
1735   \g_@@_renderers_seq
1736   { ellipsis }
1737 \prop_gput:Nnn
1738   \g_@@_renderer_arities_prop
1739   { ellipsis }
1740   { 0 }
1741 \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.

```
1742 \def\markdownRendererEmphasis{%
1743   \markdownRendererEmphasisPrototype}%
1744 \ExplSyntaxOn
1745 \seq_gput_right:Nn
1746   \g_@@_renderers_seq
1747   { emphasis }
1748 \prop_gput:Nnn
1749   \g_@@_renderer_arities_prop
1750   { emphasis }
1751   { 1 }
```

```
1752 \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.

```
1753 \def\markdownRendererStrongEmphasis{%
1754   \markdownRendererStrongEmphasisPrototype}%
1755 \ExplSyntaxOn
1756 \seq_gput_right:Nn
1757   \g_@@_renderers_seq
1758   { strongEmphasis }
1759 \prop_gput:Nnn
1760   \g_@@_renderer_arities_prop
1761   { strongEmphasis }
1762   { 1 }
1763 \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.

```
1764 \def\markdownRendererFencedCodeAttributeContextBegin{%
1765   \markdownRendererFencedCodeAttributeContextBeginPrototype}%
1766 \ExplSyntaxOn
1767 \seq_gput_right:Nn
1768   \g_@@_renderers_seq
1769   { fencedCodeAttributeContextBegin }
1770 \prop_gput:Nnn
1771   \g_@@_renderer_arities_prop
1772   { fencedCodeAttributeContextBegin }
1773   { 0 }
1774 \ExplSyntaxOff
1775 \def\markdownRendererFencedCodeAttributeContextEnd{%
1776   \markdownRendererFencedCodeAttributeContextEndPrototype}%
1777 \ExplSyntaxOn
1778 \seq_gput_right:Nn
1779   \g_@@_renderers_seq
1780   { fencedCodeAttributeContextEnd }
1781 \prop_gput:Nnn
1782   \g_@@_renderer_arities_prop
1783   { fencedCodeAttributeContextEnd }
1784   { 0 }
1785 \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.

```
1786 \def\markdownRendererFencedDivAttributeContextBegin{%
1787   \markdownRendererFencedDivAttributeContextBeginPrototype}%
1788 \ExplSyntaxOn
1789 \seq_gput_right:Nn
1790   \g_@@_renderers_seq
1791   { fencedDivAttributeContextBegin }
1792 \prop_gput:Nnn
1793   \g_@@_renderer_arities_prop
1794   { fencedDivAttributeContextBegin }
1795   { 0 }
1796 \ExplSyntaxOff
1797 \def\markdownRendererFencedDivAttributeContextEnd{%
1798   \markdownRendererFencedDivAttributeContextEndPrototype}%
1799 \ExplSyntaxOn
1800 \seq_gput_right:Nn
1801   \g_@@_renderers_seq
1802   { fencedDivAttributeContextEnd }
1803 \prop_gput:Nnn
1804   \g_@@_renderer_arities_prop
1805   { fencedDivAttributeContextEnd }
1806   { 0 }
1807 \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.

```
1808 \def\markdownRendererHeaderAttributeContextBegin{%
1809   \markdownRendererHeaderAttributeContextBeginPrototype}%
1810 \ExplSyntaxOn
1811 \seq_gput_right:Nn
1812   \g_@@_renderers_seq
1813   { headerAttributeContextBegin }
1814 \prop_gput:Nnn
1815   \g_@@_renderer_arities_prop
1816   { headerAttributeContextBegin }
1817   { 0 }
1818 \ExplSyntaxOff
```

```

1819 \def\markdownRendererHeaderAttributeContextEnd{%
1820   \markdownRendererHeaderAttributeContextEndPrototype}%
1821 \ExplSyntaxOn
1822 \seq_gput_right:Nn
1823   \g_@@_renderers_seq
1824   { headerAttributeContextEnd }
1825 \prop_gput:Nnn
1826   \g_@@_renderer_arities_prop
1827   { headerAttributeContextEnd }
1828   { 0 }
1829 \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.

```

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

```

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

```

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

```

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

```

1852 \def\markdownRendererHeadingThree{%
1853   \markdownRendererHeadingThreePrototype}%
1854 \ExplSyntaxOn

```

```

1855 \seq_gput_right:Nn
1856   \g_@@_renderers_seq
1857   { headingThree }
1858 \prop_gput:Nnn
1859   \g_@@_renderer_arities_prop
1860   { headingThree }
1861   { 1 }
1862 \ExplSyntaxOff

```

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

```

1863 \def\markdownRendererHeadingFour{%
1864   \markdownRendererHeadingFourPrototype}%
1865 \ExplSyntaxOn
1866 \seq_gput_right:Nn
1867   \g_@@_renderers_seq
1868   { headingFour }
1869 \prop_gput:Nnn
1870   \g_@@_renderer_arities_prop
1871   { headingFour }
1872   { 1 }
1873 \ExplSyntaxOff

```

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

```

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

```

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

```

1885 \def\markdownRendererHeadingSix{%
1886   \markdownRendererHeadingSixPrototype}%
1887 \ExplSyntaxOn
1888 \seq_gput_right:Nn
1889   \g_@@_renderers_seq
1890   { headingSix }
1891 \prop_gput:Nnn
1892   \g_@@_renderer_arities_prop

```



```

1893 { headingSix }
1894 { 1 }
1895 \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.

```

1896 \def\markdownRendererInlineHtmlComment{%
1897   \markdownRendererInlineHtmlCommentPrototype}%
1898 \ExplSyntaxOn
1899 \seq_gput_right:Nn
1900   \g_@@_renderers_seq
1901   { inlineHtmlComment }
1902 \prop_gput:Nnn
1903   \g_@@_renderer_arities_prop
1904   { inlineHtmlComment }
1905   { 1 }
1906 \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.

```

1907 \def\markdownRendererInlineHtmlTag{%
1908   \markdownRendererInlineHtmlTagPrototype}%
1909 \ExplSyntaxOn
1910 \seq_gput_right:Nn
1911   \g_@@_renderers_seq
1912   { inlineHtmlTag }
1913 \prop_gput:Nnn
1914   \g_@@_renderer_arities_prop
1915   { inlineHtmlTag }
1916   { 1 }
1917 \ExplSyntaxOff
1918 \def\markdownRendererInputBlockHtmlElement{%
1919   \markdownRendererInputBlockHtmlElementPrototype}%
1920 \ExplSyntaxOn

```

```

1921 \seq_gput_right:Nn
1922   \g_@@_renderers_seq
1923   { inputBlockHtmlElement }
1924 \prop_gput:Nnn
1925   \g_@@_renderer_arities_prop
1926   { inputBlockHtmlElement }
1927   { 1 }
1928 \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.

```

1929 \def\markdownRendererImage{%
1930   \markdownRendererImagePrototype}%
1931 \ExplSyntaxOn
1932 \seq_gput_right:Nn
1933   \g_@@_renderers_seq
1934   { image }
1935 \prop_gput:Nnn
1936   \g_@@_renderer_arities_prop
1937   { image }
1938   { 4 }
1939 \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.

```

1940 \def\markdownRendererImageAttributeContextBegin{%
1941   \markdownRendererImageAttributeContextBeginPrototype}%
1942 \ExplSyntaxOn
1943 \seq_gput_right:Nn
1944   \g_@@_renderers_seq
1945   { imageAttributeContextBegin }
1946 \prop_gput:Nnn
1947   \g_@@_renderer_arities_prop
1948   { imageAttributeContextBegin }
1949   { 0 }
1950 \ExplSyntaxOff
1951 \def\markdownRendererImageAttributeContextEnd{%
1952   \markdownRendererImageAttributeContextEndPrototype}%

```

```

1953 \ExplSyntaxOn
1954 \seq_gput_right:Nn
1955   \g_@@_renderers_seq
1956   { imageAttributeContextEnd }
1957 \prop_gput:Nnn
1958   \g_@@_renderer_arities_prop
1959   { imageAttributeContextEnd }
1960   { 0 }
1961 \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.

```

1962 \def\markdownRendererInterblockSeparator{%
1963   \markdownRendererInterblockSeparatorPrototype}%
1964 \ExplSyntaxOn
1965 \seq_gput_right:Nn
1966   \g_@@_renderers_seq
1967   { interblockSeparator }
1968 \prop_gput:Nnn
1969   \g_@@_renderer_arities_prop
1970   { interblockSeparator }
1971   { 0 }
1972 \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.

```

1973 \def\markdownRendererParagraphSeparator{%
1974   \markdownRendererParagraphSeparatorPrototype}%
1975 \ExplSyntaxOn
1976 \seq_gput_right:Nn
1977   \g_@@_renderers_seq
1978   { paragraphSeparator }
1979 \prop_gput:Nnn
1980   \g_@@_renderer_arities_prop
1981   { paragraphSeparator }
1982   { 0 }
1983 \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.

```
1984 \def\markdownRendererLineBlockBegin{%
1985   \markdownRendererLineBlockBeginPrototype}%
1986 \ExplSyntaxOn
1987 \seq_gput_right:Nn
1988   \g_@@_renderers_seq
1989   { lineBlockBegin }
1990 \prop_gput:Nnn
1991   \g_@@_renderer_arities_prop
1992   { lineBlockBegin }
1993   { 0 }
1994 \ExplSyntaxOff
1995 \def\markdownRendererLineBlockEnd{%
1996   \markdownRendererLineBlockEndPrototype}%
1997 \ExplSyntaxOn
1998 \seq_gput_right:Nn
1999   \g_@@_renderers_seq
2000   { lineBlockEnd }
2001 \prop_gput:Nnn
2002   \g_@@_renderer_arities_prop
2003   { lineBlockEnd }
2004   { 0 }
2005 \ExplSyntaxOff
```

### 2.2.5.23 Line Break Renderers

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

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

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

```
2017 \def\markdownRendererHardLineBreak{%
2018   \markdownRendererHardLineBreakPrototype}%
```

```

2019 \ExplSyntaxOn
2020 \seq_gput_right:Nn
2021   \g_@@_renderers_seq
2022   { hardLineBreak }
2023 \prop_gput:Nnn
2024   \g_@@_renderer_arities_prop
2025   { hardLineBreak }
2026   { 0 }
2027 \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.

```

2028 \def\markdownRendererLink{%
2029   \markdownRendererLinkPrototype}%
2030 \ExplSyntaxOn
2031 \seq_gput_right:Nn
2032   \g_@@_renderers_seq
2033   { link }
2034 \prop_gput:Nnn
2035   \g_@@_renderer_arities_prop
2036   { link }
2037   { 4 }
2038 \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.

```

2039 \def\markdownRendererLinkAttributeContextBegin{%
2040   \markdownRendererLinkAttributeContextBeginPrototype}%
2041 \ExplSyntaxOn
2042 \seq_gput_right:Nn
2043   \g_@@_renderers_seq
2044   { linkAttributeContextBegin }
2045 \prop_gput:Nnn
2046   \g_@@_renderer_arities_prop
2047   { linkAttributeContextBegin }
2048   { 0 }
2049 \ExplSyntaxOff
2050 \def\markdownRendererLinkAttributeContextEnd{%

```

```

2051 \markdownRendererLinkAttributeContextEndPrototype}%
2052 \ExplSyntaxOn
2053 \seq_gput_right:Nn
2054 \g_@@_renderers_seq
2055 { linkAttributeContextEnd }
2056 \prop_gput:Nnn
2057 \g_@@_renderer_arities_prop
2058 { linkAttributeContextEnd }
2059 { 0 }
2060 \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.

```

2061 \def\markdownRendererMark{%
2062 \markdownRendererMarkPrototype}%
2063 \ExplSyntaxOn
2064 \seq_gput_right:Nn
2065 \g_@@_renderers_seq
2066 { mark }
2067 \prop_gput:Nnn
2068 \g_@@_renderer_arities_prop
2069 { mark }
2070 { 1 }
2071 \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.

```

2072 \def\markdownRendererDocumentBegin{%
2073 \markdownRendererDocumentBeginPrototype}%
2074 \ExplSyntaxOn
2075 \seq_gput_right:Nn
2076 \g_@@_renderers_seq
2077 { documentBegin }
2078 \prop_gput:Nnn
2079 \g_@@_renderer_arities_prop
2080 { documentBegin }

```

```

2081 { 0 }
2082 \ExplSyntaxOff
2083 \def\markdownRendererDocumentEnd{%
2084 \markdownRendererDocumentEndPrototype}%
2085 \ExplSyntaxOn
2086 \seq_gput_right:Nn
2087 \g_@@_renderers_seq
2088 { documentEnd }
2089 \prop_gput:Nnn
2090 \g_@@_renderer_arities_prop
2091 { documentEnd }
2092 { 0 }
2093 \ExplSyntaxOff

```

### 2.2.5.28 Non-Breaking Space Renderer

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

```

2094 \def\markdownRendererNbsp{%
2095 \markdownRendererNbspPrototype}%
2096 \ExplSyntaxOn
2097 \seq_gput_right:Nn
2098 \g_@@_renderers_seq
2099 { nbsp }
2100 \prop_gput:Nnn
2101 \g_@@_renderer_arities_prop
2102 { nbsp }
2103 { 0 }
2104 \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.

```

2105 \def\markdownRendererNote{%
2106 \markdownRendererNotePrototype}%
2107 \ExplSyntaxOn
2108 \seq_gput_right:Nn
2109 \g_@@_renderers_seq
2110 { note }
2111 \prop_gput:Nnn
2112 \g_@@_renderer_arities_prop
2113 { note }
2114 { 1 }
2115 \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.

```
2116 \def\markdownRendererOlBegin{%
2117   \markdownRendererOlBeginPrototype}%
2118 \ExplSyntaxOn
2119 \seq_gput_right:Nn
2120   \g_@@_renderers_seq
2121   { olBegin }
2122 \prop_gput:Nnn
2123   \g_@@_renderer_arities_prop
2124   { olBegin }
2125   { 0 }
2126 \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.

```
2127 \def\markdownRendererOlBeginTight{%
2128   \markdownRendererOlBeginTightPrototype}%
2129 \ExplSyntaxOn
2130 \seq_gput_right:Nn
2131   \g_@@_renderers_seq
2132   { olBeginTight }
2133 \prop_gput:Nnn
2134   \g_@@_renderer_arities_prop
2135   { olBeginTight }
2136   { 0 }
2137 \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`).

```
2138 \def\markdownRendererFancyOlBegin{%
2139   \markdownRendererFancyOlBeginPrototype}%
2140 \ExplSyntaxOn
2141 \seq_gput_right:Nn
2142   \g_@@_renderers_seq
2143   { fancyOlBegin }
```



```

2144 \prop_gput:Nnn
2145   \g_@@_renderer_arities_prop
2146   { fancyO1Begin }
2147   { 2 }
2148 \ExplSyntaxOff

```

The `\markdownRendererFancyO1BeginTight` 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 `\markdownRendererFancyO1Begin` macro for the valid style values.

```

2149 \def\markdownRendererFancyO1BeginTight{%
2150   \markdownRendererFancyO1BeginTightPrototype}%
2151 \ExplSyntaxOn
2152 \seq_gput_right:Nn
2153   \g_@@_renderers_seq
2154   { fancyO1BeginTight }
2155 \prop_gput:Nnn
2156   \g_@@_renderer_arities_prop
2157   { fancyO1BeginTight }
2158   { 2 }
2159 \ExplSyntaxOff

```

The `\markdownRendererO1Item` 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.

```

2160 \def\markdownRendererO1Item{%
2161   \markdownRendererO1ItemPrototype}%
2162 \ExplSyntaxOn
2163 \seq_gput_right:Nn
2164   \g_@@_renderers_seq
2165   { olItem }
2166 \prop_gput:Nnn
2167   \g_@@_renderer_arities_prop
2168   { olItem }
2169   { 0 }
2170 \ExplSyntaxOff

```

The `\markdownRendererO1ItemEnd` 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.

```

2171 \def\markdownRendererO1ItemEnd{%
2172   \markdownRendererO1ItemEndPrototype}%
2173 \ExplSyntaxOn
2174 \seq_gput_right:Nn

```

```

2175 \g_@@_renderers_seq
2176 { olItemEnd }
2177 \prop_gput:Nnn
2178 \g_@@_renderer_arities_prop
2179 { olItemEnd }
2180 { 0 }
2181 \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.

```

2182 \def\markdownRendererOlItemWithNumber{%
2183 \markdownRendererOlItemWithNumberPrototype}%
2184 \ExplSyntaxOn
2185 \seq_gput_right:Nn
2186 \g_@@_renderers_seq
2187 { olItemWithNumber }
2188 \prop_gput:Nnn
2189 \g_@@_renderer_arities_prop
2190 { olItemWithNumber }
2191 { 1 }
2192 \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.

```

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

```

The `\markdownRendererFancyOlItemEnd` 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.

```

2204 \def\markdownRendererFancyOlItemEnd{%
2205 \markdownRendererFancyOlItemEndPrototype}%
2206 \ExplSyntaxOn
2207 \seq_gput_right:Nn

```

```

2208 \g_@@_renderers_seq
2209 { fancyO1ItemEnd }
2210 \prop_gput:Nnn
2211 \g_@@_renderer_arities_prop
2212 { fancyO1ItemEnd }
2213 { 0 }
2214 \ExplSyntaxOff

```

The `\markdownRendererFancyO1ItemWithNumber` macro represents an item in a fancy ordered list. This macro will only be produced, when the `startNumber` and `fancyLists` options are enabled. The macro receives a single numeric argument that corresponds to the item number.

```

2215 \def\markdownRendererFancyO1ItemWithNumber{%
2216 \markdownRendererFancyO1ItemWithNumberPrototype}%
2217 \ExplSyntaxOn
2218 \seq_gput_right:Nn
2219 \g_@@_renderers_seq
2220 { fancyO1ItemWithNumber }
2221 \prop_gput:Nnn
2222 \g_@@_renderer_arities_prop
2223 { fancyO1ItemWithNumber }
2224 { 1 }
2225 \ExplSyntaxOff

```

The `\markdownRendererO1End` macro represents the end of an ordered list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the `fancyLists` option is disabled. The macro receives no arguments.

```

2226 \def\markdownRendererO1End{%
2227 \markdownRendererO1EndPrototype}%
2228 \ExplSyntaxOn
2229 \seq_gput_right:Nn
2230 \g_@@_renderers_seq
2231 { olEnd }
2232 \prop_gput:Nnn
2233 \g_@@_renderer_arities_prop
2234 { olEnd }
2235 { 0 }
2236 \ExplSyntaxOff

```

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

```

2237 \def\markdownRendererO1EndTight{%
2238 \markdownRendererO1EndTightPrototype}%

```

```

2239 \ExplSyntaxOn
2240 \seq_gput_right:Nn
2241   \g_@@_renderers_seq
2242   { olEndTight }
2243 \prop_gput:Nnn
2244   \g_@@_renderer_arities_prop
2245   { olEndTight }
2246   { 0 }
2247 \ExplSyntaxOff

```

The `\markdownRendererFancyO1End` macro represents the end of a fancy ordered list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the `fancyLists` option is enabled. The macro receives no arguments.

```

2248 \def\markdownRendererFancyO1End{%
2249   \markdownRendererFancyO1EndPrototype}%
2250 \ExplSyntaxOn
2251 \seq_gput_right:Nn
2252   \g_@@_renderers_seq
2253   { fancyO1End }
2254 \prop_gput:Nnn
2255   \g_@@_renderer_arities_prop
2256   { fancyO1End }
2257   { 0 }
2258 \ExplSyntaxOff

```

The `\markdownRendererFancyO1EndTight` macro represents the end of a fancy ordered list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `fancyLists` and `tightLists` options are enabled. The macro receives no arguments.

```

2259 \def\markdownRendererFancyO1EndTight{%
2260   \markdownRendererFancyO1EndTightPrototype}%
2261 \ExplSyntaxOn
2262 \seq_gput_right:Nn
2263   \g_@@_renderers_seq
2264   { fancyO1EndTight }
2265 \prop_gput:Nnn
2266   \g_@@_renderer_arities_prop
2267   { fancyO1EndTight }
2268   { 0 }
2269 \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.

```

2270 \def\markdownRendererInputRawInline{%
2271   \markdownRendererInputRawInlinePrototype}%
2272 \ExplSyntaxOn
2273 \seq_gput_right:Nn
2274   \g_@@_renderers_seq
2275   { inputRawInline }
2276 \prop_gput:Nnn
2277   \g_@@_renderer_arities_prop
2278   { inputRawInline }
2279   { 2 }
2280 \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.

```

2281 \def\markdownRendererInputRawBlock{%
2282   \markdownRendererInputRawBlockPrototype}%
2283 \ExplSyntaxOn
2284 \seq_gput_right:Nn
2285   \g_@@_renderers_seq
2286   { inputRawBlock }
2287 \prop_gput:Nnn
2288   \g_@@_renderer_arities_prop
2289   { inputRawBlock }
2290   { 2 }
2291 \ExplSyntaxOff

```

### 2.2.5.32 Section Renderers

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

```

2292 \def\markdownRendererSectionBegin{%
2293   \markdownRendererSectionBeginPrototype}%
2294 \ExplSyntaxOn
2295 \seq_gput_right:Nn
2296   \g_@@_renderers_seq
2297   { sectionBegin }
2298 \prop_gput:Nnn
2299   \g_@@_renderer_arities_prop
2300   { sectionBegin }
2301   { 0 }
2302 \ExplSyntaxOff
2303 \def\markdownRendererSectionEnd{%

```

```

2304 \markdownRendererSectionEndPrototype}%
2305 \ExplSyntaxOn
2306 \seq_gput_right:Nn
2307 \g_@@_renderers_seq
2308 { sectionEnd }
2309 \prop_gput:Nnn
2310 \g_@@_renderer_arities_prop
2311 { sectionEnd }
2312 { 0 }
2313 \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.

```

2314 \def\markdownRendererReplacementCharacter{%
2315 \markdownRendererReplacementCharacterPrototype}%
2316 \ExplSyntaxOn
2317 \seq_gput_right:Nn
2318 \g_@@_renderers_seq
2319 { replacementCharacter }
2320 \prop_gput:Nnn
2321 \g_@@_renderer_arities_prop
2322 { replacementCharacter }
2323 { 0 }
2324 \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`.

```

2325 \def\markdownRendererLeftBrace{%
2326 \markdownRendererLeftBracePrototype}%
2327 \ExplSyntaxOn
2328 \seq_gput_right:Nn
2329 \g_@@_renderers_seq
2330 { leftBrace }
2331 \prop_gput:Nnn
2332 \g_@@_renderer_arities_prop
2333 { leftBrace }
2334 { 0 }
2335 \ExplSyntaxOff
2336 \def\markdownRendererRightBrace{%
2337 \markdownRendererRightBracePrototype}%
2338 \ExplSyntaxOn
2339 \seq_gput_right:Nn

```

```

2340 \g_@@_renderers_seq
2341 { rightBrace }
2342 \prop_gput:Nnn
2343 \g_@@_renderer_arities_prop
2344 { rightBrace }
2345 { 0 }
2346 \ExplSyntaxOff
2347 \def\markdownRendererDollarSign{%
2348 \markdownRendererDollarSignPrototype}%
2349 \ExplSyntaxOn
2350 \seq_gput_right:Nn
2351 \g_@@_renderers_seq
2352 { dollarSign }
2353 \prop_gput:Nnn
2354 \g_@@_renderer_arities_prop
2355 { dollarSign }
2356 { 0 }
2357 \ExplSyntaxOff
2358 \def\markdownRendererPercentSign{%
2359 \markdownRendererPercentSignPrototype}%
2360 \ExplSyntaxOn
2361 \seq_gput_right:Nn
2362 \g_@@_renderers_seq
2363 { percentSign }
2364 \prop_gput:Nnn
2365 \g_@@_renderer_arities_prop
2366 { percentSign }
2367 { 0 }
2368 \ExplSyntaxOff
2369 \def\markdownRendererAmpersand{%
2370 \markdownRendererAmpersandPrototype}%
2371 \ExplSyntaxOn
2372 \seq_gput_right:Nn
2373 \g_@@_renderers_seq
2374 { ampersand }
2375 \prop_gput:Nnn
2376 \g_@@_renderer_arities_prop
2377 { ampersand }
2378 { 0 }
2379 \ExplSyntaxOff
2380 \def\markdownRendererUnderscore{%
2381 \markdownRendererUnderscorePrototype}%
2382 \ExplSyntaxOn
2383 \seq_gput_right:Nn
2384 \g_@@_renderers_seq
2385 { underscore }
2386 \prop_gput:Nnn

```

```

2387 \g_@@_renderer_arities_prop
2388 { underscore }
2389 { 0 }
2390 \ExplSyntaxOff
2391 \def\markdownRendererHash{%
2392 \markdownRendererHashPrototype}%
2393 \ExplSyntaxOn
2394 \seq_gput_right:Nn
2395 \g_@@_renderers_seq
2396 { hash }
2397 \prop_gput:Nnn
2398 \g_@@_renderer_arities_prop
2399 { hash }
2400 { 0 }
2401 \ExplSyntaxOff
2402 \def\markdownRendererCircumflex{%
2403 \markdownRendererCircumflexPrototype}%
2404 \ExplSyntaxOn
2405 \seq_gput_right:Nn
2406 \g_@@_renderers_seq
2407 { circumflex }
2408 \prop_gput:Nnn
2409 \g_@@_renderer_arities_prop
2410 { circumflex }
2411 { 0 }
2412 \ExplSyntaxOff
2413 \def\markdownRendererBackslash{%
2414 \markdownRendererBackslashPrototype}%
2415 \ExplSyntaxOn
2416 \seq_gput_right:Nn
2417 \g_@@_renderers_seq
2418 { backslash }
2419 \prop_gput:Nnn
2420 \g_@@_renderer_arities_prop
2421 { backslash }
2422 { 0 }
2423 \ExplSyntaxOff
2424 \def\markdownRendererTilde{%
2425 \markdownRendererTildePrototype}%
2426 \ExplSyntaxOn
2427 \seq_gput_right:Nn
2428 \g_@@_renderers_seq
2429 { tilde }
2430 \prop_gput:Nnn
2431 \g_@@_renderer_arities_prop
2432 { tilde }
2433 { 0 }

```



```

2434 \ExplSyntaxOff
2435 \def\markdownRendererPipe{%
2436   \markdownRendererPipePrototype}%
2437 \ExplSyntaxOn
2438 \seq_gput_right:Nn
2439   \g_@@_renderers_seq
2440   { pipe }
2441 \prop_gput:Nnn
2442   \g_@@_renderer_arities_prop
2443   { pipe }
2444   { 0 }
2445 \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.

```

2446 \def\markdownRendererStrikeThrough{%
2447   \markdownRendererStrikeThroughPrototype}%
2448 \ExplSyntaxOn
2449 \seq_gput_right:Nn
2450   \g_@@_renderers_seq
2451   { strikeThrough }
2452 \prop_gput:Nnn
2453   \g_@@_renderer_arities_prop
2454   { strikeThrough }
2455   { 1 }
2456 \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.

```

2457 \def\markdownRendererSubscript{%
2458   \markdownRendererSubscriptPrototype}%
2459 \ExplSyntaxOn
2460 \seq_gput_right:Nn
2461   \g_@@_renderers_seq
2462   { subscript }
2463 \prop_gput:Nnn
2464   \g_@@_renderer_arities_prop
2465   { subscript }
2466   { 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.

```
2467 \def\markdownRendererSuperscript{%
2468   \markdownRendererSuperscriptPrototype}%
2469 \ExplSyntaxOn
2470 \seq_gput_right:Nn
2471   \g_@@_renderers_seq
2472   { superscript }
2473 \prop_gput:Nnn
2474   \g_@@_renderer_arities_prop
2475   { superscript }
2476   { 1 }
2477 \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.

```
2478 \def\markdownRendererTableAttributeContextBegin{%
2479   \markdownRendererTableAttributeContextBeginPrototype}%
2480 \ExplSyntaxOn
2481 \seq_gput_right:Nn
2482   \g_@@_renderers_seq
2483   { tableAttributeContextBegin }
2484 \prop_gput:Nnn
2485   \g_@@_renderer_arities_prop
2486   { tableAttributeContextBegin }
2487   { 0 }
2488 \ExplSyntaxOff
2489 \def\markdownRendererTableAttributeContextEnd{%
2490   \markdownRendererTableAttributeContextEndPrototype}%
2491 \ExplSyntaxOn
2492 \seq_gput_right:Nn
2493   \g_@@_renderers_seq
2494   { tableAttributeContextEnd }
2495 \prop_gput:Nnn
2496   \g_@@_renderer_arities_prop
2497   { tableAttributeContextEnd }
2498   { 0 }
2499 \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.

```
2500 \def\markdownRendererTable{%
2501   \markdownRendererTablePrototype}%
2502 \ExplSyntaxOn
2503 \seq_gput_right:Nn
2504   \g_@@_renderers_seq
2505   { table }
2506 \prop_gput:Nnn
2507   \g_@@_renderer_arities_prop
2508   { table }
2509   { 3 }
2510 \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.

```
2511 \def\markdownRendererInlineMath{%
2512   \markdownRendererInlineMathPrototype}%
2513 \ExplSyntaxOn
2514 \seq_gput_right:Nn
2515   \g_@@_renderers_seq
2516   { inlineMath }
2517 \prop_gput:Nnn
2518   \g_@@_renderer_arities_prop
2519   { inlineMath }
2520   { 1 }
2521 \ExplSyntaxOff
2522 \def\markdownRendererDisplayMath{%
2523   \markdownRendererDisplayMathPrototype}%
```

```

2524 \ExplSyntaxOn
2525 \seq_gput_right:Nn
2526   \g_@@_renderers_seq
2527   { displayMath }
2528 \prop_gput:Nnn
2529   \g_@@_renderer_arities_prop
2530   { displayMath }
2531   { 1 }
2532 \ExplSyntaxOff

```

#### 2.2.5.41 Thematic Break Renderer

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

```

2533 \def\markdownRendererThematicBreak{%
2534   \markdownRendererThematicBreakPrototype}%
2535 \ExplSyntaxOn
2536 \seq_gput_right:Nn
2537   \g_@@_renderers_seq
2538   { thematicBreak }
2539 \prop_gput:Nnn
2540   \g_@@_renderer_arities_prop
2541   { thematicBreak }
2542   { 0 }
2543 \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.

```

2544 \def\markdownRendererTickedBox{%
2545   \markdownRendererTickedBoxPrototype}%
2546 \ExplSyntaxOn
2547 \seq_gput_right:Nn
2548   \g_@@_renderers_seq
2549   { tickedBox }
2550 \prop_gput:Nnn
2551   \g_@@_renderer_arities_prop
2552   { tickedBox }
2553   { 0 }
2554 \ExplSyntaxOff
2555 \def\markdownRendererHalfTickedBox{%
2556   \markdownRendererHalfTickedBoxPrototype}%

```

```

2557 \ExplSyntaxOn
2558 \seq_gput_right:Nn
2559   \g_@@_renderers_seq
2560   { halfTickedBox }
2561 \prop_gput:Nnn
2562   \g_@@_renderer_arities_prop
2563   { halfTickedBox }
2564   { 0 }
2565 \ExplSyntaxOff
2566 \def\markdownRendererUntickedBox{%
2567   \markdownRendererUntickedBoxPrototype}%
2568 \ExplSyntaxOn
2569 \seq_gput_right:Nn
2570   \g_@@_renderers_seq
2571   { untickedBox }
2572 \prop_gput:Nnn
2573   \g_@@_renderer_arities_prop
2574   { untickedBox }
2575   { 0 }
2576 \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.

```

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

```

2588 \def\markdownRendererJekyllDataEnd{%
2589   \markdownRendererJekyllDataEndPrototype}%
2590 \ExplSyntaxOn
2591 \seq_gput_right:Nn
2592   \g_@@_renderers_seq
2593   { jekyllDataEnd }

```

```

2594 \prop_gput:Nnn
2595   \g_@@_renderer_arities_prop
2596   { jekyllDataEnd }
2597   { 0 }
2598 \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.

```

2599 \def\markdownRendererJekyllDataMappingBegin{%
2600   \markdownRendererJekyllDataMappingBeginPrototype}%
2601 \ExplSyntaxOn
2602 \seq_gput_right:Nn
2603   \g_@@_renderers_seq
2604   { jekyllDataMappingBegin }
2605 \prop_gput:Nnn
2606   \g_@@_renderer_arities_prop
2607   { jekyllDataMappingBegin }
2608   { 2 }
2609 \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.

```

2610 \def\markdownRendererJekyllDataMappingEnd{%
2611   \markdownRendererJekyllDataMappingEndPrototype}%
2612 \ExplSyntaxOn
2613 \seq_gput_right:Nn
2614   \g_@@_renderers_seq
2615   { jekyllDataMappingEnd }
2616 \prop_gput:Nnn
2617   \g_@@_renderer_arities_prop
2618   { jekyllDataMappingEnd }
2619   { 0 }
2620 \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.

```

2621 \def\markdownRendererJekyllDataSequenceBegin{%
2622   \markdownRendererJekyllDataSequenceBeginPrototype}%
2623 \ExplSyntaxOn

```

```

2624 \seq_gput_right:Nn
2625   \g_@@_renderers_seq
2626   { jekyllDataSequenceBegin }
2627 \prop_gput:Nnn
2628   \g_@@_renderer_arities_prop
2629   { jekyllDataSequenceBegin }
2630   { 2 }
2631 \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.

```

2632 \def\markdownRendererJekyllDataSequenceEnd{%
2633   \markdownRendererJekyllDataSequenceEndPrototype}%
2634 \ExplSyntaxOn
2635 \seq_gput_right:Nn
2636   \g_@@_renderers_seq
2637   { jekyllDataSequenceEnd }
2638 \prop_gput:Nnn
2639   \g_@@_renderer_arities_prop
2640   { jekyllDataSequenceEnd }
2641   { 0 }
2642 \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.

```

2643 \def\markdownRendererJekyllDataBoolean{%
2644   \markdownRendererJekyllDataBooleanPrototype}%
2645 \ExplSyntaxOn
2646 \seq_gput_right:Nn
2647   \g_@@_renderers_seq
2648   { jekyllDataBoolean }
2649 \prop_gput:Nnn
2650   \g_@@_renderer_arities_prop
2651   { jekyllDataBoolean }
2652   { 2 }
2653 \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.

```

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

```

2665 \def\markdownRendererJekyllDataString{%
2666   \markdownRendererJekyllDataStringPrototype}%
2667 \ExplSyntaxOn
2668 \seq_gput_right:Nn
2669   \g_@@_renderers_seq
2670   { jekyllDataString }
2671 \prop_gput:Nnn
2672   \g_@@_renderer_arities_prop
2673   { jekyllDataString }
2674   { 2 }
2675 \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.

```

2676 \def\markdownRendererJekyllDataEmpty{%
2677   \markdownRendererJekyllDataEmptyPrototype}%
2678 \ExplSyntaxOn
2679 \seq_gput_right:Nn
2680   \g_@@_renderers_seq
2681   { jekyllDataEmpty }
2682 \prop_gput:Nnn
2683   \g_@@_renderer_arities_prop
2684   { jekyllDataEmpty }
2685   { 1 }
2686 \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.

```
2687 \ExplSyntaxOn
2688 \cs_new:Nn \@@_define_renderers:
2689   {
2690     \seq_map_function:NN
2691       \g_@@_renderers_seq
2692       \@@_define_renderer:n
2693   }
2694 \cs_new:Nn \@@_define_renderer:n
2695   {
2696     \@@_renderer_tl_to_csname:nN
2697       { #1 }
2698     \l_tmpa_tl
2699     \prop_get:NnN
2700       \g_@@_renderer_arities_prop
2701       { #1 }
2702     \l_tmpb_tl
2703     \@@_define_renderer:ncV
2704       { #1 }
2705     { \l_tmpa_tl }
2706     \l_tmpb_tl
2707   }
2708 \cs_new:Nn \@@_renderer_tl_to_csname:nN
2709   {
2710     \tl_set:Nn
2711       \l_tmpa_tl
2712       { \str_uppercase:n { #1 } }
2713     \tl_set:Nx
2714       #2
2715       {
2716         markdownRenderer
2717         \tl_head:f { \l_tmpa_tl }
2718         \tl_tail:n { #1 }
2719       }
2720   }
2721 \tl_new:N
2722   \l_@@_renderer_definition_tl
2723 \bool_new:N
2724   \g_@@_appending_renderer_bool
2725 \cs_new:Nn \@@_define_renderer:nNn
2726   {
```

```

2727 \keys_define:nn
2728   { markdown/options/renderers }
2729   {
2730     #1 .code:n = {
2731       \tl_set:Nn
2732         \l_@@_renderer_definition_tl
2733         { ##1 }
2734       \regex_replace_all:nnN
2735         { \cP\#0 }
2736         { #1 }
2737       \l_@@_renderer_definition_tl
2738       \bool_if:NT
2739         \g_@@_appending_renderer_bool
2740         {
2741           \@@_tl_set_from_cs:NNn
2742           \l_tmpa_tl
2743           #2
2744           { #3 }
2745           \tl_put_left:NV
2746             \l_@@_renderer_definition_tl
2747             \l_tmpa_tl
2748         }
2749       \cs_generate_from_arg_count:NNnV
2750         #2
2751         \cs_set:Npn
2752         { #3 }
2753         \l_@@_renderer_definition_tl
2754     },
2755   }
2756 }

```

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

```

2757 \cs_new_protected:Nn
2758   \@@_tl_set_from_cs:NNn
2759   {
2760     \tl_set:Nn
2761       \l_tmpa_tl
2762       { #2 }
2763     \int_step_inline:nn
2764       { #3 }
2765     {
2766       \exp_args:Nnc
2767         \tl_put_right:Nn
2768         \l_tmpa_tl

```

```

2769         { @@_tl_set_from_cs_parameter_ ##1 }
2770     }
2771     \exp_args:NNV
2772     \tl_set:No
2773     \l_tmpb_tl
2774     \l_tmpa_tl
2775     \regex_replace_all:nnN
2776     { \cP. }
2777     { \0\0 }
2778     \l_tmpb_tl
2779     \int_step_inline:nn
2780     { #3 }
2781     {
2782         \regex_replace_all:nnN
2783         { \c { @@_tl_set_from_cs_parameter_ ##1 } }
2784         { \cP\# ##1 }
2785         \l_tmpb_tl
2786     }
2787     \tl_set:NV
2788     #1
2789     \l_tmpb_tl
2790 }
2791 \cs_generate_variant:Nn
2792 \@@_define_renderer:nNn
2793 { ncV }
2794 \cs_generate_variant:Nn
2795 \cs_generate_from_arg_count:NNnn
2796 { NNnV }
2797 \cs_generate_variant:Nn
2798 \tl_put_left:Nn
2799 { Nv }
2800 \keys_define:nn
2801 { markdown/options }
2802 {
2803     renderers .code:n = {
2804         \keys_set:nn
2805         { markdown/options/renderers }
2806         { #1 }
2807     },
2808 }

```

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.
  }
}

```

```

2809 \tl_new:N
2810   \l_@@_renderer_glob_definition_tl
2811 \seq_new:N
2812   \l_@@_renderer_glob_results_seq
2813 \regex_const:Nn
2814   \c_@@_appending_key_regex
2815   { \s*+$ }
2816 \keys_define:nn
2817   { markdown/options/renderers }
2818   {
2819     unknown .code:n = {

```

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

```

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

```

```

2820     \regex_match:NVTF
2821     \c_@@_appending_key_regex
2822     \l_keys_key_str
2823     {
2824         \bool_gset_true:N
2825         \g_@@_appending_renderer_bool
2826         \tl_set:NV
2827         \l_tmpa_tl
2828         \l_keys_key_str
2829         \regex_replace_once:NnN
2830         \c_@@_appending_key_regex
2831         { }
2832         \l_tmpa_tl
2833         \tl_set:Nx
2834         \l_tmpb_tl
2835         { { \l_tmpa_tl } = }
2836         \tl_put_right:Nn
2837         \l_tmpb_tl
2838         { { #1 } }
2839         \keys_set:nV
2840         { markdown/options/renderers }
2841         \l_tmpb_tl
2842         \bool_gset_false:N
2843         \g_@@_appending_renderer_bool
2844     }

```

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

```

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

```

Wildcards and enumerations can be combined:

```

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

```

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

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

```
2845     {
2846         \@@_glob_seq:VnN
2847         \l_keys_key_str
2848         { g_@@_renderers_seq }
2849         \l_@@_renderer_glob_results_seq
2850     \seq_if_empty:NTF
2851     \l_@@_renderer_glob_results_seq
2852     {
2853         \msg_error:nnV
2854         { markdown }
2855         { undefined-renderer }
2856         \l_keys_key_str
2857     }
2858     {
2859         \tl_set:Nn
2860         \l_@@_renderer_glob_definition_tl
2861         { \exp_not:n { #1 } }
2862     \seq_map_inline:Nn
2863     \l_@@_renderer_glob_results_seq
2864     {
2865         \tl_set:Nn
2866         \l_tmpa_tl
2867         { { ##1 } = }
2868         \tl_put_right:Nx
2869         \l_tmpa_tl
2870         { { \l_@@_renderer_glob_definition_tl } }
2871     \keys_set:nV
2872     { markdown/options/renderers }
2873     \l_tmpa_tl
2874     }
2875     }
2876 }
2877 },
2878 }
2879 \msg_new:nnn
2880 { markdown }
2881 { undefined-renderer }
2882 {
```

```

2883     Renderer~#1~is~undefined.
2884 }
2885 \cs_generate_variant:Nn
2886   \@@_glob_seq:nnN
2887   { VnN }
2888 \cs_generate_variant:Nn
2889   \cs_generate_from_arg_count:NNnn
2890   { cNVV }
2891 \cs_generate_variant:Nn
2892   \msg_error:nnn
2893   { nnV }
2894 \prg_generate_conditional_variant:Nnn
2895   \regex_match:Nn
2896   { NV }
2897   { TF }
2898 \prop_new:N
2899   \g_@@_glob_cache_prop
2900 \tl_new:N
2901   \l_@@_current_glob_tl
2902 \cs_new:Nn
2903   \@@_glob_seq:nnN
2904   {
2905     \tl_set:Nn
2906       \l_@@_current_glob_tl
2907       { ~ #1 $ }
2908     \prop_get:NeNTF
2909       \g_@@_glob_cache_prop
2910       { #2 / \l_@@_current_glob_tl }
2911     \l_tmpa_clist
2912     {
2913       \seq_set_from_clist:NN
2914         #3
2915         \l_tmpa_clist
2916     }
2917     {
2918       \seq_clear:N
2919         #3
2920       \regex_replace_all:nnN
2921         { \* }
2922         { .* }
2923         \l_@@_current_glob_tl
2924       \regex_set:NV
2925         \l_tmpa_regex
2926         \l_@@_current_glob_tl
2927       \seq_map_inline:cn
2928         { #2 }
2929         {

```

```

2930         \regex_match:NnT
2931         \l_tmpa_regex
2932         { ##1 }
2933         {
2934             \seq_put_right:Nn
2935             #3
2936             { ##1 }
2937         }
2938     }
2939     \clist_set_from_seq:NN
2940     \l_tmpa_clist
2941     #3
2942     \prop_gput:NeV
2943     \g_@@_glob_cache_prop
2944     { #2 / \l_@@_current_glob_tl }
2945     \l_tmpa_clist
2946 }
2947 }
2948 % TODO: Remove in TeX Live 2023.
2949 \prg_generate_conditional_variant:Nnn
2950 \prop_get:NnN
2951 { NeN }
2952 { TF }
2953 \cs_generate_variant:Nn
2954 \regex_set:Nn
2955 { NV }
2956 \cs_generate_variant:Nn
2957 \prop_gput:Nnn
2958 { NeV }

```

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

```

2959 \str_if_eq:VVT
2960 \c_@@_top_layer_tl
2961 \c_@@_option_layer_plain_tex_tl
2962 {
2963     \@@_define_renderers:
2964 }
2965 \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.

```
2966 \ExplSyntaxOn
2967 \keys_define:nn
2968   { markdown/jekyllData }
2969   { }
2970 \ExplSyntaxOff
```

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

```
2971 \ExplSyntaxOn
2972 \@@_with_various_cases:nn
2973   { jekyllDataRenderers }
2974   {
2975     \keys_define:nn
2976       { markdown/options }
2977       {
2978         #1 .code:n = {
2979           \tl_set:Nn
2980             \l_tmpa_tl
2981             { ##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.

```
2982         \tl_replace_all:NnV
2983         \l_tmpa_tl
2984         { / }
2985         \c_backslash_str
2986         \keys_set:nV
2987         { markdown/options/jekyll-data-renderers }
2988         \l_tmpa_tl
2989     },
2990   }
2991 }
2992 \keys_define:nn
2993   { markdown/options/jekyll-data-renderers }
2994   {
2995     unknown .code:n = {
2996       \tl_set_eq:NN
2997       \l_tmpa_tl
2998       \l_keys_key_str
2999       \tl_replace_all:NVn
3000       \l_tmpa_tl
```

```

3001     \c_backslash_str
3002     { / }
3003     \tl_put_right:Nn
3004     \l_tmpa_tl
3005     {
3006         .code:n = { #1 }
3007     }
3008     \keys_define:nV
3009     { markdown/jekyllData }
3010     \l_tmpa_tl
3011 }
3012 }
3013 \cs_generate_variant:Nn
3014 \keys_define:nn
3015 { nV }
3016 \ExplSyntaxOff

```

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

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

```

3017 \ExplSyntaxOn
3018 \cs_new:Nn \@@_define_renderer_prototypes:
3019 {
3020     \seq_map_function:NN
3021     \g_@@_renderers_seq
3022     \@@_define_renderer_prototype:n
3023 }
3024 \cs_new:Nn \@@_define_renderer_prototype:n
3025 {
3026     \@@_renderer_prototype_tl_to_csname:nN
3027     { #1 }
3028     \l_tmpa_tl
3029     \prop_get:NnN
3030     \g_@@_renderer_arities_prop
3031     { #1 }
3032     \l_tmpb_tl
3033     \@@_define_renderer_prototype:ncV
3034     { #1 }
3035     { \l_tmpa_tl }
3036     \l_tmpb_tl
3037 }
3038 \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN

```

```

3039 {
3040   \tl_set:Nn
3041     \l_tmpa_tl
3042     { \str_uppercase:n { #1 } }
3043   \tl_set:Nx
3044     #2
3045     {
3046       markdownRenderer
3047       \tl_head:f { \l_tmpa_tl }
3048       \tl_tail:n { #1 }
3049       Prototype
3050     }
3051 }
3052 \tl_new:N
3053   \l_@@_renderer_prototype_definition_tl
3054 \bool_new:N
3055   \g_@@_appending_renderer_prototype_bool
3056 \cs_new:Nn \@@_define_renderer_prototype:nNn
3057 {
3058   \keys_define:nn
3059     { markdown/options/renderer-prototypes }
3060     {
3061       #1 .code:n = {
3062         \tl_set:Nn
3063           \l_@@_renderer_prototype_definition_tl
3064           { ##1 }
3065         \regex_replace_all:nnN
3066           { \cP\#0 }
3067           { #1 }
3068         \l_@@_renderer_prototype_definition_tl
3069         \bool_if:NT
3070           \g_@@_appending_renderer_prototype_bool
3071           {
3072             \@@_tl_set_from_cs:NNn
3073               \l_tmpa_tl
3074               #2
3075               { #3 }
3076             \tl_put_left:NV
3077               \l_@@_renderer_prototype_definition_tl
3078               \l_tmpa_tl
3079           }
3080         \cs_generate_from_arg_count:NNnV
3081           #2
3082           \cs_set:Npn
3083             { #3 }
3084             \l_@@_renderer_prototype_definition_tl
3085       },

```

```
3086     }
```

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

```
3087     \cs_if_free:NT
3088       #2
3089     {
3090       \cs_generate_from_arg_count:NNnn
3091         #2
3092         \cs_set:Npn
3093           { #3 }
3094           { }
3095     }
3096   }
3097 \cs_generate_variant:Nn
3098   \@@_define_renderer_prototype:nNn
3099   { ncV }
```

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

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

```
3100 \keys_define:nn
3101   { markdown/options/renderer-prototypes }
3102   {
3103     unknown .code:n = {
```

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

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

```

headerAttributeContextBegin += {
  \markdownSetup{
    rendererPrototypes = {
      attributeClassName += {...},
    },
  },
  % Define the processing of a single specific HTML identifier.
headerAttributeContextBegin += {
  \markdownSetup{
    rendererPrototypes = {
      attributeIdentifier += {...},
    },
  },
},
}

```

```

3104   \regex_match:NVTF
3105   \c_@@_appending_key_regex
3106   \l_keys_key_str
3107   {
3108     \bool_gset_true:N
3109     \g_@@_appending_renderer_prototype_bool
3110   \tl_set:NV
3111     \l_tmpa_tl
3112     \l_keys_key_str
3113   \regex_replace_once:NnN
3114     \c_@@_appending_key_regex
3115     { }
3116     \l_tmpa_tl
3117   \tl_set:Nx
3118     \l_tmpb_tl
3119     { { \l_tmpa_tl } = }
3120   \tl_put_right:Nn
3121     \l_tmpb_tl
3122     { { #1 } }
3123   \keys_set:nV
3124     { markdown/options/renderer-prototypes }
3125     \l_tmpb_tl
3126   \bool_gset_false:N
3127     \g_@@_appending_renderer_prototype_bool
3128   }

```

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

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

Wildcards and enumerations can be combined:

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

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

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

```
3129     {
3130         \@_glob_seq:VnN
3131         \l_keys_key_str
3132         { g_@_renderers_seq }
3133         \l_@_renderer_glob_results_seq
3134         \seq_if_empty:NTF
3135         \l_@_renderer_glob_results_seq
3136         {
3137             \msg_error:nnV
3138             { markdown }
3139             { undefined-renderer-prototype }
3140             \l_keys_key_str
3141         }
```

```

3142     {
3143     \tl_set:Nn
3144     \l_@@_renderer_glob_definition_tl
3145     { \exp_not:n { #1 } }
3146     \seq_map_inline:Nn
3147     \l_@@_renderer_glob_results_seq
3148     {
3149     \tl_set:Nn
3150     \l_tmpa_tl
3151     { { ##1 } = }
3152     \tl_put_right:Nx
3153     \l_tmpa_tl
3154     { { \l_@@_renderer_glob_definition_tl } }
3155     \keys_set:nV
3156     { markdown/options/renderer-prototypes }
3157     \l_tmpa_tl
3158     }
3159     }
3160   }
3161 },
3162 }
3163 \msg_new:nnn
3164 { markdown }
3165 { undefined-renderer-prototype }
3166 {
3167   Renderer~prototype~#1~is~undefined.
3168 }
3169 \@@_with_various_cases:nn
3170 { rendererPrototypes }
3171 {
3172   \keys_define:nn
3173   { markdown/options }
3174   {
3175     #1 .code:n = {
3176       \keys_set:nn
3177       { markdown/options/renderer-prototypes }
3178       { ##1 }
3179     },
3180   }
3181 }

```

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

```

3182 \str_if_eq:VVT
3183 \c_@@_top_layer_tl
3184 \c_@@_option_layer_plain_tex_tl

```

```

3185 {
3186   \@@_define_renderer_prototypes:
3187 }
3188 \ExplSyntaxOff

```

### 2.2.7 Logging Facilities

The `\markdownInfo`, `\markdownWarning`, and `\markdownError` macros perform logging for the Markdown package. Their first argument specifies the text of the info, warning, or error message. The `\markdownError` macro receives a second argument that provides a help text. You may redefine these macros to redirect and process the info, warning, and error messages.

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

### 2.2.8 Miscellanea

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

```

3189 \let\markdownMakeOther\relax

```

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

```

3190 \let\markdownReadAndConvert\relax
3191 \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*.

```

3192   \catcode`\|=0\catcode`\=12%
3193   |gdef|markdownBegin{%
3194     |markdownReadAndConvert{\markdownEnd}%
3195                               {|\markdownEnd}}%
3196 |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.

```
3197 \ExplSyntaxOn
3198 \keys_define:nn
3199   { markdown/options }
3200   {
3201     code .code:n = { #1 },
3202   }
3203 \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.

```
3204 \ExplSyntaxOn
3205 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
3206 \cs_generate_variant:Nn
3207   \tl_const:Nn
3208   { NV }
3209 \tl_if_exist:NF
3210   \c_@@_top_layer_tl
3211   {
3212     \tl_const:NV
3213       \c_@@_top_layer_tl
3214       \c_@@_option_layer_latex_tl
3215   }
3216 \ExplSyntaxOff
3217 \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  $\LaTeX 2_{\epsilon}$  parses package options.

### 2.3.1 Typesetting Markdown

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

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

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

```
3218 \newenvironment{markdown}\relax\relax
3219 \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`  $\LaTeX$  environment (and likewise for the starred version).

Note that the `markdown` and `markdown*`  $\LaTeX$  environments are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros exposed by the plain  $\TeX$  interface.

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

<code>\documentclass{article}</code>	<code>\documentclass{article}</code>
<code>\usepackage{markdown}</code>	<code>\usepackage{markdown}</code>
<code>\begin{document}</code>	<code>\begin{document}</code>
<code>% ...</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 `\markinline` macro accepts a single mandatory parameter containing inline markdown content and expands to the result of the conversion of the input markdown document to plain  $\TeX$ . Unlike the `\markinline` macro provided by the plain  $\TeX$  interface, this macro also accepts  $\LaTeX$  interface options (see Section 2.3.2) as its optional argument. These options will only influence this markdown content.

The `\markdownInput` macro accepts a single mandatory parameter containing the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain  $\TeX$ . Unlike the `\markdownInput` macro

provided by the plain  $\text{T}_{\text{E}}\text{X}$  interface, this macro also accepts  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  interface options (see Section 2.3.2) as its optional argument. These options will only influence this markdown document.

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

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

## 2.3.2 Options

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

$\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  options map directly to the options recognized by the plain  $\text{T}_{\text{E}}\text{X}$  interface (see Section 2.2.2) and to the markdown token renderers and their prototypes recognized by the plain  $\text{T}_{\text{E}}\text{X}$  interface (see Sections 2.2.5 and 2.2.6).

The  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  options may be specified when loading the  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  package, when using the `markdown*`  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  environment or the `\markdownInput` macro (see Section 2.3), or via the `\markdownSetup` macro.

### 2.3.2.1 Finalizing and Freezing the Cache

To ensure compatibility with the `minted` package [10, Section 5.1], which supports the `finalizcache` and `frozenscache` package options with similar semantics to the `finalizeCache` and `frozenCache` plain  $\text{T}_{\text{E}}\text{X}$  options, the Markdown package also recognizes these as aliases and accepts them as document class options. By passing `finalizcache` and `frozenscache` as document class options, you may conveniently control the behavior of both packages at once:

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

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

```
3220 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
3221 \DeclareOption{frozenscache}{\markdownSetup{frozenCache}}
```

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

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

```
3222 \ExplSyntaxOn
3223 \str_if_eq:VVT
3224   \c_@@_top_layer_tl
3225   \c_@@_option_layer_latex_tl
3226   {
3227     \@@_define_option_commands_and_keyvals:
3228     \@@_define_renderers:
3229     \@@_define_renderer_prototypes:
3230   }
3231 \ExplSyntaxOff
```

The following example L<sup>A</sup>T<sub>E</sub>X code showcases a possible configuration of plain T<sub>E</sub>X interface options `hybrid`, `smartEllipses`, and `cacheDir`.

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

### 2.3.3 Themes

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

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

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

the theme(s) will be loaded immediately. For example, there is a theme named `witiko/dot`, which typesets fenced code blocks with the `dot` infostring as images of directed graphs rendered by the Graphviz tools. The following code would first load the Markdown package, then the `markdownthemewitiko_beamer_MU.sty` L<sup>A</sup>T<sub>E</sub>X package, and finally the `markdownthemewitiko_dot.sty` L<sup>A</sup>T<sub>E</sub>X package:

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

```
3232 \newif\ifmarkdownLaTeXLoaded
3233 \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";

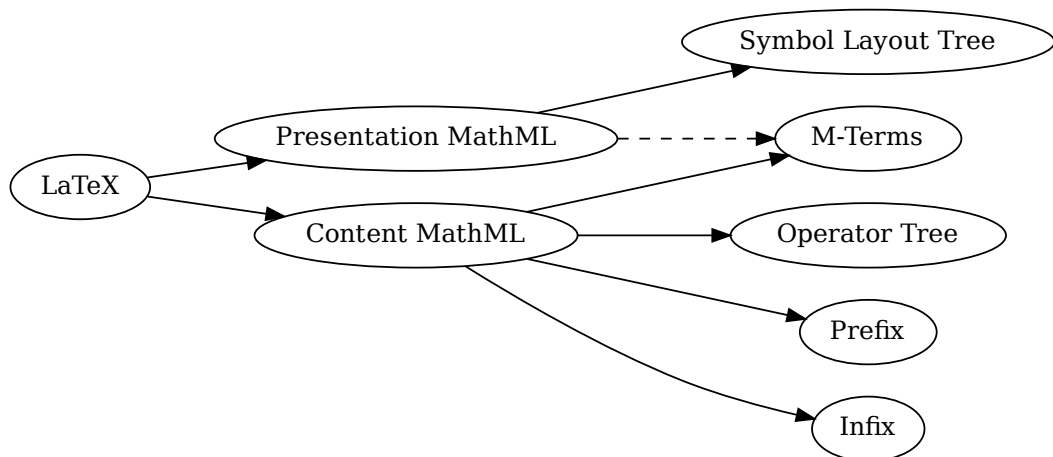
  latex -> pmml;
  latex -> cmml;
  pmml -> slt;
  cmml -> opt;
  cmml -> prefix;
  cmml -> infix;
  pmml -> mterms [style=dashed];
  cmml -> mterms;
```

```

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  $\TeX$  option is enabled.

3234 \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

```

\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 requires the catchfile `LATEX` package and a Unix-like operating system with GNU Coreutils `md5sum` and either GNU `Wget` or `cURL` installed. The theme also requires shell access unless the `frozenCache` plain `TEX` option is enabled.

```
3235 \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.

```
3236 \AtEndOfPackage{
3237   \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).

```

3238 \markdownIfOption{noDefaults}{-}{
3239   \markdownSetup{theme=witiko/markdown/defaults}
3240 }
3241 }

3242 \ProvidesPackage{markdownthemewitiko_markdown_defaults}[2024/01/03]%

```

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

## 2.4 ConTeXt Interface

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

```

3243 \ExplSyntaxOn
3244 \tl_const:Nn \c_@@_option_layer_context_tl { context }
3245 \cs_generate_variant:Nn
3246   \tl_const:Nn
3247   { NV }
3248 \tl_if_exist:NF
3249   \c_@@_top_layer_tl
3250   {
3251     \tl_const:NV
3252       \c_@@_top_layer_tl
3253       \c_@@_option_layer_context_tl
3254   }
3255 \ExplSyntaxOff

```

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

```

3256 \writestatus{loading}{ConTeXt User Module / markdown}%
3257 \startmodule [markdown]
3258 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
3259   \do\#\do\^\do\_do\%do\~}%
3260 \input markdown/markdown

```

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

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

It is expected that the special plain  $\TeX$  characters have the expected category codes, when `\inputting` the file.



### 2.4.1 Typesetting Markdown

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

```
3261 \let\startmarkdown\relax
3262 \let\stopmarkdown\relax
3263 \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 Con $\TeX$ t 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 Con $\TeX$ t interface options (see Section 2.4.2) as its optional argument. These options will only influence this markdown document.

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

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

### 2.4.2 Options

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

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}`.

```

3264 \ExplSyntaxOn
3265 \cs_new:Npn
3266   \setupmarkdown
3267   [ #1 ]
3268   {
3269     \@@_setup:n
3270     { #1 }
3271   }
3272 \ExplSyntaxOff

```

#### 2.4.2.1 Generating Plain TeX Option Macros and Key-Values

Unlike plain TeX, we also accept caseless variants of options in line with the style of ConTeXt.

```

3273 \ExplSyntaxOn
3274 \cs_new:Nn \@@_caseless:N
3275   {
3276     \regex_replace_all:nnN
3277     { ([a-z])([A-Z]) }
3278     { \1 \c { str_lowercase:n } \cB{\ \2 \cE\} }
3279     #1
3280     \tl_set:Nx
3281     #1
3282     { #1 }
3283   }
3284 \seq_gput_right:Nn \g_@@_cases_seq { @@_caseless:N }

```

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

```

3285 \str_if_eq:VVT
3286   \c_@@_top_layer_tl
3287   \c_@@_option_layer_context_tl
3288   {
3289     \@@_define_option_commands_and_keyvals:
3290     \@@_define_renderers:
3291     \@@_define_renderer_prototypes:
3292   }
3293 \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.

```
3294 \startmodule[markdownthemewitiko_markdown_defaults]
3295 \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).

```
3296 local upper, format, length =
3297     string.upper, string.format, string.len
3298 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, Cp, any =
3299     lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
3300     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.

```
3301 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.

```
3302 function util.err(msg, exit_code)
3303     io.stderr:write("markdown.lua: " .. msg .. "\n")
3304     os.exit(exit_code or 1)
3305 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.

```
3306 function util.cache(dir, string, salt, transform, suffix)
3307     local digest = md5.sumhexa(string .. (salt or ""))
3308     local name = util.pathname(dir, digest .. suffix)
3309     local file = io.open(name, "r")
3310     if file == nil then -- If no cache entry exists, then create a new one.
3311         file = assert(io.open(name, "w"),
3312             [[Could not open file ]] .. name .. [[ for writing]])
3313         local result = string
3314         if transform ~= nil then
3315             result = transform(result)
3316         end
3317         assert(file:write(result))
```

```

3318     assert(file:close())
3319   end
3320   return name
3321 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.

```

3322 function util.cache_verbatim(dir, string)
3323   local name = util.cache(dir, string, nil, nil, ".verbatim")
3324   return name
3325 end

```

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

```

3326 function util.table_copy(t)
3327   local u = { }
3328   for k, v in pairs(t) do u[k] = v end
3329   return setmetatable(u, getmetatable(t))
3330 end

```

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

```

3331 function util.encode_json_string(s)
3332   s = s:gsub([[\\]], [[\\]])
3333   s = s:gsub([[\"]], [[\"]])
3334   return [[\"]] .. s .. [[\"]]
3335 end

```

The `util.expand_tabs_in_line` expands tabs in string `s`. If `tabstop` is specified, it is used as the tab stop width. Otherwise, the tab stop width of 4 characters is used. The method is a copy of the tab expansion algorithm from Ierusalimschy [11, Chapter 21].

```

3336 function util.expand_tabs_in_line(s, tabstop)
3337   local tab = tabstop or 4
3338   local corr = 0
3339   return (s:gsub("(\\t)", function(p)
3340     local sp = tab - (p - 1 + corr) % tab
3341     corr = corr - 1 + sp
3342     return string.rep(" ", sp)
3343   end))
3344 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.

```

3345 function util.walk(t, f)
3346   local typ = type(t)
3347   if typ == "string" then

```

```

3348     f(t)
3349 elseif typ == "table" then
3350     local i = 1
3351     local n
3352     n = t[i]
3353     while n do
3354         util.walk(n, f)
3355         i = i + 1
3356         n = t[i]
3357     end
3358 elseif typ == "function" then
3359     local ok, val = pcall(t)
3360     if ok then
3361         util.walk(val,f)
3362     end
3363 else
3364     f(tostring(t))
3365 end
3366 end

```

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

```

3367 function util.flatten(ary)
3368     local new = {}
3369     for _,v in ipairs(ary) do
3370         if type(v) == "table" then
3371             for _,w in ipairs(util.flatten(v)) do
3372                 new[#new + 1] = w
3373             end
3374         else
3375             new[#new + 1] = v
3376         end
3377     end
3378     return new
3379 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.

```

3380 function util.rope_to_string(rope)
3381     local buffer = {}
3382     util.walk(rope, function(x) buffer[#buffer + 1] = x end)
3383     return table.concat(buffer)
3384 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.

```

3385 function util.rope_last(rope)
3386     if #rope == 0 then

```

```

3387     return nil
3388 else
3389     local l = rope[#rope]
3390     if type(l) == "table" then
3391         return util.rope_last(l)
3392     else
3393         return l
3394     end
3395 end
3396 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  $1 \leq i \leq \#ary$ .

```

3397 function util.intersperse(ary, x)
3398     local new = {}
3399     local l = #ary
3400     for i,v in ipairs(ary) do
3401         local n = #new
3402         new[n + 1] = v
3403         if i ~= l then
3404             new[n + 2] = x
3405         end
3406     end
3407     return new
3408 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  $1 \leq i \leq \#ary$ .

```

3409 function util.map(ary, f)
3410     local new = {}
3411     for i,v in ipairs(ary) do
3412         new[i] = f(v)
3413     end
3414     return new
3415 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.

```

3416 function util.escaper(char_escapes, string_escapes)

```

Build a string of escapable characters.

```

3417     local char_escapes_list = ""
3418     for i,_ in pairs(char_escapes) do
3419         char_escapes_list = char_escapes_list .. i

```

```
3420 end
```

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

```
3421 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.

```
3422 if string_escapes then
3423   for k,v in pairs(string_escapes) do
3424     escapable = P(k) / v + escapable
3425   end
3426 end
```

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

```
3427 local escape_string = Cs((escapable + any)^0)
```

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

```
3428 return function(s)
3429   return lpeg.match(escape_string, s)
3430 end
3431 end
```

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

```
3432 function util.pathname(dir, file)
3433   if #dir == 0 then
3434     return file
3435   else
3436     return dir .. "/" .. file
3437   end
3438 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.

```
3439 local entities = {}
```



```
3440
3441 local character_entities = {
3442     ["Tab"] = 9,
3443     ["NewLine"] = 10,
3444     ["excl"] = 33,
3445     ["QUOT"] = 34,
3446     ["quot"] = 34,
3447     ["num"] = 35,
3448     ["dollar"] = 36,
3449     ["percent"] = 37,
3450     ["AMP"] = 38,
3451     ["amp"] = 38,
3452     ["apos"] = 39,
3453     ["lpar"] = 40,
3454     ["rpar"] = 41,
3455     ["ast"] = 42,
3456     ["midast"] = 42,
3457     ["plus"] = 43,
3458     ["comma"] = 44,
3459     ["period"] = 46,
3460     ["sol"] = 47,
3461     ["colon"] = 58,
3462     ["semi"] = 59,
3463     ["LT"] = 60,
3464     ["lt"] = 60,
3465     ["nvlT"] = {60, 8402},
3466     ["bne"] = {61, 8421},
3467     ["equals"] = 61,
3468     ["GT"] = 62,
3469     ["gt"] = 62,
3470     ["nvgt"] = {62, 8402},
3471     ["quest"] = 63,
3472     ["commat"] = 64,
3473     ["lbrack"] = 91,
3474     ["lsqb"] = 91,
3475     ["bsol"] = 92,
3476     ["rbrack"] = 93,
3477     ["rsqb"] = 93,
3478     ["Hat"] = 94,
3479     ["UnderBar"] = 95,
3480     ["lowbar"] = 95,
3481     ["DiacriticalGrave"] = 96,
3482     ["grave"] = 96,
3483     ["fjlig"] = {102, 106},
3484     ["lbrace"] = 123,
3485     ["lcub"] = 123,
3486     ["VerticalLine"] = 124,
```

3487 ["verbar"] = 124,  
3488 ["vert"] = 124,  
3489 ["rbrace"] = 125,  
3490 ["rcub"] = 125,  
3491 ["NonBreakingSpace"] = 160,  
3492 ["nbsp"] = 160,  
3493 ["iexcl"] = 161,  
3494 ["cent"] = 162,  
3495 ["pound"] = 163,  
3496 ["curren"] = 164,  
3497 ["yen"] = 165,  
3498 ["brvbar"] = 166,  
3499 ["sect"] = 167,  
3500 ["Dot"] = 168,  
3501 ["DoubleDot"] = 168,  
3502 ["die"] = 168,  
3503 ["uml"] = 168,  
3504 ["COPY"] = 169,  
3505 ["copy"] = 169,  
3506 ["ordf"] = 170,  
3507 ["laquo"] = 171,  
3508 ["not"] = 172,  
3509 ["shy"] = 173,  
3510 ["REG"] = 174,  
3511 ["circledR"] = 174,  
3512 ["reg"] = 174,  
3513 ["macr"] = 175,  
3514 ["strns"] = 175,  
3515 ["deg"] = 176,  
3516 ["PlusMinus"] = 177,  
3517 ["plusmn"] = 177,  
3518 ["pm"] = 177,  
3519 ["sup2"] = 178,  
3520 ["sup3"] = 179,  
3521 ["DiacriticalAcute"] = 180,  
3522 ["acute"] = 180,  
3523 ["micro"] = 181,  
3524 ["para"] = 182,  
3525 ["CenterDot"] = 183,  
3526 ["centerdot"] = 183,  
3527 ["middot"] = 183,  
3528 ["Cedilla"] = 184,  
3529 ["cedil"] = 184,  
3530 ["sup1"] = 185,  
3531 ["ordm"] = 186,  
3532 ["raquo"] = 187,  
3533 ["frac14"] = 188,

3534 ["frac12"] = 189,  
3535 ["half"] = 189,  
3536 ["frac34"] = 190,  
3537 ["iquest"] = 191,  
3538 ["Agrave"] = 192,  
3539 ["Aacute"] = 193,  
3540 ["Acirc"] = 194,  
3541 ["Atilde"] = 195,  
3542 ["Auml"] = 196,  
3543 ["Aring"] = 197,  
3544 ["angst"] = 197,  
3545 ["AElig"] = 198,  
3546 ["Ccedil"] = 199,  
3547 ["Egrave"] = 200,  
3548 ["Eacute"] = 201,  
3549 ["Ecirc"] = 202,  
3550 ["Euml"] = 203,  
3551 ["Igrave"] = 204,  
3552 ["Iacute"] = 205,  
3553 ["Icirc"] = 206,  
3554 ["Iuml"] = 207,  
3555 ["ETH"] = 208,  
3556 ["Ntilde"] = 209,  
3557 ["Ograve"] = 210,  
3558 ["Oacute"] = 211,  
3559 ["Ocirc"] = 212,  
3560 ["Otilde"] = 213,  
3561 ["Ouml"] = 214,  
3562 ["times"] = 215,  
3563 ["Oslash"] = 216,  
3564 ["Ugrave"] = 217,  
3565 ["Uacute"] = 218,  
3566 ["Ucirc"] = 219,  
3567 ["Uuml"] = 220,  
3568 ["Yacute"] = 221,  
3569 ["THORN"] = 222,  
3570 ["szlig"] = 223,  
3571 ["agrave"] = 224,  
3572 ["aacute"] = 225,  
3573 ["acirc"] = 226,  
3574 ["atilde"] = 227,  
3575 ["auml"] = 228,  
3576 ["aring"] = 229,  
3577 ["aelig"] = 230,  
3578 ["ccedil"] = 231,  
3579 ["egrave"] = 232,  
3580 ["eacute"] = 233,

3581 ["ecirc"] = 234,  
3582 ["euml"] = 235,  
3583 ["igrave"] = 236,  
3584 ["iacute"] = 237,  
3585 ["icirc"] = 238,  
3586 ["iuml"] = 239,  
3587 ["eth"] = 240,  
3588 ["ntilde"] = 241,  
3589 ["ograve"] = 242,  
3590 ["oacute"] = 243,  
3591 ["ocirc"] = 244,  
3592 ["otilde"] = 245,  
3593 ["ouml"] = 246,  
3594 ["div"] = 247,  
3595 ["divide"] = 247,  
3596 ["oslash"] = 248,  
3597 ["ugrave"] = 249,  
3598 ["uacute"] = 250,  
3599 ["ucirc"] = 251,  
3600 ["uuml"] = 252,  
3601 ["yacute"] = 253,  
3602 ["thorn"] = 254,  
3603 ["yuml"] = 255,  
3604 ["Amacr"] = 256,  
3605 ["amacr"] = 257,  
3606 ["Abreve"] = 258,  
3607 ["abreve"] = 259,  
3608 ["Aogon"] = 260,  
3609 ["aogon"] = 261,  
3610 ["Cacute"] = 262,  
3611 ["cacute"] = 263,  
3612 ["Ccirc"] = 264,  
3613 ["ccirc"] = 265,  
3614 ["Cdot"] = 266,  
3615 ["cdot"] = 267,  
3616 ["Ccaron"] = 268,  
3617 ["ccaron"] = 269,  
3618 ["Dcaron"] = 270,  
3619 ["dcaron"] = 271,  
3620 ["Dstrok"] = 272,  
3621 ["dstrok"] = 273,  
3622 ["Emacr"] = 274,  
3623 ["emacr"] = 275,  
3624 ["Edot"] = 278,  
3625 ["edot"] = 279,  
3626 ["Eogon"] = 280,  
3627 ["eogon"] = 281,

3628 ["Ecaron"] = 282,  
3629 ["ecaron"] = 283,  
3630 ["Gcirc"] = 284,  
3631 ["gcirc"] = 285,  
3632 ["Gbreve"] = 286,  
3633 ["gbreve"] = 287,  
3634 ["Gdot"] = 288,  
3635 ["gdot"] = 289,  
3636 ["Gcedil"] = 290,  
3637 ["Hcirc"] = 292,  
3638 ["hcirc"] = 293,  
3639 ["Hstrook"] = 294,  
3640 ["hstrook"] = 295,  
3641 ["Itilde"] = 296,  
3642 ["itilde"] = 297,  
3643 ["Imacr"] = 298,  
3644 ["imacr"] = 299,  
3645 ["Iogon"] = 302,  
3646 ["iogon"] = 303,  
3647 ["Idot"] = 304,  
3648 ["imath"] = 305,  
3649 ["inodot"] = 305,  
3650 ["IJlig"] = 306,  
3651 ["ijlig"] = 307,  
3652 ["Jcirc"] = 308,  
3653 ["jcirc"] = 309,  
3654 ["Kcedil"] = 310,  
3655 ["kcedil"] = 311,  
3656 ["kgreen"] = 312,  
3657 ["Lacute"] = 313,  
3658 ["lacute"] = 314,  
3659 ["Lcedil"] = 315,  
3660 ["lcedil"] = 316,  
3661 ["Lcaron"] = 317,  
3662 ["lcaron"] = 318,  
3663 ["Lmidot"] = 319,  
3664 ["lmidot"] = 320,  
3665 ["Lstrook"] = 321,  
3666 ["lstrook"] = 322,  
3667 ["Nacute"] = 323,  
3668 ["nacute"] = 324,  
3669 ["Ncedil"] = 325,  
3670 ["ncedil"] = 326,  
3671 ["Ncaron"] = 327,  
3672 ["ncaron"] = 328,  
3673 ["napos"] = 329,  
3674 ["ENG"] = 330,

3675 ["eng"] = 331,  
3676 ["Omacr"] = 332,  
3677 ["omacr"] = 333,  
3678 ["Odblac"] = 336,  
3679 ["odblac"] = 337,  
3680 ["OElig"] = 338,  
3681 ["oelig"] = 339,  
3682 ["Racute"] = 340,  
3683 ["racute"] = 341,  
3684 ["Rcedil"] = 342,  
3685 ["rcedil"] = 343,  
3686 ["Rcaron"] = 344,  
3687 ["rcaron"] = 345,  
3688 ["Sacute"] = 346,  
3689 ["sacute"] = 347,  
3690 ["Scirc"] = 348,  
3691 ["scirc"] = 349,  
3692 ["Scedil"] = 350,  
3693 ["scedil"] = 351,  
3694 ["Scaron"] = 352,  
3695 ["scaron"] = 353,  
3696 ["Tcedil"] = 354,  
3697 ["tcedil"] = 355,  
3698 ["Tcaron"] = 356,  
3699 ["tcaron"] = 357,  
3700 ["Tstrok"] = 358,  
3701 ["tstrok"] = 359,  
3702 ["Utilde"] = 360,  
3703 ["utilde"] = 361,  
3704 ["Umacr"] = 362,  
3705 ["umacr"] = 363,  
3706 ["Ubreve"] = 364,  
3707 ["ubreve"] = 365,  
3708 ["Uring"] = 366,  
3709 ["uring"] = 367,  
3710 ["Udblac"] = 368,  
3711 ["udblac"] = 369,  
3712 ["Uogon"] = 370,  
3713 ["uogon"] = 371,  
3714 ["Wcirc"] = 372,  
3715 ["wcirc"] = 373,  
3716 ["Ycirc"] = 374,  
3717 ["ycirc"] = 375,  
3718 ["Yuml"] = 376,  
3719 ["Zacute"] = 377,  
3720 ["zacute"] = 378,  
3721 ["Zdot"] = 379,

3722 ["zdot"] = 380,  
3723 ["Zcaron"] = 381,  
3724 ["zcaron"] = 382,  
3725 ["fnof"] = 402,  
3726 ["imped"] = 437,  
3727 ["gacute"] = 501,  
3728 ["jmath"] = 567,  
3729 ["circ"] = 710,  
3730 ["Hacek"] = 711,  
3731 ["caron"] = 711,  
3732 ["Breve"] = 728,  
3733 ["breve"] = 728,  
3734 ["DiacriticalDot"] = 729,  
3735 ["dot"] = 729,  
3736 ["ring"] = 730,  
3737 ["ogon"] = 731,  
3738 ["DiacriticalTilde"] = 732,  
3739 ["tilde"] = 732,  
3740 ["DiacriticalDoubleAcute"] = 733,  
3741 ["dblac"] = 733,  
3742 ["DownBreve"] = 785,  
3743 ["Alpha"] = 913,  
3744 ["Beta"] = 914,  
3745 ["Gamma"] = 915,  
3746 ["Delta"] = 916,  
3747 ["Epsilon"] = 917,  
3748 ["Zeta"] = 918,  
3749 ["Eta"] = 919,  
3750 ["Theta"] = 920,  
3751 ["Iota"] = 921,  
3752 ["Kappa"] = 922,  
3753 ["Lambda"] = 923,  
3754 ["Mu"] = 924,  
3755 ["Nu"] = 925,  
3756 ["Xi"] = 926,  
3757 ["Omicron"] = 927,  
3758 ["Pi"] = 928,  
3759 ["Rho"] = 929,  
3760 ["Sigma"] = 931,  
3761 ["Tau"] = 932,  
3762 ["Upsilon"] = 933,  
3763 ["Phi"] = 934,  
3764 ["Chi"] = 935,  
3765 ["Psi"] = 936,  
3766 ["Omega"] = 937,  
3767 ["ohm"] = 937,  
3768 ["alpha"] = 945,

3769 ["beta"] = 946,  
3770 ["gamma"] = 947,  
3771 ["delta"] = 948,  
3772 ["epsi"] = 949,  
3773 ["epsilon"] = 949,  
3774 ["zeta"] = 950,  
3775 ["eta"] = 951,  
3776 ["theta"] = 952,  
3777 ["iota"] = 953,  
3778 ["kappa"] = 954,  
3779 ["lambda"] = 955,  
3780 ["mu"] = 956,  
3781 ["nu"] = 957,  
3782 ["xi"] = 958,  
3783 ["omicron"] = 959,  
3784 ["pi"] = 960,  
3785 ["rho"] = 961,  
3786 ["sigmaf"] = 962,  
3787 ["sigmav"] = 962,  
3788 ["varsigma"] = 962,  
3789 ["sigma"] = 963,  
3790 ["tau"] = 964,  
3791 ["upsilon"] = 965,  
3792 ["upsilon"] = 965,  
3793 ["phi"] = 966,  
3794 ["chi"] = 967,  
3795 ["psi"] = 968,  
3796 ["omega"] = 969,  
3797 ["thetasym"] = 977,  
3798 ["thetav"] = 977,  
3799 ["vartheta"] = 977,  
3800 ["Upsilon"] = 978,  
3801 ["upsih"] = 978,  
3802 ["phiv"] = 981,  
3803 ["straightphi"] = 981,  
3804 ["varphi"] = 981,  
3805 ["piv"] = 982,  
3806 ["varpi"] = 982,  
3807 ["Gammad"] = 988,  
3808 ["digamma"] = 989,  
3809 ["gammad"] = 989,  
3810 ["kappav"] = 1008,  
3811 ["varkappa"] = 1008,  
3812 ["rhov"] = 1009,  
3813 ["varrho"] = 1009,  
3814 ["epsiv"] = 1013,  
3815 ["straightepsilon"] = 1013,



3816 ["varepsilon"] = 1013,  
3817 ["backepsilon"] = 1014,  
3818 ["bepsi"] = 1014,  
3819 ["IOcy"] = 1025,  
3820 ["DJcy"] = 1026,  
3821 ["GJcy"] = 1027,  
3822 ["Jukcy"] = 1028,  
3823 ["DScy"] = 1029,  
3824 ["Iukcy"] = 1030,  
3825 ["YIcy"] = 1031,  
3826 ["Jsercy"] = 1032,  
3827 ["LJcy"] = 1033,  
3828 ["NJcy"] = 1034,  
3829 ["TSHcy"] = 1035,  
3830 ["KJcy"] = 1036,  
3831 ["Ubrcy"] = 1038,  
3832 ["DZcy"] = 1039,  
3833 ["Acy"] = 1040,  
3834 ["Bcy"] = 1041,  
3835 ["Vcy"] = 1042,  
3836 ["Gcy"] = 1043,  
3837 ["Dcy"] = 1044,  
3838 ["IEcy"] = 1045,  
3839 ["ZHcy"] = 1046,  
3840 ["Zcy"] = 1047,  
3841 ["Icy"] = 1048,  
3842 ["Jcy"] = 1049,  
3843 ["Kcy"] = 1050,  
3844 ["Lcy"] = 1051,  
3845 ["Mcy"] = 1052,  
3846 ["Ncy"] = 1053,  
3847 ["Ocy"] = 1054,  
3848 ["Pcy"] = 1055,  
3849 ["Rcy"] = 1056,  
3850 ["Scy"] = 1057,  
3851 ["Tcy"] = 1058,  
3852 ["Ucy"] = 1059,  
3853 ["Fcy"] = 1060,  
3854 ["KHcy"] = 1061,  
3855 ["TScy"] = 1062,  
3856 ["CHcy"] = 1063,  
3857 ["SHcy"] = 1064,  
3858 ["SHCHcy"] = 1065,  
3859 ["HARDcy"] = 1066,  
3860 ["Ycy"] = 1067,  
3861 ["SOFTcy"] = 1068,  
3862 ["Ecy"] = 1069,

3863 ["YUcy"] = 1070,  
3864 ["YAcy"] = 1071,  
3865 ["acy"] = 1072,  
3866 ["bcy"] = 1073,  
3867 ["vcy"] = 1074,  
3868 ["gcy"] = 1075,  
3869 ["dcy"] = 1076,  
3870 ["iecy"] = 1077,  
3871 ["zhcy"] = 1078,  
3872 ["zcy"] = 1079,  
3873 ["icy"] = 1080,  
3874 ["jcy"] = 1081,  
3875 ["kcy"] = 1082,  
3876 ["lcy"] = 1083,  
3877 ["mcy"] = 1084,  
3878 ["ncy"] = 1085,  
3879 ["ocy"] = 1086,  
3880 ["pcy"] = 1087,  
3881 ["rcy"] = 1088,  
3882 ["scy"] = 1089,  
3883 ["tcy"] = 1090,  
3884 ["ucy"] = 1091,  
3885 ["fcy"] = 1092,  
3886 ["khcy"] = 1093,  
3887 ["tscy"] = 1094,  
3888 ["chcy"] = 1095,  
3889 ["shcy"] = 1096,  
3890 ["shchcy"] = 1097,  
3891 ["hardcy"] = 1098,  
3892 ["ycy"] = 1099,  
3893 ["softcy"] = 1100,  
3894 ["ecy"] = 1101,  
3895 ["yucy"] = 1102,  
3896 ["yacy"] = 1103,  
3897 ["iocy"] = 1105,  
3898 ["djcy"] = 1106,  
3899 ["gjcy"] = 1107,  
3900 ["jukcy"] = 1108,  
3901 ["dscy"] = 1109,  
3902 ["iukcy"] = 1110,  
3903 ["yicy"] = 1111,  
3904 ["jsercy"] = 1112,  
3905 ["ljcy"] = 1113,  
3906 ["njcy"] = 1114,  
3907 ["tshcy"] = 1115,  
3908 ["kjcy"] = 1116,  
3909 ["ubrcy"] = 1118,

3910 ["dzcycy"] = 1119,  
3911 ["ensp"] = 8194,  
3912 ["emsp"] = 8195,  
3913 ["emsp13"] = 8196,  
3914 ["emsp14"] = 8197,  
3915 ["numsp"] = 8199,  
3916 ["puncsp"] = 8200,  
3917 ["ThinSpace"] = 8201,  
3918 ["thinsp"] = 8201,  
3919 ["VeryThinSpace"] = 8202,  
3920 ["hairsp"] = 8202,  
3921 ["NegativeMediumSpace"] = 8203,  
3922 ["NegativeThickSpace"] = 8203,  
3923 ["NegativeThinSpace"] = 8203,  
3924 ["NegativeVeryThinSpace"] = 8203,  
3925 ["ZeroWidthSpace"] = 8203,  
3926 ["zwnj"] = 8204,  
3927 ["zwj"] = 8205,  
3928 ["lrm"] = 8206,  
3929 ["rlm"] = 8207,  
3930 ["dash"] = 8208,  
3931 ["hyphen"] = 8208,  
3932 ["ndash"] = 8211,  
3933 ["mdash"] = 8212,  
3934 ["horbar"] = 8213,  
3935 ["Verbar"] = 8214,  
3936 ["Vert"] = 8214,  
3937 ["OpenCurlyQuote"] = 8216,  
3938 ["lsquo"] = 8216,  
3939 ["CloseCurlyQuote"] = 8217,  
3940 ["rsquo"] = 8217,  
3941 ["rsquor"] = 8217,  
3942 ["lsquor"] = 8218,  
3943 ["sbquo"] = 8218,  
3944 ["OpenCurlyDoubleQuote"] = 8220,  
3945 ["ldquo"] = 8220,  
3946 ["CloseCurlyDoubleQuote"] = 8221,  
3947 ["rdquo"] = 8221,  
3948 ["rdquor"] = 8221,  
3949 ["bdquo"] = 8222,  
3950 ["ldquor"] = 8222,  
3951 ["dagger"] = 8224,  
3952 ["Dagger"] = 8225,  
3953 ["ddagger"] = 8225,  
3954 ["bull"] = 8226,  
3955 ["bullet"] = 8226,  
3956 ["nldr"] = 8229,

3957 ["hellip"] = 8230,  
 3958 ["mldr"] = 8230,  
 3959 ["permil"] = 8240,  
 3960 ["pertenk"] = 8241,  
 3961 ["prime"] = 8242,  
 3962 ["Prime"] = 8243,  
 3963 ["tprime"] = 8244,  
 3964 ["backprime"] = 8245,  
 3965 ["bprime"] = 8245,  
 3966 ["lsaquo"] = 8249,  
 3967 ["rsaquo"] = 8250,  
 3968 ["OverBar"] = 8254,  
 3969 ["oline"] = 8254,  
 3970 ["caret"] = 8257,  
 3971 ["hybull"] = 8259,  
 3972 ["frasl"] = 8260,  
 3973 ["bsemi"] = 8271,  
 3974 ["qprime"] = 8279,  
 3975 ["MediumSpace"] = 8287,  
 3976 ["ThickSpace"] = {8287, 8202},  
 3977 ["NoBreak"] = 8288,  
 3978 ["ApplyFunction"] = 8289,  
 3979 ["af"] = 8289,  
 3980 ["InvisibleTimes"] = 8290,  
 3981 ["it"] = 8290,  
 3982 ["InvisibleComma"] = 8291,  
 3983 ["ic"] = 8291,  
 3984 ["euro"] = 8364,  
 3985 ["TripleDot"] = 8411,  
 3986 ["tdot"] = 8411,  
 3987 ["DotDot"] = 8412,  
 3988 ["Copf"] = 8450,  
 3989 ["complexes"] = 8450,  
 3990 ["incare"] = 8453,  
 3991 ["gscr"] = 8458,  
 3992 ["HilbertSpace"] = 8459,  
 3993 ["Hscr"] = 8459,  
 3994 ["hamilt"] = 8459,  
 3995 ["Hfr"] = 8460,  
 3996 ["Poincareplane"] = 8460,  
 3997 ["Hopf"] = 8461,  
 3998 ["quaternions"] = 8461,  
 3999 ["planckh"] = 8462,  
 4000 ["hbar"] = 8463,  
 4001 ["hslash"] = 8463,  
 4002 ["planck"] = 8463,  
 4003 ["plankv"] = 8463,

4004 ["Iscr"] = 8464,  
4005 ["imagline"] = 8464,  
4006 ["Ifr"] = 8465,  
4007 ["Im"] = 8465,  
4008 ["image"] = 8465,  
4009 ["imagpart"] = 8465,  
4010 ["Laplacetrif"] = 8466,  
4011 ["Lscr"] = 8466,  
4012 ["lagran"] = 8466,  
4013 ["ell"] = 8467,  
4014 ["Nopf"] = 8469,  
4015 ["naturals"] = 8469,  
4016 ["numero"] = 8470,  
4017 ["copysr"] = 8471,  
4018 ["weierp"] = 8472,  
4019 ["wp"] = 8472,  
4020 ["Popf"] = 8473,  
4021 ["primes"] = 8473,  
4022 ["Qopf"] = 8474,  
4023 ["rationals"] = 8474,  
4024 ["Rscr"] = 8475,  
4025 ["realine"] = 8475,  
4026 ["Re"] = 8476,  
4027 ["Rfr"] = 8476,  
4028 ["real"] = 8476,  
4029 ["realpart"] = 8476,  
4030 ["Ropf"] = 8477,  
4031 ["reals"] = 8477,  
4032 ["rx"] = 8478,  
4033 ["TRADE"] = 8482,  
4034 ["trade"] = 8482,  
4035 ["Zopf"] = 8484,  
4036 ["integers"] = 8484,  
4037 ["mho"] = 8487,  
4038 ["Zfr"] = 8488,  
4039 ["zeetrif"] = 8488,  
4040 ["iiota"] = 8489,  
4041 ["Bernoullis"] = 8492,  
4042 ["Bscr"] = 8492,  
4043 ["bernou"] = 8492,  
4044 ["Cayleys"] = 8493,  
4045 ["Cfr"] = 8493,  
4046 ["escr"] = 8495,  
4047 ["Escr"] = 8496,  
4048 ["expectation"] = 8496,  
4049 ["Fouriertrif"] = 8497,  
4050 ["Fscr"] = 8497,

4051 ["Mellintrf"] = 8499,  
 4052 ["Mscr"] = 8499,  
 4053 ["phmmat"] = 8499,  
 4054 ["order"] = 8500,  
 4055 ["orderof"] = 8500,  
 4056 ["oscr"] = 8500,  
 4057 ["alefsym"] = 8501,  
 4058 ["aleph"] = 8501,  
 4059 ["beth"] = 8502,  
 4060 ["gimel"] = 8503,  
 4061 ["daleth"] = 8504,  
 4062 ["CapitalDifferentialD"] = 8517,  
 4063 ["DD"] = 8517,  
 4064 ["DifferentialD"] = 8518,  
 4065 ["dd"] = 8518,  
 4066 ["ExponentialE"] = 8519,  
 4067 ["ee"] = 8519,  
 4068 ["exponentiale"] = 8519,  
 4069 ["ImaginaryI"] = 8520,  
 4070 ["ii"] = 8520,  
 4071 ["frac13"] = 8531,  
 4072 ["frac23"] = 8532,  
 4073 ["frac15"] = 8533,  
 4074 ["frac25"] = 8534,  
 4075 ["frac35"] = 8535,  
 4076 ["frac45"] = 8536,  
 4077 ["frac16"] = 8537,  
 4078 ["frac56"] = 8538,  
 4079 ["frac18"] = 8539,  
 4080 ["frac38"] = 8540,  
 4081 ["frac58"] = 8541,  
 4082 ["frac78"] = 8542,  
 4083 ["LeftArrow"] = 8592,  
 4084 ["ShortLeftArrow"] = 8592,  
 4085 ["larr"] = 8592,  
 4086 ["leftarrow"] = 8592,  
 4087 ["slarr"] = 8592,  
 4088 ["ShortUpArrow"] = 8593,  
 4089 ["UpArrow"] = 8593,  
 4090 ["uarr"] = 8593,  
 4091 ["uparrow"] = 8593,  
 4092 ["RightArrow"] = 8594,  
 4093 ["ShortRightArrow"] = 8594,  
 4094 ["rarr"] = 8594,  
 4095 ["rightarrow"] = 8594,  
 4096 ["srarr"] = 8594,  
 4097 ["DownArrow"] = 8595,

4098 ["ShortDownArrow"] = 8595,  
 4099 ["darr"] = 8595,  
 4100 ["downarrow"] = 8595,  
 4101 ["LeftRightArrow"] = 8596,  
 4102 ["harr"] = 8596,  
 4103 ["leftrightarrow"] = 8596,  
 4104 ["UpDownArrow"] = 8597,  
 4105 ["updownarrow"] = 8597,  
 4106 ["varr"] = 8597,  
 4107 ["UpperLeftArrow"] = 8598,  
 4108 ["nwarr"] = 8598,  
 4109 ["nwarrow"] = 8598,  
 4110 ["UpperRightArrow"] = 8599,  
 4111 ["nearr"] = 8599,  
 4112 ["nearrow"] = 8599,  
 4113 ["LowerRightArrow"] = 8600,  
 4114 ["searr"] = 8600,  
 4115 ["searrow"] = 8600,  
 4116 ["LowerLeftArrow"] = 8601,  
 4117 ["swarr"] = 8601,  
 4118 ["swarrow"] = 8601,  
 4119 ["nlarr"] = 8602,  
 4120 ["nleftarrow"] = 8602,  
 4121 ["nrarr"] = 8603,  
 4122 ["nrightarrow"] = 8603,  
 4123 ["nrarrw"] = {8605, 824},  
 4124 ["rarrw"] = 8605,  
 4125 ["rightsquigarrow"] = 8605,  
 4126 ["Larr"] = 8606,  
 4127 ["twoheadleftarrow"] = 8606,  
 4128 ["Uarr"] = 8607,  
 4129 ["Rarr"] = 8608,  
 4130 ["twoheadrightarrow"] = 8608,  
 4131 ["Darr"] = 8609,  
 4132 ["larrtl"] = 8610,  
 4133 ["leftarrowtail"] = 8610,  
 4134 ["rarrtl"] = 8611,  
 4135 ["rightarrowtail"] = 8611,  
 4136 ["LeftTeeArrow"] = 8612,  
 4137 ["mapstoleft"] = 8612,  
 4138 ["UpTeeArrow"] = 8613,  
 4139 ["mapstoup"] = 8613,  
 4140 ["RightTeeArrow"] = 8614,  
 4141 ["map"] = 8614,  
 4142 ["mapsto"] = 8614,  
 4143 ["DownTeeArrow"] = 8615,  
 4144 ["mapstodown"] = 8615,

4145 ["hookleftarrow"] = 8617,  
 4146 ["larrhk"] = 8617,  
 4147 ["hookrightarrow"] = 8618,  
 4148 ["rarrhk"] = 8618,  
 4149 ["larrlp"] = 8619,  
 4150 ["looparrowleft"] = 8619,  
 4151 ["looparrowright"] = 8620,  
 4152 ["rarrlp"] = 8620,  
 4153 ["harrw"] = 8621,  
 4154 ["leftrightsquigarrow"] = 8621,  
 4155 ["nharr"] = 8622,  
 4156 ["nletrightarrow"] = 8622,  
 4157 ["Lsh"] = 8624,  
 4158 ["lsh"] = 8624,  
 4159 ["Rsh"] = 8625,  
 4160 ["rsh"] = 8625,  
 4161 ["ldsh"] = 8626,  
 4162 ["rdsh"] = 8627,  
 4163 ["crarr"] = 8629,  
 4164 ["cularr"] = 8630,  
 4165 ["curvearrowleft"] = 8630,  
 4166 ["curarr"] = 8631,  
 4167 ["curvearrowright"] = 8631,  
 4168 ["circlearrowleft"] = 8634,  
 4169 ["olarr"] = 8634,  
 4170 ["circlearrowright"] = 8635,  
 4171 ["orarr"] = 8635,  
 4172 ["LeftVector"] = 8636,  
 4173 ["leftharpoonup"] = 8636,  
 4174 ["lharu"] = 8636,  
 4175 ["DownLeftVector"] = 8637,  
 4176 ["leftharpoondown"] = 8637,  
 4177 ["lhard"] = 8637,  
 4178 ["RightUpVector"] = 8638,  
 4179 ["uharr"] = 8638,  
 4180 ["upharpoonright"] = 8638,  
 4181 ["LeftUpVector"] = 8639,  
 4182 ["uharl"] = 8639,  
 4183 ["upharpoonleft"] = 8639,  
 4184 ["RightVector"] = 8640,  
 4185 ["rharu"] = 8640,  
 4186 ["rightharpoonup"] = 8640,  
 4187 ["DownRightVector"] = 8641,  
 4188 ["rhard"] = 8641,  
 4189 ["rightharpoondown"] = 8641,  
 4190 ["RightDownVector"] = 8642,  
 4191 ["dharr"] = 8642,



4192 ["downharpoonright"] = 8642,  
 4193 ["LeftDownVector"] = 8643,  
 4194 ["dharl"] = 8643,  
 4195 ["downharpoonleft"] = 8643,  
 4196 ["RightArrowLeftArrow"] = 8644,  
 4197 ["rightleftarrows"] = 8644,  
 4198 ["rlarr"] = 8644,  
 4199 ["UpArrowDownArrow"] = 8645,  
 4200 ["udarr"] = 8645,  
 4201 ["LeftArrowRightArrow"] = 8646,  
 4202 ["leftrightarrows"] = 8646,  
 4203 ["lrarr"] = 8646,  
 4204 ["leftleftarrows"] = 8647,  
 4205 ["llarr"] = 8647,  
 4206 ["upuparrows"] = 8648,  
 4207 ["uuarr"] = 8648,  
 4208 ["rightrightarrows"] = 8649,  
 4209 ["rrarr"] = 8649,  
 4210 ["ddarr"] = 8650,  
 4211 ["downdownarrows"] = 8650,  
 4212 ["ReverseEquilibrium"] = 8651,  
 4213 ["leftrightharpoons"] = 8651,  
 4214 ["lrhar"] = 8651,  
 4215 ["Equilibrium"] = 8652,  
 4216 ["rightleftharpoons"] = 8652,  
 4217 ["rlhar"] = 8652,  
 4218 ["nLeftarrow"] = 8653,  
 4219 ["nLArr"] = 8653,  
 4220 ["nLeftrightarrow"] = 8654,  
 4221 ["nhArr"] = 8654,  
 4222 ["nRightarrow"] = 8655,  
 4223 ["nrArr"] = 8655,  
 4224 ["DoubleLeftArrow"] = 8656,  
 4225 ["Leftarrow"] = 8656,  
 4226 ["lArr"] = 8656,  
 4227 ["DoubleUpArrow"] = 8657,  
 4228 ["Uparrow"] = 8657,  
 4229 ["uArr"] = 8657,  
 4230 ["DoubleRightArrow"] = 8658,  
 4231 ["Implies"] = 8658,  
 4232 ["Rightarrow"] = 8658,  
 4233 ["rArr"] = 8658,  
 4234 ["DoubleDownArrow"] = 8659,  
 4235 ["Downarrow"] = 8659,  
 4236 ["dArr"] = 8659,  
 4237 ["DoubleLeftRightArrow"] = 8660,  
 4238 ["Leftrightarrow"] = 8660,

4239 ["hArr"] = 8660,  
 4240 ["iff"] = 8660,  
 4241 ["DoubleUpDownArrow"] = 8661,  
 4242 ["Updownarrow"] = 8661,  
 4243 ["vArr"] = 8661,  
 4244 ["nwArr"] = 8662,  
 4245 ["neArr"] = 8663,  
 4246 ["seArr"] = 8664,  
 4247 ["swArr"] = 8665,  
 4248 ["Lleftarrow"] = 8666,  
 4249 ["lAarr"] = 8666,  
 4250 ["Rrightarrow"] = 8667,  
 4251 ["rAarr"] = 8667,  
 4252 ["zigrarr"] = 8669,  
 4253 ["LeftArrowBar"] = 8676,  
 4254 ["larrb"] = 8676,  
 4255 ["RightArrowBar"] = 8677,  
 4256 ["rarrb"] = 8677,  
 4257 ["DownArrowUpArrow"] = 8693,  
 4258 ["duarr"] = 8693,  
 4259 ["loarr"] = 8701,  
 4260 ["roarr"] = 8702,  
 4261 ["hoarr"] = 8703,  
 4262 ["ForAll"] = 8704,  
 4263 ["forall"] = 8704,  
 4264 ["comp"] = 8705,  
 4265 ["complement"] = 8705,  
 4266 ["PartialD"] = 8706,  
 4267 ["npart"] = {8706, 824},  
 4268 ["part"] = 8706,  
 4269 ["Exists"] = 8707,  
 4270 ["exist"] = 8707,  
 4271 ["NotExists"] = 8708,  
 4272 ["nexist"] = 8708,  
 4273 ["nexists"] = 8708,  
 4274 ["empty"] = 8709,  
 4275 ["emptyset"] = 8709,  
 4276 ["emptyv"] = 8709,  
 4277 ["varnothing"] = 8709,  
 4278 ["Del"] = 8711,  
 4279 ["nabla"] = 8711,  
 4280 ["Element"] = 8712,  
 4281 ["in"] = 8712,  
 4282 ["isin"] = 8712,  
 4283 ["isinv"] = 8712,  
 4284 ["NotElement"] = 8713,  
 4285 ["notin"] = 8713,

4286 ["notinva"] = 8713,  
 4287 ["ReverseElement"] = 8715,  
 4288 ["SuchThat"] = 8715,  
 4289 ["ni"] = 8715,  
 4290 ["niv"] = 8715,  
 4291 ["NotReverseElement"] = 8716,  
 4292 ["notni"] = 8716,  
 4293 ["notniva"] = 8716,  
 4294 ["Product"] = 8719,  
 4295 ["prod"] = 8719,  
 4296 ["Coproduct"] = 8720,  
 4297 ["coprod"] = 8720,  
 4298 ["Sum"] = 8721,  
 4299 ["sum"] = 8721,  
 4300 ["minus"] = 8722,  
 4301 ["MinusPlus"] = 8723,  
 4302 ["mnplus"] = 8723,  
 4303 ["mp"] = 8723,  
 4304 ["dotplus"] = 8724,  
 4305 ["plusdo"] = 8724,  
 4306 ["Backslash"] = 8726,  
 4307 ["setminus"] = 8726,  
 4308 ["setmn"] = 8726,  
 4309 ["smallsetminus"] = 8726,  
 4310 ["ssetmn"] = 8726,  
 4311 ["lowast"] = 8727,  
 4312 ["SmallCircle"] = 8728,  
 4313 ["compfn"] = 8728,  
 4314 ["Sqrt"] = 8730,  
 4315 ["radic"] = 8730,  
 4316 ["Proportional"] = 8733,  
 4317 ["prop"] = 8733,  
 4318 ["propto"] = 8733,  
 4319 ["varpropto"] = 8733,  
 4320 ["vprop"] = 8733,  
 4321 ["infin"] = 8734,  
 4322 ["angrt"] = 8735,  
 4323 ["ang"] = 8736,  
 4324 ["angle"] = 8736,  
 4325 ["nang"] = {8736, 8402},  
 4326 ["angmsd"] = 8737,  
 4327 ["measuredangle"] = 8737,  
 4328 ["angsph"] = 8738,  
 4329 ["VerticalBar"] = 8739,  
 4330 ["mid"] = 8739,  
 4331 ["shortmid"] = 8739,  
 4332 ["smid"] = 8739,

4333 ["NotVerticalBar"] = 8740,  
 4334 ["nmid"] = 8740,  
 4335 ["nshortmid"] = 8740,  
 4336 ["nsmid"] = 8740,  
 4337 ["DoubleVerticalBar"] = 8741,  
 4338 ["par"] = 8741,  
 4339 ["parallel"] = 8741,  
 4340 ["shortparallel"] = 8741,  
 4341 ["spar"] = 8741,  
 4342 ["NotDoubleVerticalBar"] = 8742,  
 4343 ["npar"] = 8742,  
 4344 ["nparallel"] = 8742,  
 4345 ["nshortparallel"] = 8742,  
 4346 ["nspar"] = 8742,  
 4347 ["and"] = 8743,  
 4348 ["wedge"] = 8743,  
 4349 ["or"] = 8744,  
 4350 ["vee"] = 8744,  
 4351 ["cap"] = 8745,  
 4352 ["caps"] = {8745, 65024},  
 4353 ["cup"] = 8746,  
 4354 ["cups"] = {8746, 65024},  
 4355 ["Integral"] = 8747,  
 4356 ["int"] = 8747,  
 4357 ["Int"] = 8748,  
 4358 ["iiint"] = 8749,  
 4359 ["tint"] = 8749,  
 4360 ["ContourIntegral"] = 8750,  
 4361 ["conint"] = 8750,  
 4362 ["oint"] = 8750,  
 4363 ["Conint"] = 8751,  
 4364 ["DoubleContourIntegral"] = 8751,  
 4365 ["Cconint"] = 8752,  
 4366 ["cwint"] = 8753,  
 4367 ["ClockwiseContourIntegral"] = 8754,  
 4368 ["cwconint"] = 8754,  
 4369 ["CounterClockwiseContourIntegral"] = 8755,  
 4370 ["awconint"] = 8755,  
 4371 ["Therefore"] = 8756,  
 4372 ["there4"] = 8756,  
 4373 ["therefore"] = 8756,  
 4374 ["Because"] = 8757,  
 4375 ["because"] = 8757,  
 4376 ["because"] = 8757,  
 4377 ["ratio"] = 8758,  
 4378 ["Colon"] = 8759,  
 4379 ["Proportion"] = 8759,

4380 ["dotminus"] = 8760,  
4381 ["minusd"] = 8760,  
4382 ["mDDot"] = 8762,  
4383 ["homtht"] = 8763,  
4384 ["Tilde"] = 8764,  
4385 ["nvsim"] = {8764, 8402},  
4386 ["sim"] = 8764,  
4387 ["thicksim"] = 8764,  
4388 ["thksim"] = 8764,  
4389 ["backsim"] = 8765,  
4390 ["bsim"] = 8765,  
4391 ["race"] = {8765, 817},  
4392 ["ac"] = 8766,  
4393 ["acE"] = {8766, 819},  
4394 ["mstpos"] = 8766,  
4395 ["acd"] = 8767,  
4396 ["VerticalTilde"] = 8768,  
4397 ["wr"] = 8768,  
4398 ["wreath"] = 8768,  
4399 ["NotTilde"] = 8769,  
4400 ["nsim"] = 8769,  
4401 ["EqualTilde"] = 8770,  
4402 ["NotEqualTilde"] = {8770, 824},  
4403 ["eqsim"] = 8770,  
4404 ["esim"] = 8770,  
4405 ["nesim"] = {8770, 824},  
4406 ["TildeEqual"] = 8771,  
4407 ["sime"] = 8771,  
4408 ["simeq"] = 8771,  
4409 ["NotTildeEqual"] = 8772,  
4410 ["nsime"] = 8772,  
4411 ["nsimeq"] = 8772,  
4412 ["TildeFullEqual"] = 8773,  
4413 ["cong"] = 8773,  
4414 ["simne"] = 8774,  
4415 ["NotTildeFullEqual"] = 8775,  
4416 ["ncong"] = 8775,  
4417 ["TildeTilde"] = 8776,  
4418 ["ap"] = 8776,  
4419 ["approx"] = 8776,  
4420 ["asyp"] = 8776,  
4421 ["thickapprox"] = 8776,  
4422 ["thkap"] = 8776,  
4423 ["NotTildeTilde"] = 8777,  
4424 ["nap"] = 8777,  
4425 ["napprox"] = 8777,  
4426 ["ape"] = 8778,

```

4427 ["approxeq"] = 8778,
4428 ["apid"] = 8779,
4429 ["napid"] = {8779, 824},
4430 ["backcong"] = 8780,
4431 ["bcong"] = 8780,
4432 ["CupCap"] = 8781,
4433 ["asympeq"] = 8781,
4434 ["nvap"] = {8781, 8402},
4435 ["Bumpeq"] = 8782,
4436 ["HumpDownHump"] = 8782,
4437 ["NotHumpDownHump"] = {8782, 824},
4438 ["bump"] = 8782,
4439 ["nbump"] = {8782, 824},
4440 ["HumpEqual"] = 8783,
4441 ["NotHumpEqual"] = {8783, 824},
4442 ["bumpe"] = 8783,
4443 ["bumpeq"] = 8783,
4444 ["nbumpe"] = {8783, 824},
4445 ["DotEqual"] = 8784,
4446 ["doteq"] = 8784,
4447 ["esdot"] = 8784,
4448 ["nedot"] = {8784, 824},
4449 ["doteqdot"] = 8785,
4450 ["eDot"] = 8785,
4451 ["efDot"] = 8786,
4452 ["fallingdotseq"] = 8786,
4453 ["erDot"] = 8787,
4454 ["risingdotseq"] = 8787,
4455 ["Assign"] = 8788,
4456 ["colone"] = 8788,
4457 ["coloneq"] = 8788,
4458 ["ecolon"] = 8789,
4459 ["eqcolon"] = 8789,
4460 ["ecir"] = 8790,
4461 ["eqcirc"] = 8790,
4462 ["circeq"] = 8791,
4463 ["cire"] = 8791,
4464 ["wedgeq"] = 8793,
4465 ["veeeq"] = 8794,
4466 ["triangleq"] = 8796,
4467 ["trie"] = 8796,
4468 ["equest"] = 8799,
4469 ["questeq"] = 8799,
4470 ["NotEqual"] = 8800,
4471 ["ne"] = 8800,
4472 ["Congruent"] = 8801,
4473 ["bnequiv"] = {8801, 8421},

```

4474 ["equiv"] = 8801,  
4475 ["NotCongruent"] = 8802,  
4476 ["nequiv"] = 8802,  
4477 ["le"] = 8804,  
4478 ["leq"] = 8804,  
4479 ["nvle"] = {8804, 8402},  
4480 ["GreaterEqual"] = 8805,  
4481 ["ge"] = 8805,  
4482 ["geq"] = 8805,  
4483 ["nvge"] = {8805, 8402},  
4484 ["LessFullEqual"] = 8806,  
4485 ["lE"] = 8806,  
4486 ["leqq"] = 8806,  
4487 ["nLE"] = {8806, 824},  
4488 ["nleqq"] = {8806, 824},  
4489 ["GreaterFullEqual"] = 8807,  
4490 ["NotGreaterFullEqual"] = {8807, 824},  
4491 ["gE"] = 8807,  
4492 ["geqq"] = 8807,  
4493 ["ngE"] = {8807, 824},  
4494 ["ngeqq"] = {8807, 824},  
4495 ["lnE"] = 8808,  
4496 ["lneqq"] = 8808,  
4497 ["lvertneqq"] = {8808, 65024},  
4498 ["lvnE"] = {8808, 65024},  
4499 ["gnE"] = 8809,  
4500 ["gneqq"] = 8809,  
4501 ["gvertneqq"] = {8809, 65024},  
4502 ["gvnE"] = {8809, 65024},  
4503 ["Lt"] = 8810,  
4504 ["NestedLessLess"] = 8810,  
4505 ["NotLessLess"] = {8810, 824},  
4506 ["ll"] = 8810,  
4507 ["nLt"] = {8810, 8402},  
4508 ["nLtv"] = {8810, 824},  
4509 ["Gt"] = 8811,  
4510 ["NestedGreaterGreater"] = 8811,  
4511 ["NotGreaterGreater"] = {8811, 824},  
4512 ["gg"] = 8811,  
4513 ["nGt"] = {8811, 8402},  
4514 ["nGtv"] = {8811, 824},  
4515 ["between"] = 8812,  
4516 ["twixt"] = 8812,  
4517 ["NotCupCap"] = 8813,  
4518 ["NotLess"] = 8814,  
4519 ["nless"] = 8814,  
4520 ["nlt"] = 8814,

```

4521 ["NotGreater"] = 8815,
4522 ["ngt"] = 8815,
4523 ["ngtr"] = 8815,
4524 ["NotLessEqual"] = 8816,
4525 ["nle"] = 8816,
4526 ["nleq"] = 8816,
4527 ["NotGreaterEqual"] = 8817,
4528 ["nge"] = 8817,
4529 ["ngeq"] = 8817,
4530 ["LessTilde"] = 8818,
4531 ["lesssim"] = 8818,
4532 ["lsim"] = 8818,
4533 ["GreaterTilde"] = 8819,
4534 ["gsim"] = 8819,
4535 ["gtrsim"] = 8819,
4536 ["NotLessTilde"] = 8820,
4537 ["nlsim"] = 8820,
4538 ["NotGreaterTilde"] = 8821,
4539 ["ngsim"] = 8821,
4540 ["LessGreater"] = 8822,
4541 ["lessgtr"] = 8822,
4542 ["lg"] = 8822,
4543 ["GreaterLess"] = 8823,
4544 ["gl"] = 8823,
4545 ["gtrless"] = 8823,
4546 ["NotLessGreater"] = 8824,
4547 ["ntlg"] = 8824,
4548 ["NotGreaterLess"] = 8825,
4549 ["ntgl"] = 8825,
4550 ["Precedes"] = 8826,
4551 ["pr"] = 8826,
4552 ["prec"] = 8826,
4553 ["Succeeds"] = 8827,
4554 ["sc"] = 8827,
4555 ["succ"] = 8827,
4556 ["PrecedesSlantEqual"] = 8828,
4557 ["prcue"] = 8828,
4558 ["preccurlyeq"] = 8828,
4559 ["SucceedsSlantEqual"] = 8829,
4560 ["sccue"] = 8829,
4561 ["succcurlyeq"] = 8829,
4562 ["PrecedesTilde"] = 8830,
4563 ["precsim"] = 8830,
4564 ["prsim"] = 8830,
4565 ["NotSucceedsTilde"] = {8831, 824},
4566 ["SucceedsTilde"] = 8831,
4567 ["scsim"] = 8831,

```



4568 ["succsim"] = 8831,  
4569 ["NotPrecedes"] = 8832,  
4570 ["npr"] = 8832,  
4571 ["nprec"] = 8832,  
4572 ["NotSucceeds"] = 8833,  
4573 ["nsc"] = 8833,  
4574 ["nsucc"] = 8833,  
4575 ["NotSubset"] = {8834, 8402},  
4576 ["nsubset"] = {8834, 8402},  
4577 ["sub"] = 8834,  
4578 ["subset"] = 8834,  
4579 ["vnsup"] = {8834, 8402},  
4580 ["NotSuperset"] = {8835, 8402},  
4581 ["Superset"] = 8835,  
4582 ["nsupset"] = {8835, 8402},  
4583 ["sup"] = 8835,  
4584 ["supset"] = 8835,  
4585 ["vnsup"] = {8835, 8402},  
4586 ["nsub"] = 8836,  
4587 ["nsup"] = 8837,  
4588 ["SubsetEqual"] = 8838,  
4589 ["sube"] = 8838,  
4590 ["subseteq"] = 8838,  
4591 ["SupersetEqual"] = 8839,  
4592 ["supe"] = 8839,  
4593 ["supseteq"] = 8839,  
4594 ["NotSubsetEqual"] = 8840,  
4595 ["nsube"] = 8840,  
4596 ["nsubseteq"] = 8840,  
4597 ["NotSupersetEqual"] = 8841,  
4598 ["nsupe"] = 8841,  
4599 ["nsupseteq"] = 8841,  
4600 ["subne"] = 8842,  
4601 ["subsetneq"] = 8842,  
4602 ["varsubsetneq"] = {8842, 65024},  
4603 ["vsubne"] = {8842, 65024},  
4604 ["supne"] = 8843,  
4605 ["supsetneq"] = 8843,  
4606 ["varsupsetneq"] = {8843, 65024},  
4607 ["vsupne"] = {8843, 65024},  
4608 ["cupdot"] = 8845,  
4609 ["UnionPlus"] = 8846,  
4610 ["uplus"] = 8846,  
4611 ["NotSquareSubset"] = {8847, 824},  
4612 ["SquareSubset"] = 8847,  
4613 ["sqsub"] = 8847,  
4614 ["sqsubset"] = 8847,

4615 ["NotSquareSuperset"] = {8848, 824},  
 4616 ["SquareSuperset"] = 8848,  
 4617 ["sqsup"] = 8848,  
 4618 ["sqsupset"] = 8848,  
 4619 ["SquareSubsetEqual"] = 8849,  
 4620 ["sqsube"] = 8849,  
 4621 ["sqsubseteq"] = 8849,  
 4622 ["SquareSupersetEqual"] = 8850,  
 4623 ["sqsupe"] = 8850,  
 4624 ["sqsupseteq"] = 8850,  
 4625 ["SquareIntersection"] = 8851,  
 4626 ["sqcap"] = 8851,  
 4627 ["sqcaps"] = {8851, 65024},  
 4628 ["SquareUnion"] = 8852,  
 4629 ["sqcup"] = 8852,  
 4630 ["sqcups"] = {8852, 65024},  
 4631 ["CirclePlus"] = 8853,  
 4632 ["oplus"] = 8853,  
 4633 ["CircleMinus"] = 8854,  
 4634 ["ominus"] = 8854,  
 4635 ["CircleTimes"] = 8855,  
 4636 ["otimes"] = 8855,  
 4637 ["osol"] = 8856,  
 4638 ["CircleDot"] = 8857,  
 4639 ["odot"] = 8857,  
 4640 ["circledcirc"] = 8858,  
 4641 ["ocir"] = 8858,  
 4642 ["circledast"] = 8859,  
 4643 ["oast"] = 8859,  
 4644 ["circleddash"] = 8861,  
 4645 ["odash"] = 8861,  
 4646 ["boxplus"] = 8862,  
 4647 ["plusb"] = 8862,  
 4648 ["boxminus"] = 8863,  
 4649 ["minusb"] = 8863,  
 4650 ["boxtimes"] = 8864,  
 4651 ["timesb"] = 8864,  
 4652 ["dotsquare"] = 8865,  
 4653 ["sdotb"] = 8865,  
 4654 ["RightTee"] = 8866,  
 4655 ["vdash"] = 8866,  
 4656 ["LeftTee"] = 8867,  
 4657 ["dashv"] = 8867,  
 4658 ["DownTee"] = 8868,  
 4659 ["top"] = 8868,  
 4660 ["UpTee"] = 8869,  
 4661 ["bot"] = 8869,

4662 ["bottom"] = 8869,  
 4663 ["perp"] = 8869,  
 4664 ["models"] = 8871,  
 4665 ["DoubleRightTee"] = 8872,  
 4666 ["vDash"] = 8872,  
 4667 ["Vdash"] = 8873,  
 4668 ["Vvdash"] = 8874,  
 4669 ["VDash"] = 8875,  
 4670 ["nvdash"] = 8876,  
 4671 ["nvDash"] = 8877,  
 4672 ["nVdash"] = 8878,  
 4673 ["nVDash"] = 8879,  
 4674 ["prurel"] = 8880,  
 4675 ["LeftTriangle"] = 8882,  
 4676 ["vartriangleleft"] = 8882,  
 4677 ["vltri"] = 8882,  
 4678 ["RightTriangle"] = 8883,  
 4679 ["vartriangleright"] = 8883,  
 4680 ["vrtri"] = 8883,  
 4681 ["LeftTriangleEqual"] = 8884,  
 4682 ["ltrie"] = 8884,  
 4683 ["nvltrie"] = {8884, 8402},  
 4684 ["trianglelefteq"] = 8884,  
 4685 ["RightTriangleEqual"] = 8885,  
 4686 ["nvrtrie"] = {8885, 8402},  
 4687 ["rtrie"] = 8885,  
 4688 ["trianglerighteq"] = 8885,  
 4689 ["origof"] = 8886,  
 4690 ["imof"] = 8887,  
 4691 ["multimap"] = 8888,  
 4692 ["mumap"] = 8888,  
 4693 ["hercon"] = 8889,  
 4694 ["intcal"] = 8890,  
 4695 ["intercal"] = 8890,  
 4696 ["veebar"] = 8891,  
 4697 ["barvee"] = 8893,  
 4698 ["angrtvb"] = 8894,  
 4699 ["ltri"] = 8895,  
 4700 ["Wedge"] = 8896,  
 4701 ["bigwedge"] = 8896,  
 4702 ["xwedge"] = 8896,  
 4703 ["Vee"] = 8897,  
 4704 ["bigvee"] = 8897,  
 4705 ["xvee"] = 8897,  
 4706 ["Intersection"] = 8898,  
 4707 ["bigcap"] = 8898,  
 4708 ["xcap"] = 8898,

4709 ["Union"] = 8899,  
4710 ["bigcup"] = 8899,  
4711 ["xcup"] = 8899,  
4712 ["Diamond"] = 8900,  
4713 ["diam"] = 8900,  
4714 ["diamond"] = 8900,  
4715 ["sdot"] = 8901,  
4716 ["Star"] = 8902,  
4717 ["sstarf"] = 8902,  
4718 ["divideontimes"] = 8903,  
4719 ["divonx"] = 8903,  
4720 ["bowtie"] = 8904,  
4721 ["ltimes"] = 8905,  
4722 ["rtimes"] = 8906,  
4723 ["leftthreetimes"] = 8907,  
4724 ["lthree"] = 8907,  
4725 ["rightthreetimes"] = 8908,  
4726 ["rthree"] = 8908,  
4727 ["backsimeq"] = 8909,  
4728 ["bsime"] = 8909,  
4729 ["curlyvee"] = 8910,  
4730 ["cuvee"] = 8910,  
4731 ["curlywedge"] = 8911,  
4732 ["cuwed"] = 8911,  
4733 ["Sub"] = 8912,  
4734 ["Subset"] = 8912,  
4735 ["Sup"] = 8913,  
4736 ["Supset"] = 8913,  
4737 ["Cap"] = 8914,  
4738 ["Cup"] = 8915,  
4739 ["fork"] = 8916,  
4740 ["pitchfork"] = 8916,  
4741 ["epar"] = 8917,  
4742 ["lessdot"] = 8918,  
4743 ["ltdot"] = 8918,  
4744 ["gtdot"] = 8919,  
4745 ["gtrdot"] = 8919,  
4746 ["L1"] = 8920,  
4747 ["nL1"] = {8920, 824},  
4748 ["Gg"] = 8921,  
4749 ["ggg"] = 8921,  
4750 ["nGg"] = {8921, 824},  
4751 ["LessEqualGreater"] = 8922,  
4752 ["leg"] = 8922,  
4753 ["lesg"] = {8922, 65024},  
4754 ["lesseqgtr"] = 8922,  
4755 ["GreaterEqualLess"] = 8923,

4756 ["gel"] = 8923,  
4757 ["gesl"] = {8923, 65024},  
4758 ["gtreqless"] = 8923,  
4759 ["cuepr"] = 8926,  
4760 ["curlyeqprec"] = 8926,  
4761 ["cuesc"] = 8927,  
4762 ["curlyeqsucc"] = 8927,  
4763 ["NotPrecedesSlantEqual"] = 8928,  
4764 ["nprcue"] = 8928,  
4765 ["NotSucceedsSlantEqual"] = 8929,  
4766 ["nsccue"] = 8929,  
4767 ["NotSquareSubsetEqual"] = 8930,  
4768 ["nsqsube"] = 8930,  
4769 ["NotSquareSupersetEqual"] = 8931,  
4770 ["nsqsupe"] = 8931,  
4771 ["lnsim"] = 8934,  
4772 ["gnsim"] = 8935,  
4773 ["precnsim"] = 8936,  
4774 ["prnsim"] = 8936,  
4775 ["scnsim"] = 8937,  
4776 ["succnsim"] = 8937,  
4777 ["NotLeftTriangle"] = 8938,  
4778 ["nltri"] = 8938,  
4779 ["ntriangleleft"] = 8938,  
4780 ["NotRightTriangle"] = 8939,  
4781 ["nrtri"] = 8939,  
4782 ["ntriangleright"] = 8939,  
4783 ["NotLeftTriangleEqual"] = 8940,  
4784 ["nltrie"] = 8940,  
4785 ["ntrianglelefteq"] = 8940,  
4786 ["NotRightTriangleEqual"] = 8941,  
4787 ["nrtrie"] = 8941,  
4788 ["ntrianglerighteq"] = 8941,  
4789 ["vellip"] = 8942,  
4790 ["ctdot"] = 8943,  
4791 ["utdot"] = 8944,  
4792 ["dtdot"] = 8945,  
4793 ["disin"] = 8946,  
4794 ["isinsv"] = 8947,  
4795 ["isins"] = 8948,  
4796 ["isindot"] = 8949,  
4797 ["notindot"] = {8949, 824},  
4798 ["notinvc"] = 8950,  
4799 ["notinvb"] = 8951,  
4800 ["isinE"] = 8953,  
4801 ["notinE"] = {8953, 824},  
4802 ["nisd"] = 8954,

4803 ["xnis"] = 8955,  
4804 ["nis"] = 8956,  
4805 ["notnivc"] = 8957,  
4806 ["notnivb"] = 8958,  
4807 ["barwed"] = 8965,  
4808 ["barwedge"] = 8965,  
4809 ["Barwed"] = 8966,  
4810 ["doublebarwedge"] = 8966,  
4811 ["LeftCeiling"] = 8968,  
4812 ["lceil"] = 8968,  
4813 ["RightCeiling"] = 8969,  
4814 ["rceil"] = 8969,  
4815 ["LeftFloor"] = 8970,  
4816 ["lfloor"] = 8970,  
4817 ["RightFloor"] = 8971,  
4818 ["rfloor"] = 8971,  
4819 ["drcrop"] = 8972,  
4820 ["dlcrop"] = 8973,  
4821 ["urcrop"] = 8974,  
4822 ["ulcrop"] = 8975,  
4823 ["bnot"] = 8976,  
4824 ["proflines"] = 8978,  
4825 ["profsurf"] = 8979,  
4826 ["telrec"] = 8981,  
4827 ["target"] = 8982,  
4828 ["ulcorn"] = 8988,  
4829 ["ulcorner"] = 8988,  
4830 ["urcorn"] = 8989,  
4831 ["urcorner"] = 8989,  
4832 ["dlcorn"] = 8990,  
4833 ["llcorner"] = 8990,  
4834 ["drcorn"] = 8991,  
4835 ["lrcorn"] = 8991,  
4836 ["frown"] = 8994,  
4837 ["sfrown"] = 8994,  
4838 ["smile"] = 8995,  
4839 ["ssmile"] = 8995,  
4840 ["cylcty"] = 9005,  
4841 ["profalar"] = 9006,  
4842 ["topbot"] = 9014,  
4843 ["ovbar"] = 9021,  
4844 ["solbar"] = 9023,  
4845 ["angzarr"] = 9084,  
4846 ["lmoust"] = 9136,  
4847 ["lmoustache"] = 9136,  
4848 ["rmoust"] = 9137,  
4849 ["rmoustache"] = 9137,

4850 ["OverBracket"] = 9140,  
 4851 ["tbrk"] = 9140,  
 4852 ["UnderBracket"] = 9141,  
 4853 ["bbrk"] = 9141,  
 4854 ["bbrktbrk"] = 9142,  
 4855 ["OverParenthesis"] = 9180,  
 4856 ["UnderParenthesis"] = 9181,  
 4857 ["OverBrace"] = 9182,  
 4858 ["UnderBrace"] = 9183,  
 4859 ["trpezium"] = 9186,  
 4860 ["elinters"] = 9191,  
 4861 ["blank"] = 9251,  
 4862 ["circledS"] = 9416,  
 4863 ["oS"] = 9416,  
 4864 ["HorizontalLine"] = 9472,  
 4865 ["boxh"] = 9472,  
 4866 ["boxv"] = 9474,  
 4867 ["boxdr"] = 9484,  
 4868 ["boxdl"] = 9488,  
 4869 ["boxur"] = 9492,  
 4870 ["boxul"] = 9496,  
 4871 ["boxvr"] = 9500,  
 4872 ["boxvl"] = 9508,  
 4873 ["boxhd"] = 9516,  
 4874 ["boxhu"] = 9524,  
 4875 ["boxvh"] = 9532,  
 4876 ["boxH"] = 9552,  
 4877 ["boxV"] = 9553,  
 4878 ["boxdR"] = 9554,  
 4879 ["boxDr"] = 9555,  
 4880 ["boxDR"] = 9556,  
 4881 ["boxdL"] = 9557,  
 4882 ["boxDL"] = 9558,  
 4883 ["boxDL"] = 9559,  
 4884 ["boxuR"] = 9560,  
 4885 ["boxUr"] = 9561,  
 4886 ["boxUR"] = 9562,  
 4887 ["boxuL"] = 9563,  
 4888 ["boxUL"] = 9564,  
 4889 ["boxUL"] = 9565,  
 4890 ["boxvR"] = 9566,  
 4891 ["boxVr"] = 9567,  
 4892 ["boxVR"] = 9568,  
 4893 ["boxvL"] = 9569,  
 4894 ["boxVl"] = 9570,  
 4895 ["boxVL"] = 9571,  
 4896 ["boxHd"] = 9572,

4897 ["boxhD"] = 9573,  
 4898 ["boxHD"] = 9574,  
 4899 ["boxHu"] = 9575,  
 4900 ["boxhU"] = 9576,  
 4901 ["boxHU"] = 9577,  
 4902 ["boxvH"] = 9578,  
 4903 ["boxVh"] = 9579,  
 4904 ["boxVH"] = 9580,  
 4905 ["uhblk"] = 9600,  
 4906 ["lhblk"] = 9604,  
 4907 ["block"] = 9608,  
 4908 ["blk14"] = 9617,  
 4909 ["blk12"] = 9618,  
 4910 ["blk34"] = 9619,  
 4911 ["Square"] = 9633,  
 4912 ["squ"] = 9633,  
 4913 ["square"] = 9633,  
 4914 ["FilledVerySmallSquare"] = 9642,  
 4915 ["blacksquare"] = 9642,  
 4916 ["squarf"] = 9642,  
 4917 ["squf"] = 9642,  
 4918 ["EmptyVerySmallSquare"] = 9643,  
 4919 ["rect"] = 9645,  
 4920 ["marker"] = 9646,  
 4921 ["fltns"] = 9649,  
 4922 ["bigtriangleup"] = 9651,  
 4923 ["xutri"] = 9651,  
 4924 ["blacktriangle"] = 9652,  
 4925 ["utrif"] = 9652,  
 4926 ["triangle"] = 9653,  
 4927 ["utri"] = 9653,  
 4928 ["blacktriangleright"] = 9656,  
 4929 ["rtrif"] = 9656,  
 4930 ["rtri"] = 9657,  
 4931 ["triangleright"] = 9657,  
 4932 ["bigtriangledown"] = 9661,  
 4933 ["xdtri"] = 9661,  
 4934 ["blacktriangledown"] = 9662,  
 4935 ["dtrif"] = 9662,  
 4936 ["dtri"] = 9663,  
 4937 ["triangledown"] = 9663,  
 4938 ["blacktriangleleft"] = 9666,  
 4939 ["ltrif"] = 9666,  
 4940 ["ltri"] = 9667,  
 4941 ["triangleleft"] = 9667,  
 4942 ["loz"] = 9674,  
 4943 ["lozenge"] = 9674,



4944 ["cir"] = 9675,  
4945 ["tridot"] = 9708,  
4946 ["bigcirc"] = 9711,  
4947 ["xcirc"] = 9711,  
4948 ["ultri"] = 9720,  
4949 ["urtri"] = 9721,  
4950 ["lltri"] = 9722,  
4951 ["EmptySmallSquare"] = 9723,  
4952 ["FilledSmallSquare"] = 9724,  
4953 ["bigstar"] = 9733,  
4954 ["starf"] = 9733,  
4955 ["star"] = 9734,  
4956 ["phone"] = 9742,  
4957 ["female"] = 9792,  
4958 ["male"] = 9794,  
4959 ["spades"] = 9824,  
4960 ["spadesuit"] = 9824,  
4961 ["clubs"] = 9827,  
4962 ["clubsuit"] = 9827,  
4963 ["hearts"] = 9829,  
4964 ["heartsuit"] = 9829,  
4965 ["diamondsuit"] = 9830,  
4966 ["diams"] = 9830,  
4967 ["sung"] = 9834,  
4968 ["flat"] = 9837,  
4969 ["natur"] = 9838,  
4970 ["natural"] = 9838,  
4971 ["sharp"] = 9839,  
4972 ["check"] = 10003,  
4973 ["checkmark"] = 10003,  
4974 ["cross"] = 10007,  
4975 ["malt"] = 10016,  
4976 ["maltese"] = 10016,  
4977 ["sext"] = 10038,  
4978 ["VerticalSeparator"] = 10072,  
4979 ["lbrk"] = 10098,  
4980 ["rbrk"] = 10099,  
4981 ["bsolhsub"] = 10184,  
4982 ["suphsol"] = 10185,  
4983 ["LeftDoubleBracket"] = 10214,  
4984 ["lbrk"] = 10214,  
4985 ["RightDoubleBracket"] = 10215,  
4986 ["robrk"] = 10215,  
4987 ["LeftAngleBracket"] = 10216,  
4988 ["lang"] = 10216,  
4989 ["langle"] = 10216,  
4990 ["RightAngleBracket"] = 10217,

4991 ["rang"] = 10217,  
 4992 ["rangle"] = 10217,  
 4993 ["Lang"] = 10218,  
 4994 ["Rang"] = 10219,  
 4995 ["loang"] = 10220,  
 4996 ["roang"] = 10221,  
 4997 ["LongLeftArrow"] = 10229,  
 4998 ["longleftarrow"] = 10229,  
 4999 ["xlarr"] = 10229,  
 5000 ["LongRightArrow"] = 10230,  
 5001 ["longrightarrow"] = 10230,  
 5002 ["xrarr"] = 10230,  
 5003 ["LongLeftRightArrow"] = 10231,  
 5004 ["longleftrightarrow"] = 10231,  
 5005 ["xharr"] = 10231,  
 5006 ["DoubleLongLeftArrow"] = 10232,  
 5007 ["Longleftarrow"] = 10232,  
 5008 ["xlArr"] = 10232,  
 5009 ["DoubleLongRightArrow"] = 10233,  
 5010 ["Longrightarrow"] = 10233,  
 5011 ["xrArr"] = 10233,  
 5012 ["DoubleLongLeftRightArrow"] = 10234,  
 5013 ["Longleftrightarrow"] = 10234,  
 5014 ["xhArr"] = 10234,  
 5015 ["longmapsto"] = 10236,  
 5016 ["xmap"] = 10236,  
 5017 ["dzigrarr"] = 10239,  
 5018 ["nvlArr"] = 10498,  
 5019 ["nvrArr"] = 10499,  
 5020 ["nvHarr"] = 10500,  
 5021 ["Map"] = 10501,  
 5022 ["lbarr"] = 10508,  
 5023 ["bkarow"] = 10509,  
 5024 ["rbarr"] = 10509,  
 5025 ["lBarr"] = 10510,  
 5026 ["dbkarow"] = 10511,  
 5027 ["rBarr"] = 10511,  
 5028 ["RBarr"] = 10512,  
 5029 ["drbkarow"] = 10512,  
 5030 ["DDottrahd"] = 10513,  
 5031 ["UpArrowBar"] = 10514,  
 5032 ["DownArrowBar"] = 10515,  
 5033 ["Rarrtl"] = 10518,  
 5034 ["latail"] = 10521,  
 5035 ["ratail"] = 10522,  
 5036 ["lAtail"] = 10523,  
 5037 ["rAtail"] = 10524,

5038 ["larrfs"] = 10525,  
5039 ["rarrfs"] = 10526,  
5040 ["larrbfs"] = 10527,  
5041 ["rarrbfs"] = 10528,  
5042 ["nwarhk"] = 10531,  
5043 ["nearhk"] = 10532,  
5044 ["hksearow"] = 10533,  
5045 ["searhk"] = 10533,  
5046 ["hkswarow"] = 10534,  
5047 ["swarhk"] = 10534,  
5048 ["nwnear"] = 10535,  
5049 ["nesear"] = 10536,  
5050 ["toea"] = 10536,  
5051 ["seswar"] = 10537,  
5052 ["tosa"] = 10537,  
5053 ["swnwar"] = 10538,  
5054 ["nrarrc"] = {10547, 824},  
5055 ["rarrc"] = 10547,  
5056 ["cudarr"] = 10549,  
5057 ["ldca"] = 10550,  
5058 ["rdca"] = 10551,  
5059 ["cudarrl"] = 10552,  
5060 ["larrpl"] = 10553,  
5061 ["curarrm"] = 10556,  
5062 ["cularrp"] = 10557,  
5063 ["rarrpl"] = 10565,  
5064 ["harrcir"] = 10568,  
5065 ["Uarrocir"] = 10569,  
5066 ["lurdshar"] = 10570,  
5067 ["ldrushar"] = 10571,  
5068 ["LeftRightVector"] = 10574,  
5069 ["RightUpDownVector"] = 10575,  
5070 ["DownLeftRightVector"] = 10576,  
5071 ["LeftUpDownVector"] = 10577,  
5072 ["LeftVectorBar"] = 10578,  
5073 ["RightVectorBar"] = 10579,  
5074 ["RightUpVectorBar"] = 10580,  
5075 ["RightDownVectorBar"] = 10581,  
5076 ["DownLeftVectorBar"] = 10582,  
5077 ["DownRightVectorBar"] = 10583,  
5078 ["LeftUpVectorBar"] = 10584,  
5079 ["LeftDownVectorBar"] = 10585,  
5080 ["LeftTeeVector"] = 10586,  
5081 ["RightTeeVector"] = 10587,  
5082 ["RightUpTeeVector"] = 10588,  
5083 ["RightDownTeeVector"] = 10589,  
5084 ["DownLeftTeeVector"] = 10590,

5085 ["DownRightTeeVector"] = 10591,  
5086 ["LeftUpTeeVector"] = 10592,  
5087 ["LeftDownTeeVector"] = 10593,  
5088 ["lHar"] = 10594,  
5089 ["uHar"] = 10595,  
5090 ["rHar"] = 10596,  
5091 ["dHar"] = 10597,  
5092 ["luruhar"] = 10598,  
5093 ["ldrdhar"] = 10599,  
5094 ["ruluhar"] = 10600,  
5095 ["rdldhar"] = 10601,  
5096 ["lharul"] = 10602,  
5097 ["llhard"] = 10603,  
5098 ["rharul"] = 10604,  
5099 ["lrhard"] = 10605,  
5100 ["UpEquilibrium"] = 10606,  
5101 ["udhar"] = 10606,  
5102 ["ReverseUpEquilibrium"] = 10607,  
5103 ["duhar"] = 10607,  
5104 ["RoundImplies"] = 10608,  
5105 ["erarr"] = 10609,  
5106 ["simrarr"] = 10610,  
5107 ["larrsim"] = 10611,  
5108 ["rarrsim"] = 10612,  
5109 ["rarrap"] = 10613,  
5110 ["ltlarr"] = 10614,  
5111 ["gtrarr"] = 10616,  
5112 ["subrarr"] = 10617,  
5113 ["suplarr"] = 10619,  
5114 ["lfisht"] = 10620,  
5115 ["rfisht"] = 10621,  
5116 ["ufisht"] = 10622,  
5117 ["dfisht"] = 10623,  
5118 ["lopar"] = 10629,  
5119 ["ropar"] = 10630,  
5120 ["lbrke"] = 10635,  
5121 ["rbrke"] = 10636,  
5122 ["lbrkslu"] = 10637,  
5123 ["rbrksld"] = 10638,  
5124 ["lbrksld"] = 10639,  
5125 ["rbrkslu"] = 10640,  
5126 ["langd"] = 10641,  
5127 ["rangd"] = 10642,  
5128 ["lparlt"] = 10643,  
5129 ["rpargt"] = 10644,  
5130 ["gtlPar"] = 10645,  
5131 ["ltrPar"] = 10646,

5132 ["vzigzag"] = 10650,  
5133 ["vangrt"] = 10652,  
5134 ["angrtvbd"] = 10653,  
5135 ["ange"] = 10660,  
5136 ["range"] = 10661,  
5137 ["dwangle"] = 10662,  
5138 ["uwangle"] = 10663,  
5139 ["angmsdaa"] = 10664,  
5140 ["angmsdab"] = 10665,  
5141 ["angmsdac"] = 10666,  
5142 ["angmsdad"] = 10667,  
5143 ["angmsdae"] = 10668,  
5144 ["angmsdaf"] = 10669,  
5145 ["angmsdag"] = 10670,  
5146 ["angmsdah"] = 10671,  
5147 ["bemptyv"] = 10672,  
5148 ["demptyv"] = 10673,  
5149 ["cemptyv"] = 10674,  
5150 ["raemptyv"] = 10675,  
5151 ["laemptyv"] = 10676,  
5152 ["ohbar"] = 10677,  
5153 ["omid"] = 10678,  
5154 ["opar"] = 10679,  
5155 ["operp"] = 10681,  
5156 ["olcross"] = 10683,  
5157 ["odsold"] = 10684,  
5158 ["olcir"] = 10686,  
5159 ["ofcir"] = 10687,  
5160 ["olt"] = 10688,  
5161 ["ogt"] = 10689,  
5162 ["cirscir"] = 10690,  
5163 ["cirE"] = 10691,  
5164 ["solb"] = 10692,  
5165 ["bsolb"] = 10693,  
5166 ["boxbox"] = 10697,  
5167 ["trisb"] = 10701,  
5168 ["rtriltri"] = 10702,  
5169 ["LeftTriangleBar"] = 10703,  
5170 ["NotLeftTriangleBar"] = {10703, 824},  
5171 ["NotRightTriangleBar"] = {10704, 824},  
5172 ["RightTriangleBar"] = 10704,  
5173 ["iinfin"] = 10716,  
5174 ["infintie"] = 10717,  
5175 ["nvinfin"] = 10718,  
5176 ["eparsl"] = 10723,  
5177 ["smeparsl"] = 10724,  
5178 ["eqvparsl"] = 10725,

5179 ["blacklozenge"] = 10731,  
5180 ["lozf"] = 10731,  
5181 ["RuleDelayed"] = 10740,  
5182 ["dsol"] = 10742,  
5183 ["bigodot"] = 10752,  
5184 ["xodot"] = 10752,  
5185 ["bigoplus"] = 10753,  
5186 ["xoplus"] = 10753,  
5187 ["bigotimes"] = 10754,  
5188 ["xotime"] = 10754,  
5189 ["biguplus"] = 10756,  
5190 ["xuplus"] = 10756,  
5191 ["bigsqcup"] = 10758,  
5192 ["xsqlcup"] = 10758,  
5193 ["iiiint"] = 10764,  
5194 ["qint"] = 10764,  
5195 ["fpartint"] = 10765,  
5196 ["cirfnint"] = 10768,  
5197 ["awint"] = 10769,  
5198 ["rppolint"] = 10770,  
5199 ["scpolint"] = 10771,  
5200 ["npolint"] = 10772,  
5201 ["pointint"] = 10773,  
5202 ["quatint"] = 10774,  
5203 ["intlarhk"] = 10775,  
5204 ["pluscir"] = 10786,  
5205 ["plusacir"] = 10787,  
5206 ["simplus"] = 10788,  
5207 ["plusdu"] = 10789,  
5208 ["plussim"] = 10790,  
5209 ["plustwo"] = 10791,  
5210 ["mcomma"] = 10793,  
5211 ["minusdu"] = 10794,  
5212 ["loplus"] = 10797,  
5213 ["roplus"] = 10798,  
5214 ["Cross"] = 10799,  
5215 ["timesd"] = 10800,  
5216 ["timesbar"] = 10801,  
5217 ["smashp"] = 10803,  
5218 ["lotimes"] = 10804,  
5219 ["rotimes"] = 10805,  
5220 ["otimesas"] = 10806,  
5221 ["Otimes"] = 10807,  
5222 ["odiv"] = 10808,  
5223 ["triplus"] = 10809,  
5224 ["triminus"] = 10810,  
5225 ["tritime"] = 10811,

5226 ["intprod"] = 10812,  
5227 ["iproduct"] = 10812,  
5228 ["amalg"] = 10815,  
5229 ["capdot"] = 10816,  
5230 ["ncup"] = 10818,  
5231 ["ncap"] = 10819,  
5232 ["capand"] = 10820,  
5233 ["cupor"] = 10821,  
5234 ["cupcap"] = 10822,  
5235 ["capcup"] = 10823,  
5236 ["cupbrcap"] = 10824,  
5237 ["capbrcup"] = 10825,  
5238 ["cupcup"] = 10826,  
5239 ["capcap"] = 10827,  
5240 ["ccups"] = 10828,  
5241 ["ccaps"] = 10829,  
5242 ["ccupssm"] = 10832,  
5243 ["And"] = 10835,  
5244 ["Or"] = 10836,  
5245 ["andand"] = 10837,  
5246 ["oror"] = 10838,  
5247 ["orslope"] = 10839,  
5248 ["andslope"] = 10840,  
5249 ["andv"] = 10842,  
5250 ["orv"] = 10843,  
5251 ["andd"] = 10844,  
5252 ["ord"] = 10845,  
5253 ["wedbar"] = 10847,  
5254 ["sdote"] = 10854,  
5255 ["simdot"] = 10858,  
5256 ["congdote"] = 10861,  
5257 ["ncongdote"] = {10861, 824},  
5258 ["easter"] = 10862,  
5259 ["apacir"] = 10863,  
5260 ["apE"] = 10864,  
5261 ["napE"] = {10864, 824},  
5262 ["eplus"] = 10865,  
5263 ["pluse"] = 10866,  
5264 ["Esim"] = 10867,  
5265 ["Colone"] = 10868,  
5266 ["Equal"] = 10869,  
5267 ["ddotseq"] = 10871,  
5268 ["eDDot"] = 10871,  
5269 ["equivDD"] = 10872,  
5270 ["ltcir"] = 10873,  
5271 ["gtcir"] = 10874,  
5272 ["ltquest"] = 10875,

5273 ["gtquest"] = 10876,  
 5274 ["LessSlantEqual"] = 10877,  
 5275 ["NotLessSlantEqual"] = {10877, 824},  
 5276 ["leqslant"] = 10877,  
 5277 ["les"] = 10877,  
 5278 ["nleqslant"] = {10877, 824},  
 5279 ["nles"] = {10877, 824},  
 5280 ["GreaterSlantEqual"] = 10878,  
 5281 ["NotGreaterSlantEqual"] = {10878, 824},  
 5282 ["geqslant"] = 10878,  
 5283 ["ges"] = 10878,  
 5284 ["ngeqslant"] = {10878, 824},  
 5285 ["nges"] = {10878, 824},  
 5286 ["lesdot"] = 10879,  
 5287 ["gesdot"] = 10880,  
 5288 ["lesdoto"] = 10881,  
 5289 ["gesdoto"] = 10882,  
 5290 ["lesdotor"] = 10883,  
 5291 ["gesdoto1"] = 10884,  
 5292 ["lap"] = 10885,  
 5293 ["lessapprox"] = 10885,  
 5294 ["gap"] = 10886,  
 5295 ["gtrapprox"] = 10886,  
 5296 ["lne"] = 10887,  
 5297 ["lneq"] = 10887,  
 5298 ["gne"] = 10888,  
 5299 ["gneq"] = 10888,  
 5300 ["lnap"] = 10889,  
 5301 ["lnapprox"] = 10889,  
 5302 ["gnap"] = 10890,  
 5303 ["gnapprox"] = 10890,  
 5304 ["lEg"] = 10891,  
 5305 ["lesseqqgtr"] = 10891,  
 5306 ["gEl"] = 10892,  
 5307 ["gtreqqless"] = 10892,  
 5308 ["lsime"] = 10893,  
 5309 ["gsime"] = 10894,  
 5310 ["lsimg"] = 10895,  
 5311 ["gsiml"] = 10896,  
 5312 ["lgE"] = 10897,  
 5313 ["glE"] = 10898,  
 5314 ["lesges"] = 10899,  
 5315 ["gesles"] = 10900,  
 5316 ["els"] = 10901,  
 5317 ["eqslantless"] = 10901,  
 5318 ["egs"] = 10902,  
 5319 ["eqslantgtr"] = 10902,



```

5320 ["elsdot"] = 10903,
5321 ["egsdot"] = 10904,
5322 ["el"] = 10905,
5323 ["eg"] = 10906,
5324 ["siml"] = 10909,
5325 ["simg"] = 10910,
5326 ["simlE"] = 10911,
5327 ["simgE"] = 10912,
5328 ["LessLess"] = 10913,
5329 ["NotNestedLessLess"] = {10913, 824},
5330 ["GreaterGreater"] = 10914,
5331 ["NotNestedGreaterGreater"] = {10914, 824},
5332 ["glj"] = 10916,
5333 ["gla"] = 10917,
5334 ["ltcc"] = 10918,
5335 ["gtcc"] = 10919,
5336 ["lescc"] = 10920,
5337 ["gescc"] = 10921,
5338 ["smt"] = 10922,
5339 ["lat"] = 10923,
5340 ["smtE"] = 10924,
5341 ["smtes"] = {10924, 65024},
5342 ["late"] = 10925,
5343 ["lates"] = {10925, 65024},
5344 ["bumpE"] = 10926,
5345 ["NotPrecedesEqual"] = {10927, 824},
5346 ["PrecedesEqual"] = 10927,
5347 ["npre"] = {10927, 824},
5348 ["npreceq"] = {10927, 824},
5349 ["pre"] = 10927,
5350 ["preceq"] = 10927,
5351 ["NotSucceedsEqual"] = {10928, 824},
5352 ["SucceedsEqual"] = 10928,
5353 ["nsce"] = {10928, 824},
5354 ["nsucceq"] = {10928, 824},
5355 ["sce"] = 10928,
5356 ["succeq"] = 10928,
5357 ["prE"] = 10931,
5358 ["scE"] = 10932,
5359 ["precneqq"] = 10933,
5360 ["prnE"] = 10933,
5361 ["scnE"] = 10934,
5362 ["succneqq"] = 10934,
5363 ["prap"] = 10935,
5364 ["precapprox"] = 10935,
5365 ["scap"] = 10936,
5366 ["succapprox"] = 10936,

```

5367 ["precnapprox"] = 10937,  
5368 ["prnap"] = 10937,  
5369 ["scnap"] = 10938,  
5370 ["succnapprox"] = 10938,  
5371 ["Pr"] = 10939,  
5372 ["Sc"] = 10940,  
5373 ["subdot"] = 10941,  
5374 ["supdot"] = 10942,  
5375 ["subplus"] = 10943,  
5376 ["supplus"] = 10944,  
5377 ["submult"] = 10945,  
5378 ["supmult"] = 10946,  
5379 ["subedot"] = 10947,  
5380 ["supedot"] = 10948,  
5381 ["nsubE"] = {10949, 824},  
5382 ["nsubseteqq"] = {10949, 824},  
5383 ["subE"] = 10949,  
5384 ["subseteqq"] = 10949,  
5385 ["nsupE"] = {10950, 824},  
5386 ["nsupseteqq"] = {10950, 824},  
5387 ["supE"] = 10950,  
5388 ["supseteqq"] = 10950,  
5389 ["subsim"] = 10951,  
5390 ["supsim"] = 10952,  
5391 ["subnE"] = 10955,  
5392 ["subsetneqq"] = 10955,  
5393 ["varsubsetneqq"] = {10955, 65024},  
5394 ["vsubnE"] = {10955, 65024},  
5395 ["supnE"] = 10956,  
5396 ["supsetneqq"] = 10956,  
5397 ["varsupsetneqq"] = {10956, 65024},  
5398 ["vsupnE"] = {10956, 65024},  
5399 ["csub"] = 10959,  
5400 ["csup"] = 10960,  
5401 ["csube"] = 10961,  
5402 ["csupe"] = 10962,  
5403 ["subsup"] = 10963,  
5404 ["supsub"] = 10964,  
5405 ["subsub"] = 10965,  
5406 ["supsup"] = 10966,  
5407 ["suphsub"] = 10967,  
5408 ["supdsub"] = 10968,  
5409 ["forkv"] = 10969,  
5410 ["topfork"] = 10970,  
5411 ["mlcp"] = 10971,  
5412 ["Dashv"] = 10980,  
5413 ["DoubleLeftTee"] = 10980,

5414 ["Vdashl"] = 10982,  
5415 ["Barv"] = 10983,  
5416 ["vBar"] = 10984,  
5417 ["vBarv"] = 10985,  
5418 ["Vbar"] = 10987,  
5419 ["Not"] = 10988,  
5420 ["bNot"] = 10989,  
5421 ["rnmid"] = 10990,  
5422 ["cirmid"] = 10991,  
5423 ["midcir"] = 10992,  
5424 ["topcir"] = 10993,  
5425 ["nhpar"] = 10994,  
5426 ["parsim"] = 10995,  
5427 ["nparsl"] = {11005, 8421},  
5428 ["parsl"] = 11005,  
5429 ["fflig"] = 64256,  
5430 ["filig"] = 64257,  
5431 ["fllig"] = 64258,  
5432 ["ffilig"] = 64259,  
5433 ["ffllig"] = 64260,  
5434 ["Ascr"] = 119964,  
5435 ["Cscr"] = 119966,  
5436 ["Dscr"] = 119967,  
5437 ["Gscr"] = 119970,  
5438 ["Jscr"] = 119973,  
5439 ["Kscr"] = 119974,  
5440 ["Nscr"] = 119977,  
5441 ["Oscr"] = 119978,  
5442 ["Pscr"] = 119979,  
5443 ["Qscr"] = 119980,  
5444 ["Sscr"] = 119982,  
5445 ["Tscr"] = 119983,  
5446 ["Uscr"] = 119984,  
5447 ["Vscr"] = 119985,  
5448 ["Wscr"] = 119986,  
5449 ["Xscr"] = 119987,  
5450 ["Yscr"] = 119988,  
5451 ["Zscr"] = 119989,  
5452 ["ascr"] = 119990,  
5453 ["bscr"] = 119991,  
5454 ["cscr"] = 119992,  
5455 ["dscr"] = 119993,  
5456 ["fscr"] = 119995,  
5457 ["hscr"] = 119997,  
5458 ["iscr"] = 119998,  
5459 ["jscr"] = 119999,  
5460 ["kscr"] = 120000,

5461 ["lscr"] = 120001,  
5462 ["mscr"] = 120002,  
5463 ["nscr"] = 120003,  
5464 ["pscr"] = 120005,  
5465 ["qscr"] = 120006,  
5466 ["rscr"] = 120007,  
5467 ["sscr"] = 120008,  
5468 ["tscr"] = 120009,  
5469 ["uscr"] = 120010,  
5470 ["vscr"] = 120011,  
5471 ["wscr"] = 120012,  
5472 ["xscr"] = 120013,  
5473 ["yscr"] = 120014,  
5474 ["zscr"] = 120015,  
5475 ["Afr"] = 120068,  
5476 ["Bfr"] = 120069,  
5477 ["Dfr"] = 120071,  
5478 ["Efr"] = 120072,  
5479 ["Ffr"] = 120073,  
5480 ["Gfr"] = 120074,  
5481 ["Jfr"] = 120077,  
5482 ["Kfr"] = 120078,  
5483 ["Lfr"] = 120079,  
5484 ["Mfr"] = 120080,  
5485 ["Nfr"] = 120081,  
5486 ["Ofr"] = 120082,  
5487 ["Pfr"] = 120083,  
5488 ["Qfr"] = 120084,  
5489 ["Sfr"] = 120086,  
5490 ["Tfr"] = 120087,  
5491 ["Ufr"] = 120088,  
5492 ["Vfr"] = 120089,  
5493 ["Wfr"] = 120090,  
5494 ["Xfr"] = 120091,  
5495 ["Yfr"] = 120092,  
5496 ["afr"] = 120094,  
5497 ["bfr"] = 120095,  
5498 ["cfr"] = 120096,  
5499 ["dfr"] = 120097,  
5500 ["efr"] = 120098,  
5501 ["ffr"] = 120099,  
5502 ["gfr"] = 120100,  
5503 ["hfr"] = 120101,  
5504 ["ifr"] = 120102,  
5505 ["jfr"] = 120103,  
5506 ["kfr"] = 120104,  
5507 ["lfr"] = 120105,

5508 ["mfr"] = 120106,  
5509 ["nfr"] = 120107,  
5510 ["ofr"] = 120108,  
5511 ["pfr"] = 120109,  
5512 ["qfr"] = 120110,  
5513 ["rfr"] = 120111,  
5514 ["sfr"] = 120112,  
5515 ["tfr"] = 120113,  
5516 ["ufr"] = 120114,  
5517 ["vfr"] = 120115,  
5518 ["wfr"] = 120116,  
5519 ["xfr"] = 120117,  
5520 ["yfr"] = 120118,  
5521 ["zfr"] = 120119,  
5522 ["Aopf"] = 120120,  
5523 ["Bopf"] = 120121,  
5524 ["Dopf"] = 120123,  
5525 ["Eopf"] = 120124,  
5526 ["Fopf"] = 120125,  
5527 ["Gopf"] = 120126,  
5528 ["Iopf"] = 120128,  
5529 ["Jopf"] = 120129,  
5530 ["Kopf"] = 120130,  
5531 ["Lopf"] = 120131,  
5532 ["Mopf"] = 120132,  
5533 ["Oopf"] = 120134,  
5534 ["Sopf"] = 120138,  
5535 ["Topf"] = 120139,  
5536 ["Uopf"] = 120140,  
5537 ["Vopf"] = 120141,  
5538 ["Wopf"] = 120142,  
5539 ["Xopf"] = 120143,  
5540 ["Yopf"] = 120144,  
5541 ["aopf"] = 120146,  
5542 ["bopf"] = 120147,  
5543 ["copf"] = 120148,  
5544 ["dopf"] = 120149,  
5545 ["eopf"] = 120150,  
5546 ["fopf"] = 120151,  
5547 ["gopf"] = 120152,  
5548 ["hopf"] = 120153,  
5549 ["iopf"] = 120154,  
5550 ["jopf"] = 120155,  
5551 ["kopf"] = 120156,  
5552 ["lopf"] = 120157,  
5553 ["mopf"] = 120158,  
5554 ["nopf"] = 120159,

```

5555 ["oopf"] = 120160,
5556 ["popf"] = 120161,
5557 ["qopf"] = 120162,
5558 ["ropf"] = 120163,
5559 ["sopf"] = 120164,
5560 ["topf"] = 120165,
5561 ["uopf"] = 120166,
5562 ["vopf"] = 120167,
5563 ["wopf"] = 120168,
5564 ["xopf"] = 120169,
5565 ["yopf"] = 120170,
5566 ["zopf"] = 120171,
5567 }

```

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

```

5568 function entities.dec_entity(s)
5569   local n = tonumber(s)
5570   if n == nil then
5571     return "&#" .. s .. ";" -- fallback for unknown entities
5572   end
5573   return unicode.utf8.char(n)
5574 end

```

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

```

5575 function entities.hex_entity(s)
5576   local n = tonumber("0x"..s)
5577   if n == nil then
5578     return "&#x" .. s .. ";" -- fallback for unknown entities
5579   end
5580   return unicode.utf8.char(n)
5581 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.

```

5582 function entities.hex_entity_with_x_char(x, s)
5583   local n = tonumber("0x"..s)
5584   if n == nil then
5585     return "&#" .. x .. s .. ";" -- fallback for unknown entities
5586   end
5587   return unicode.utf8.char(n)
5588 end

```

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

```

5589 function entities.char_entity(s)

```

```

5590 local code_points = character_entities[s]
5591 if code_points == nil then
5592     return "&" .. s .. ";"
5593 end
5594 if type(code_points) ~= 'table' then
5595     code_points = {code_points}
5596 end
5597 local char_table = {}
5598 for _, code_point in ipairs(code_points) do
5599     table.insert(char_table, unicode.utf8.char(code_point))
5600 end
5601 return table.concat(char_table)
5602 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 Lunamark Lua module.

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

```
5603 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.

```
5604 function M.writer.new(options)
5605     local self = {}
```

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

```
5606     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.

```
5607     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.

```
5608 local slice_specifiers = {}
5609 for specifier in options.slice:gmatch("[~%s]+") do
5610   table.insert(slice_specifiers, specifier)
5611 end
5612
5613 if #slice_specifiers == 2 then
5614   self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
5615   local slice_begin_type = self.slice_begin:sub(1, 1)
5616   if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
5617     self.slice_begin = "^" .. self.slice_begin
5618   end
5619   local slice_end_type = self.slice_end:sub(1, 1)
5620   if slice_end_type ~= "^" and slice_end_type ~= "$" then
5621     self.slice_end = "$" .. self.slice_end
5622   end
5623 elseif #slice_specifiers == 1 then
5624   self.slice_begin = "^" .. slice_specifiers[1]
5625   self.slice_end = "$" .. slice_specifiers[1]
5626 end
5627
5628 self.slice_begin_type = self.slice_begin:sub(1, 1)
5629 self.slice_begin_identifier = self.slice_begin:sub(2) or ""
5630 self.slice_end_type = self.slice_end:sub(1, 1)
5631 self.slice_end_identifier = self.slice_end:sub(2) or ""
5632
5633 if self.slice_begin == "^" and self.slice_end ~= "^" then
5634   self.is_writing = true
5635 else
5636   self.is_writing = false
5637 end
```

Define `writer->suffix` as the suffix of the produced cache files.

```
5638 self.suffix = ".tex"
```

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

```
5639 self.space = " "
```

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

```
5640 self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
5641 function self.plain(s)
5642   return s
5643 end
```



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

```
5644 function self.paragraph(s)
5645     if not self.is_writing then return "" end
5646     return s
5647 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.

```
5648 function self.pack(name)
5649     return [[\input{]} .. name .. [{}\relax]]
5650 end
```

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

```
5651 self.interblocksep_text = "\\markdownRendererInterblockSeparator\n{"
5652 function self.interblocksep()
5653     if not self.is_writing then return "" end
5654     return self.interblocksep_text
5655 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.

```
5656 self.paragraphsep_text = "\\markdownRendererParagraphSeparator\n{"
5657 function self.paragraphsep()
5658     if not self.is_writing then return "" end
5659     return self.paragraphsep_text
5660 end
```

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

```
5661 self.undosep_text = "\\markdownRendererUndoSeparator\n{"
5662 function self.undosep()
5663     if not self.is_writing then return "" end
5664     return self.undosep_text
5665 end
```

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

```
5666 self.soft_line_break = function()
5667     if self.flatten_inlines then return "\n" end
5668     return "\\markdownRendererSoftLineBreak\n{"
5669 end
```

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

```
5670 self.hard_line_break = function()
5671     if self.flatten_inlines then return "\n" end
5672     return "\\markdownRendererHardLineBreak\n{"
5673 end
```

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

```
5674 self.ellipsis = "\\markdownRendererEllipsis{"
```

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

```
5675 function self.thematic_break()
5676   if not self.is_writing then return "" end
5677   return "\\markdownRendererThematicBreak{"
5678 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.

```
5679 self.escaped_uri_chars = {
5680   [{""] = "\\markdownRendererLeftBrace{"},
5681   ["}"] = "\\markdownRendererRightBrace{"},
5682   [{"\\"}] = "\\markdownRendererBackslash{"},
5683 }
5684 self.escaped_minimal_strings = {
5685   [{"^"}] = "\\markdownRendererCircumflex\\markdownRendererCircumflex ",
5686   [{"☒"}] = "\\markdownRendererTickedBox{"},
5687   [{"◻"}] = "\\markdownRendererHalfTickedBox{"},
5688   [{"□"}] = "\\markdownRendererUntickedBox{"},
5689   [entities.hex_entity('FFFD')] = "\\markdownRendererReplacementCharacter{"},
5690 }
```

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

```
5691 self.escaped_strings = util.table_copy(self.escaped_minimal_strings)
5692 self.escaped_strings[entities.hex_entity('00A0')] = self.nbsp
```

Define a table `writer->escaped_chars` containing the mapping from special plain TeX characters (including the active pipe character (`|`) of ConTeXt) that need to be escaped in typeset content.

```
5693 self.escaped_chars = {
5694   [{"{"] = "\\markdownRendererLeftBrace{"},
5695   [{"}"] = "\\markdownRendererRightBrace{"},
5696   [{"%"}] = "\\markdownRendererPercentSign{"},
5697   [{"\\"}] = "\\markdownRendererBackslash{"},
5698   [{"#"}] = "\\markdownRendererHash{"},
5699   [{"$"}] = "\\markdownRendererDollarSign{"},
5700   [{"&"}] = "\\markdownRendererAmpersand{"},
5701   [{"_"}] = "\\markdownRendererUnderscore{"},
5702   [{"^"}] = "\\markdownRendererCircumflex{"},
5703   [{"~"}] = "\\markdownRendererTilde{"},
5704   [{"|"}] = "\\markdownRendererPipe{"},
5705   [entities.hex_entity('0000')] = "\\markdownRendererReplacementCharacter{"},
5706 }
```

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

```
5707 local function create_escaper(char_escapes, string_escapes)
5708     local escape = util.escaper(char_escapes, string_escapes)
5709     return function(s)
5710         if self.flatten_inlines then return s end
5711         return escape(s)
5712     end
5713 end
5714 local escape_typographic_text = create_escaper(
5715     self.escaped_chars, self.escaped_strings)
5716 local escape_programmatic_text = create_escaper(
5717     self.escaped_uri_chars, self.escaped_minimal_strings)
5718 local escape_minimal = create_escaper(
5719     {}, self.escaped_minimal_strings)
```

Define the following semantic aliases for the escaper functions:

- `writer->escape` transforms a text string that should always be made printable.
- `writer->string` transforms a text string that should be made printable only when the `hybrid` Lua option is disabled. When `hybrid` is enabled, the text string should be kept as-is.
- `writer->math` transforms a math span.
- `writer->identifier` transforms an input programmatic identifier.
- `writer->uri` transforms an input URI.
- `writer->infostring` transforms a fence code infostring.

```
5720 self.escape = escape_typographic_text
5721 self.math = escape_minimal
5722 if options.hybrid then
5723     self.identifier = escape_minimal
5724     self.string = escape_minimal
5725     self.uri = escape_minimal
5726     self.infostring = escape_minimal
5727 else
5728     self.identifier = escape_programmatic_text
5729     self.string = escape_typographic_text
5730     self.uri = escape_programmatic_text
5731     self.infostring = escape_programmatic_text
5732 end
```

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

```
5733 function self.code(s, attributes)
5734     if self.flatten_inlines then return s end
```

```

5735     local buf = {}
5736     if attributes ~= nil then
5737         table.insert(buf,
5738             "\\markdownRendererCodeSpanAttributeContextBegin\n")
5739         table.insert(buf, self.attributes(attributes))
5740     end
5741     table.insert(buf,
5742         {"\\markdownRendererCodeSpan{" , self.escape(s), "}"})
5743     if attributes ~= nil then
5744         table.insert(buf,
5745             "\\markdownRendererCodeSpanAttributeContextEnd{")
5746     end
5747     return buf
5748 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.

```

5749     function self.link(lab, src, tit, attributes)
5750         if self.flatten_inlines then return lab end
5751         local buf = {}
5752         if attributes ~= nil then
5753             table.insert(buf,
5754                 "\\markdownRendererLinkAttributeContextBegin\n")
5755             table.insert(buf, self.attributes(attributes))
5756         end
5757         table.insert(buf, {"\\markdownRendererLink{" ,lab,"} ,
5758             {"",self.escape(src),"} ,
5759             {"",self.uri(src),"} ,
5760             {"",self.string(tit or ""),"}"}))
5761         if attributes ~= nil then
5762             table.insert(buf,
5763                 "\\markdownRendererLinkAttributeContextEnd{")
5764         end
5765         return buf
5766     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.

```

5767     function self.image(lab, src, tit, attributes)
5768         if self.flatten_inlines then return lab end
5769         local buf = {}
5770         if attributes ~= nil then
5771             table.insert(buf,
5772                 "\\markdownRendererImageAttributeContextBegin\n")
5773             table.insert(buf, self.attributes(attributes))

```

```

5774     end
5775     table.insert(buf, {"\\markdownRendererImage{" ,lab,"} ",
5776                     "{" ,self.string(src),"} ",
5777                     "{" ,self.uri(src),"} ",
5778                     "{" ,self.string(tit or ""),"}"} )
5779     if attributes ~= nil then
5780         table.insert(buf,
5781                     "\\markdownRendererImageAttributeContextEnd{ }")
5782     end
5783     return buf
5784 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.

```

5785 function self.bulletlist(items,tight)
5786     if not self.is_writing then return "" end
5787     local buffer = {}
5788     for _,item in ipairs(items) do
5789         if item ~= "" then
5790             buffer[#buffer + 1] = self.bulletitem(item)
5791         end
5792     end
5793     local contents = util.intersperse(buffer,"\n")
5794     if tight and options.tightLists then
5795         return {"\\markdownRendererUlBeginTight\n",contents,
5796             "\n\\markdownRendererUlEndTight "}
5797     else
5798         return {"\\markdownRendererUlBegin\n",contents,
5799             "\n\\markdownRendererUlEnd "}
5800     end
5801 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.

```

5802 function self.bulletitem(s)
5803     return {"\\markdownRendererUlItem ",s,
5804         "\\markdownRendererUlItemEnd "}
5805 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.

```

5806 function self.orderedlist(items,tight,startnum)
5807     if not self.is_writing then return "" end
5808     local buffer = {}
5809     local num = startnum

```

```

5810     for _,item in ipairs(items) do
5811         if item ~= "" then
5812             buffer[#buffer + 1] = self.ordereditem(item,num)
5813         end
5814         if num ~= nil and item ~= "" then
5815             num = num + 1
5816         end
5817     end
5818     local contents = util.intersperse(buffer,"\n")
5819     if tight and options.tightLists then
5820         return {"\\markdownRenderer01BeginTight\n",contents,
5821             "\\n\\markdownRenderer01EndTight "}
5822     else
5823         return {"\\markdownRenderer01Begin\n",contents,
5824             "\\n\\markdownRenderer01End "}
5825     end
5826 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.

```

5827 function self.ordereditem(s,num)
5828     if num ~= nil then
5829         return {"\\markdownRenderer01ItemWithNumber{" ,num,"} ",s,
5830             "\\markdownRenderer01ItemEnd "}
5831     else
5832         return {"\\markdownRenderer01Item ",s,
5833             "\\markdownRenderer01ItemEnd "}
5834     end
5835 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.

```

5836 function self.inline_html_comment(contents)
5837     if self.flatten_inlines then return contents end
5838     return {"\\markdownRendererInlineHtmlComment{" ,contents,"}"}
5839 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.

```

5840 function self.inline_html_tag(contents)
5841     if self.flatten_inlines then return contents end
5842     return {"\\markdownRendererInlineHtmlTag{" ,self.string(contents),"}"}
5843 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.

```
5844 function self.block_html_element(s)
5845   if not self.is_writing then return "" end
5846   local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
5847   return {"\\markdownRendererInputBlockHtmlElement{" ,name,"}"}
5848 end
```

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

```
5849 function self.emphasis(s)
5850   if self.flatten_inlines then return s end
5851   return {"\\markdownRendererEmphasis{" ,s,"}"}
5852 end
```

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

```
5853 function self.checkbox(f)
5854   if f == 1.0 then
5855     return "☒ "
5856   elseif f == 0.0 then
5857     return "☐ "
5858   else
5859     return "◻ "
5860   end
5861 end
```

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

```
5862 function self.strong(s)
5863   if self.flatten_inlines then return s end
5864   return {"\\markdownRendererStrongEmphasis{" ,s,"}"}
5865 end
```

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

```
5866 function self.blockquote(s)
5867   if not self.is_writing then return "" end
5868   return {"\\markdownRendererBlockQuoteBegin\n" ,s,
5869     "\\markdownRendererBlockQuoteEnd "}
5870 end
```

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

```
5871 function self.verbatim(s)
5872   if not self.is_writing then return "" end
5873   s = s:gsub("\\n$", "")
```

```

5874     local name = util.cache_verbatim(options.cacheDir, s)
5875     return {"\\markdownRendererInputVerbatim{" ,name,""}"}
5876 end

```

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

```

5877 function self.document(d)
5878     local buf = {"\\markdownRendererDocumentBegin\n", d}
5879
5880     -- pop all attributes
5881     table.insert(buf, self.pop_attributes())
5882
5883     table.insert(buf, "\\markdownRendererDocumentEnd")
5884
5885     return buf
5886 end

```

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

```

5887 local seen_identifiers = {}
5888 local key_value_regex = "([~= ]+)%s*=%s*(.*)"
5889 local function normalize_attributes(attributes, auto_identifiers)
5890     -- normalize attributes
5891     local normalized_attributes = {}
5892     local has_explicit_identifiers = false
5893     local key, value
5894     for _, attribute in ipairs(attributes or {}) do
5895         if attribute:sub(1, 1) == "#" then
5896             table.insert(normalized_attributes, attribute)
5897             has_explicit_identifiers = true
5898             seen_identifiers[attribute:sub(2)] = true
5899         elseif attribute:sub(1, 1) == "." then
5900             table.insert(normalized_attributes, attribute)
5901         else
5902             key, value = attribute:match(key_value_regex)
5903             if key:lower() == "id" then
5904                 table.insert(normalized_attributes, "#" .. value)
5905             elseif key:lower() == "class" then
5906                 local classes = {}
5907                 for class in value:gmatch("%S+") do
5908                     table.insert(classes, class)
5909                 end
5910                 table.sort(classes)
5911                 for _, class in ipairs(classes) do
5912                     table.insert(normalized_attributes, "." .. class)
5913                 end
5914             else
5915                 table.insert(normalized_attributes, attribute)

```



```

5916         end
5917     end
5918 end
5919
5920 -- if no explicit identifiers exist, add auto identifiers
5921 if not has_explicit_identifiers and auto_identifiers ~= nil then
5922     local seen_auto_identifiers = {}
5923     for _, auto_identifier in ipairs(auto_identifiers) do
5924         if seen_auto_identifiers[auto_identifier] == nil then
5925             seen_auto_identifiers[auto_identifier] = true
5926             if seen_identifiers[auto_identifier] == nil then
5927                 seen_identifiers[auto_identifier] = true
5928                 table.insert(normalized_attributes,
5929                     "#" .. auto_identifier)
5930             else
5931                 local auto_identifier_number = 1
5932                 while true do
5933                     local numbered_auto_identifier = auto_identifier .. "-"
5934                                             .. auto_identifier_number
5935                     if seen_identifiers[numbered_auto_identifier] == nil then
5936                         seen_identifiers[numbered_auto_identifier] = true
5937                         table.insert(normalized_attributes,
5938                             "#" .. numbered_auto_identifier)
5939                     break
5940                 end
5941                 auto_identifier_number = auto_identifier_number + 1
5942             end
5943         end
5944     end
5945 end
5946 end
5947
5948 -- sort and deduplicate normalized attributes
5949 table.sort(normalized_attributes)
5950 local seen_normalized_attributes = {}
5951 local deduplicated_normalized_attributes = {}
5952 for _, attribute in ipairs(normalized_attributes) do
5953     if seen_normalized_attributes[attribute] == nil then
5954         seen_normalized_attributes[attribute] = true
5955         table.insert(deduplicated_normalized_attributes, attribute)
5956     end
5957 end
5958
5959 return deduplicated_normalized_attributes
5960 end
5961
5962 function self.attributes(attributes, should_normalize_attributes)

```

```

5963     local normalized_attributes
5964     if should_normalize_attributes == false then
5965         normalized_attributes = attributes
5966     else
5967         normalized_attributes = normalize_attributes(attributes)
5968     end
5969
5970     local buf = {}
5971     local key, value
5972     for _, attribute in ipairs(normalized_attributes) do
5973         if attribute:sub(1, 1) == "#" then
5974             table.insert(buf, {"\\markdownRendererAttributeIdentifier{" ,
5975                                 attribute:sub(2), "}"}))
5976         elseif attribute:sub(1, 1) == "." then
5977             table.insert(buf, {"\\markdownRendererAttributeName{" ,
5978                                 attribute:sub(2), "}"}))
5979         else
5980             key, value = attribute:match(key_value_regex)
5981             table.insert(buf, {"\\markdownRendererAttributeKeyValue{" ,
5982                                 key, "}{" , value, "}"}))
5983         end
5984     end
5985
5986     return buf
5987 end

```

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

```

5988     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`.

```

5989     self.attribute_type_levels = {}
5990     setmetatable(self.attribute_type_levels,
5991                 { __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`.

```

5992     local function apply_attributes()
5993         local buf = {}
5994         for i = 1, #self.active_attributes do
5995             local start_output = self.active_attributes[i][3]
5996             if start_output ~= nil then
5997                 table.insert(buf, start_output)
5998             end
5999         end
6000     return buf

```

```

6001 end
6002
6003 local function tear_down_attributes()
6004     local buf = {}
6005     for i = #self.active_attributes, 1, -1 do
6006         local end_output = self.active_attributes[i][4]
6007         if end_output ~= nil then
6008             table.insert(buf, end_output)
6009         end
6010     end
6011     return buf
6012 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.

```

6013 function self.push_attributes(attribute_type, attributes,
6014                             start_output, end_output)
6015     local attribute_type_level = self.attribute_type_levels[attribute_type]
6016     self.attribute_type_levels[attribute_type] = attribute_type_level + 1
6017
6018     -- index attributes in a hash table for easy lookup
6019     attributes = attributes or {}
6020     for i = 1, #attributes do
6021         attributes[attributes[i]] = true
6022     end
6023
6024     local buf = {}
6025     -- handle slicing
6026     if attributes["#" .. self.slice_end_identifier] ~= nil and
6027        self.slice_end_type == "^" then
6028         if self.is_writing then
6029             table.insert(buf, self.undosep())
6030             table.insert(buf, tear_down_attributes())
6031         end
6032         self.is_writing = false
6033     end
6034     if attributes["#" .. self.slice_begin_identifier] ~= nil and
6035        self.slice_begin_type == "^" then
6036         table.insert(buf, apply_attributes())
6037         self.is_writing = true
6038     end
6039     if self.is_writing and start_output ~= nil then
6040         table.insert(buf, start_output)
6041     end

```

```

6042     table.insert(self.active_attributes,
6043                   {attribute_type, attributes,
6044                     start_output, end_output})
6045     return buf
6046 end
6047

```

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`).

```

6048 function self.pop_attributes(attribute_type)
6049     local buf = {}
6050     -- pop attributes until we find attributes of correct type
6051     -- or until no attributes remain
6052     local current_attribute_type = false
6053     while current_attribute_type ~= attribute_type and
6054           #self.active_attributes > 0 do
6055         local attributes, _, end_output
6056         current_attribute_type, attributes, _, end_output = table.unpack(
6057             self.active_attributes[#self.active_attributes])
6058         local attribute_type_level = self.attribute_type_levels[current_attribute_type]
6059         self.attribute_type_levels[current_attribute_type] = attribute_type_level - 1
6060         if self.is_writing and end_output ~= nil then
6061             table.insert(buf, end_output)
6062         end
6063         table.remove(self.active_attributes, #self.active_attributes)
6064         -- handle slicing
6065         if attributes["#" .. self.slice_end_identifiers] ~= nil
6066           and self.slice_end_type == "$" then
6067             if self.is_writing then
6068                 table.insert(buf, self.undosep())
6069                 table.insert(buf, tear_down_attributes())
6070             end
6071             self.is_writing = false
6072         end
6073         if attributes["#" .. self.slice_begin_identifiers] ~= nil and
6074           self.slice_begin_type == "$" then
6075             self.is_writing = true
6076             table.insert(buf, apply_attributes())
6077         end
6078     end
6079     return buf
6080 end

```

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

```
6081 local function create_auto_identifier(s)
6082   local buffer = {}
6083   local prev_space = false
6084   local letter_found = false
6085   local normalized_s = s
6086   if not options.unicodeNormalization or options.unicodeNormalizationForm ~= "nfc"
6087     normalized_s = uni_algos.normalize.NFC(normalized_s)
6088   end
6089
6090   for _, code in utf8.codes(normalized_s) do
6091     local char = utf8.char(code)
6092
6093     -- Remove everything up to the first letter.
6094     if not letter_found then
6095       local is_letter = unicode.utf8.match(char, "%a")
6096       if is_letter then
6097         letter_found = true
6098       else
6099         goto continue
6100       end
6101     end
6102
6103     -- Remove all non-alphanumeric characters, except underscores, hyphens, and per
6104     if not unicode.utf8.match(char, "[%w_%-%.%s]") then
6105       goto continue
6106     end
6107
6108     -- Replace all spaces and newlines with hyphens.
6109     if unicode.utf8.match(char, "[%s\n]") then
6110       char = "-"
6111       if prev_space then
6112         goto continue
6113       else
6114         prev_space = true
6115       end
6116     else
6117       -- Convert all alphabetic characters to lowercase.
6118       char = unicode.utf8.lower(char)
6119       prev_space = false
6120     end
6121
6122     table.insert(buffer, char)
6123
6124     ::continue::
6125   end
6126
```

```

6127     if prev_space then
6128         table.remove(buffer)
6129     end
6130
6131     local identifier = #buffer == 0 and "section" or table.concat(buffer, "")
6132     return identifier
6133 end

```

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

```

6134 local function create_gfm_auto_identifier(s)
6135     local buffer = {}
6136     local prev_space = false
6137     local letter_found = false
6138     local normalized_s = s
6139     if not options.unicodeNormalization or options.unicodeNormalizationForm ~= "nfc"
6140         normalized_s = uni_algos.normalize.NFC(normalized_s)
6141     end
6142
6143     for _, code in utf8.codes(normalized_s) do
6144         local char = utf8.char(code)
6145
6146         -- Remove everything up to the first non-space.
6147         if not letter_found then
6148             local is_letter = unicode.utf8.match(char, "%S")
6149             if is_letter then
6150                 letter_found = true
6151             else
6152                 goto continue
6153             end
6154         end
6155
6156         -- Remove all non-alphanumeric characters, except underscores and hyphens.
6157         if not unicode.utf8.match(char, "[%w_-%s]") then
6158             prev_space = false
6159             goto continue
6160         end
6161
6162         -- Replace all spaces and newlines with hyphens.
6163         if unicode.utf8.match(char, "[%s\n]") then
6164             char = "-"
6165             if prev_space then
6166                 goto continue
6167             else
6168                 prev_space = true
6169             end
6170         else

```

```

6171     -- Convert all alphabetic characters to lowercase.
6172     char = unicode.utf8.lower(char)
6173     prev_space = false
6174     end
6175
6176     table.insert(buffer, char)
6177
6178     ::continue::
6179     end
6180
6181     if prev_space then
6182         table.remove(buffer)
6183     end
6184
6185     local identifier = #buffer == 0 and "section" or table.concat(buffer, "")
6186     return identifier
6187     end

```

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

```

6188     self.secbegin_text = "\\markdownRendererSectionBegin\n"
6189     self.secend_text = "\\n\\markdownRendererSectionEnd "
6190     function self.heading(s, level, attributes)
6191         local buf = {}
6192         local flat_text, inlines = table.unpack(s)
6193
6194         -- push empty attributes for implied sections
6195         while self.attribute_type_levels["heading"] < level - 1 do
6196             table.insert(buf,
6197                 self.push_attributes("heading",
6198                     nil,
6199                     self.secbegin_text,
6200                     self.secend_text))
6201         end
6202
6203         -- pop attributes for sections that have ended
6204         while self.attribute_type_levels["heading"] >= level do
6205             table.insert(buf, self.pop_attributes("heading"))
6206         end
6207
6208         -- construct attributes for the new section
6209         local auto_identifiers = {}
6210         if self.options.autoIdentifiers then
6211             table.insert(auto_identifiers, create_auto_identifier(flat_text))
6212         end
6213         if self.options.gfmAutoIdentifiers then
6214             table.insert(auto_identifiers, create_gfm_auto_identifier(flat_text))

```

```

6215     end
6216     local normalized_attributes = normalize_attributes(attributes, auto_identifiers)
6217
6218     -- push attributes for the new section
6219     local start_output = {}
6220     local end_output = {}
6221     table.insert(start_output, self.secbegin_text)
6222     table.insert(end_output, self.secend_text)
6223
6224     table.insert(buf, self.push_attributes("heading",
6225                                         normalized_attributes,
6226                                         start_output,
6227                                         end_output))
6228     assert(self.attribute_type_levels["heading"] == level)
6229
6230     -- render the heading and its attributes
6231     if self.is_writing and #normalized_attributes > 0 then
6232         table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin\n")
6233         table.insert(buf, self.attributes(normalized_attributes, false))
6234     end
6235
6236     local cmd
6237     level = level + options.shiftHeadings
6238     if level <= 1 then
6239         cmd = "\\markdownRendererHeadingOne"
6240     elseif level == 2 then
6241         cmd = "\\markdownRendererHeadingTwo"
6242     elseif level == 3 then
6243         cmd = "\\markdownRendererHeadingThree"
6244     elseif level == 4 then
6245         cmd = "\\markdownRendererHeadingFour"
6246     elseif level == 5 then
6247         cmd = "\\markdownRendererHeadingFive"
6248     elseif level >= 6 then
6249         cmd = "\\markdownRendererHeadingSix"
6250     else
6251         cmd = ""
6252     end
6253     if self.is_writing then
6254         table.insert(buf, {cmd, "{", inlines, "}"})
6255     end
6256
6257     if self.is_writing and #normalized_attributes > 0 then
6258         table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd{")
6259     end
6260
6261     return buf

```



```
6262 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.

```
6263 function self.get_state()
6264   return {
6265     is_writing=self.is_writing,
6266     flatten_inlines=self.flatten_inlines,
6267     active_attributes={table.unpack(self.active_attributes)},
6268   }
6269 end
```

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

```
6270 function self.set_state(s)
6271   local previous_state = self.get_state()
6272   for key, value in pairs(s) do
6273     self[key] = value
6274   end
6275   return previous_state
6276 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`.

```
6277 function self.defer_call(f)
6278   local previous_state = self.get_state()
6279   return function(...)
6280     local state = self.set_state(previous_state)
6281     local return_value = f(...)
6282     self.set_state(state)
6283     return return_value
6284   end
6285 end
6286
6287 return self
6288 end
```

### 3.1.4 Parsers

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

```
6289 local parsers = {}
```

#### 3.1.4.1 Basic Parsers

```
6290 parsers.percent = P("%")
6291 parsers.at = P("@")
```

```

6292 parsers.comma           = P(",")
6293 parsers.asterisk        = P("*")
6294 parsers.dash             = P("-")
6295 parsers.plus             = P("+")
6296 parsers.underscore       = P("_")
6297 parsers.period           = P(".")
6298 parsers.hash             = P("#")
6299 parsers.dollar           = P("$")
6300 parsers.ampersand        = P("&")
6301 parsers.backtick         = P("`")
6302 parsers.less             = P("<")
6303 parsers.more             = P(">")
6304 parsers.space            = P(" ")
6305 parsers.squote           = P("'")
6306 parsers.dquote           = P('"')
6307 parsers.lparent          = P("(")
6308 parsers.rparent          = P(")")
6309 parsers.lbracket         = P("[")
6310 parsers.rbracket         = P("]")
6311 parsers.lbrace           = P("{")
6312 parsers.rbrace           = P("}")
6313 parsers.circumflex       = P("^")
6314 parsers.slash            = P("/")
6315 parsers.equal            = P("=")
6316 parsers.colon            = P(":")
6317 parsers.semicolon        = P(";")
6318 parsers.exclamation      = P("!")
6319 parsers.pipe             = P("|")
6320 parsers.tilde            = P("~")
6321 parsers.backslash        = P("\\")
6322 parsers.tab              = P("\t")
6323 parsers.newline          = P("\n")
6324
6325 parsers.digit            = R("09")
6326 parsers.hexdigit         = R("09","af","AF")
6327 parsers.letter           = R("AZ","az")
6328 parsers.alphanumeric     = R("AZ","az","09")
6329 parsers.keyword          = parsers.letter
6330                          * (parsers.alphanumeric + parsers.dash)^0
6331
6332 parsers.doubleasterisks   = P("**")
6333 parsers.doubleunderscores = P("__")
6334 parsers.doubletildes     = P("~~")
6335 parsers.fourspace        = P("    ")
6336
6337 parsers.any               = P(1)
6338 parsers.succeed          = P(true)

```

```

6339 parsers.fail = P(false)
6340
6341 parsers.internal_punctuation = S(":,;.?.")
6342 parsers.ascii_punctuation = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")

```

### 3.1.5 Unicode punctuation

This section documents the Unicode punctuation<sup>33</sup> recognized by the markdown reader. The punctuation is organized in the `parsers.punctuation` table according to the number of bytes occupied after conversion to UTF8.

(CommonMark Spec, Version 0.31.2 (2024-01-28))

```

6343 parsers.punctuation = {}
6344 (function()
6345   local pathname = kpse.lookup("UnicodeData.txt")
6346   local file = assert(io.open(pathname, "r"),
6347     [[Could not open file "UnicodeData.txt"]])
6348   for line in file:lines() do
6349     local codepoint, major_category = line:match("^(%x+);[^\;]*;(%)a")
6350     if major_category == "P" or major_category == "S" then
6351       local code = unicode.utf8.char(tonumber(codepoint, 16))
6352       if parsers.punctuation[#code] == nil then
6353         parsers.punctuation[#code] = parsers.fail
6354       end
6355       local code_parser = parsers.succeed
6356       for i = 1, #code do
6357         local byte = code:sub(i, i)
6358         local byte_parser = S(byte)
6359         code_parser = code_parser
6360           * byte_parser
6361       end
6362       parsers.punctuation[#code] = parsers.punctuation[#code]
6363         + code_parser
6364     end
6365   end
6366   assert(file:close())
6367 end)()
6368
6369 parsers.escapable = parsers.ascii_punctuation
6370 parsers.anyescaped = parsers.backslash / "" * parsers.escapable
6371 + parsers.any
6372
6373 parsers.spacechar = S("\t ")

```

<sup>33</sup>See <https://spec.commonmark.org/0.31.2/#unicode-punctuation-character>.

```

6374 parsers.spacing           = S(" \n\r\t")
6375 parsers.nospacechar      = parsers.any - parsers.spacing
6376 parsers.optionalspace    = parsers.spacechar^0
6377
6378 parsers.normalchar         = parsers.any - (V("SpecialChar")
6379                                     + parsers.spacing)
6380 parsers.eof                 = -parsers.any
6381 parsers.nonindentspace     = parsers.space^-3 * - parsers.spacechar
6382 parsers.indent             = parsers.space^-3 * parsers.tab
6383                             + parsers.fourspace / ""
6384 parsers.linechar           = P(1 - parsers.newline)
6385
6386 parsers.blankline          = parsers.optionalspace
6387                             * parsers.newline / "\n"
6388 parsers.blanklines         = parsers.blankline^0
6389 parsers.skipblanklines     = (parsers.optionalspace * parsers.newline)^0
6390 parsers.indentedline       = parsers.indent / ""
6391                             * C(parsers.linechar^1 * parsers.newline^-
6392                                 1)
6393 parsers.optionallyindentedline = parsers.indent^-1 / ""
6394                             * C(parsers.linechar^1 * parsers.newline^-
6395                                 1)
6396 parsers.sp                  = parsers.spacing^0
6397 parsers.spnl                = parsers.optionalspace
6398                             * (parsers.newline * parsers.optionalspace)^-
6399                             1
6400 parsers.line                 = parsers.linechar^0 * parsers.newline
6401 parsers.nonemptyline        = parsers.line - parsers.blankline

```

### 3.1.5.1 Parsers Used for Indentation

```

6399
6400 parsers.leader             = parsers.space^-3
6401

```

Check if a trail exists and is non-empty in the indent table `indent_table`.

```

6402 local function has_trail(indent_table)
6403   return indent_table ~= nil and
6404         indent_table.trail ~= nil and
6405         next(indent_table.trail) ~= nil
6406 end
6407

```

Check if indent table `indent_table` has any indents.

```

6408 local function has_indents(indent_table)
6409   return indent_table ~= nil and
6410         indent_table.indents ~= nil and
6411         next(indent_table.indents) ~= nil

```

```
6412 end
6413
```

Add a trail `trail_info` to the indent table `indent_table`.

```
6414 local function add_trail(indent_table, trail_info)
6415     indent_table.trail = trail_info
6416     return indent_table
6417 end
6418
```

Remove a trail `trail_info` from the indent table `indent_table`.

```
6419 local function remove_trail(indent_table)
6420     indent_table.trail = nil
6421     return indent_table
6422 end
6423
```

Update the indent table `indent_table` by adding or removing a new indent `add`.

```
6424 local function update_indent_table(indent_table, new_indent, add)
6425     indent_table = remove_trail(indent_table)
6426
6427     if not has_indents(indent_table) then
6428         indent_table.indents = {}
6429     end
6430
6431
6432     if add then
6433         indent_table.indents[#indent_table.indents + 1] = new_indent
6434     else
6435         if indent_table.indents[#indent_table.indents].name == new_indent.name then
6436             indent_table.indents[#indent_table.indents] = nil
6437         end
6438     end
6439
6440     return indent_table
6441 end
6442
```

Remove an indent by its name `name`.

```
6443 local function remove_indent(name)
6444     local function remove_indent_level(s, i, indent_table) -- luacheck: ignore s i
6445         indent_table = update_indent_table(indent_table, {name=name}, false)
6446         return true, indent_table
6447     end
6448
6449     return Cg(Cmt(Cb("indent_info"), remove_indent_level), "indent_info")
6450 end
6451
```

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`.

```
6452 local function process_starter_spacing(indent, spacing, minimum, left_strip_length)
6453   left_strip_length = left_strip_length or 0
6454
6455   local count = 0
6456   local tab_value = 4 - (indent) % 4
6457
6458   local code_started, minimum_found = false, false
6459   local code_start, minimum_remainder = "", ""
6460
6461   local left_total_stripped = 0
6462   local full_remainder = ""
6463
6464   if spacing ~= nil then
6465     for i = 1, #spacing do
6466       local character = spacing:sub(i, i)
6467
6468       if character == "\t" then
6469         count = count + tab_value
6470         tab_value = 4
6471       elseif character == " " then
6472         count = count + 1
6473         tab_value = 4 - (1 - tab_value) % 4
6474       end
6475
6476       if (left_strip_length ~= 0) then
6477         local possible_to_strip = math.min(count, left_strip_length)
6478         count = count - possible_to_strip
6479         left_strip_length = left_strip_length - possible_to_strip
6480         left_total_stripped = left_total_stripped + possible_to_strip
6481       else
6482         full_remainder = full_remainder .. character
6483       end
6484
6485       if (minimum_found) then
6486         minimum_remainder = minimum_remainder .. character
6487       elseif (count >= minimum) then
6488         minimum_found = true
6489         minimum_remainder = minimum_remainder .. string.rep(" ", count - minimum)
6490       end
6491
6492       if (code_started) then
```

```

6493     code_start = code_start .. character
6494     elseif (count >= minimum + 4) then
6495         code_started = true
6496         code_start = code_start .. string.rep(" ", count - (minimum + 4))
6497     end
6498 end
6499 end
6500
6501 local remainder
6502 if (code_started) then
6503     remainder = code_start
6504 else
6505     remainder = string.rep(" ", count - minimum)
6506 end
6507
6508 local is_minimum = count >= minimum
6509 return {
6510     is_code = code_started,
6511     remainder = remainder,
6512     left_total_stripped = left_total_stripped,
6513     is_minimum = is_minimum,
6514     minimum_remainder = minimum_remainder,
6515     total_length = count,
6516     full_remainder = full_remainder
6517 }
6518 end
6519

```

Count the total width of all indents in the indent table `indent_table`.

```

6520 local function count_indent_tab_level(indent_table)
6521     local count = 0
6522     if not has_indents(indent_table) then
6523         return count
6524     end
6525
6526     for i=1, #indent_table.indents do
6527         count = count + indent_table.indents[i].length
6528     end
6529     return count
6530 end
6531

```

Count the total width of a delimiter `delimiter`.

```

6532 local function total_delimiter_length(delimiter)
6533     local count = 0
6534     if type(delimiter) == "string" then return #delimiter end
6535     for _, value in pairs(delimiter) do
6536         count = count + total_delimiter_length(value)

```

```

6537 end
6538 return count
6539 end
6540

```

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`.

```

6541 local function process_starter_indent(_, _, indent_table, starter, is_blank, indent_t
6542     local last_trail = starter[1]
6543     local delimiter = starter[2]
6544     local raw_new_trail = starter[3]
6545
6546     if indent_type == "bq" and not breakable then
6547         indent_table.ignore_blockquote_blank = true
6548     end
6549
6550     if has_trail(indent_table) then
6551         local trail = indent_table.trail
6552         if trail.is_code then
6553             return false
6554         end
6555         last_trail = trail.remainder
6556     else
6557         local sp = process_starter_spacing(0, last_trail, 0, 0)
6558
6559         if sp.is_code then
6560             return false
6561         end
6562         last_trail = sp.remainder
6563     end
6564
6565     local preceding_indentation = count_indent_tab_level(indent_table) % 4
6566     local last_trail_length = #last_trail
6567     local delimiter_length = total_delimiter_length(delimiter)
6568
6569     local total_indent_level = preceding_indentation + last_trail_length + delimiter_le
6570
6571     local sp = {}
6572     if not is_blank then
6573         sp = process_starter_spacing(total_indent_level, raw_new_trail, 0, 1)
6574     end
6575
6576     local del_trail_length = sp.left_total_stripped
6577     if is_blank then
6578         del_trail_length = 1
6579     elseif not sp.is_code then
6580         del_trail_length = del_trail_length + #sp.remainder

```



```

6581 end
6582
6583 local indent_length = last_trail_length + delimiter_length + del_trail_length
6584 local new_indent_info = {name=indent_type, length=indent_length}
6585
6586 indent_table = update_indent_table(indent_table, new_indent_info, true)
6587 indent_table = add_trail(indent_table, {is_code=sp.is_code, remainder=sp.remainder,
6588                                     full_remainder=sp.full_remainder})
6589
6590 return true, indent_table
6591 end
6592

```

Return the pattern corresponding with the indent name `name`.

```

6593 local function decode_pattern(name)
6594     local delimiter = parsers.succeed
6595     if name == "bq" then
6596         delimiter = parsers.more
6597     end
6598
6599     return C(parsers.optionalspace) * C(delimiter) * C(parsers.optionalspace) * Cp()
6600 end
6601

```

Find the first blank-only indent of the indent table `indent_table` followed by blank-only indents.

```

6602 local function left_blank_starter(indent_table)
6603     local blank_starter_index
6604
6605     if not has_indents(indent_table) then
6606         return
6607     end
6608
6609     for i = #indent_table.indents,1,-1 do
6610         local value = indent_table.indents[i]
6611         if value.name == "li" then
6612             blank_starter_index = i
6613         else
6614             break
6615         end
6616     end
6617
6618     return blank_starter_index
6619 end
6620

```

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.

```
6621 local function traverse_indent(s, i, indent_table, is_optional, is_blank, current_line_index)
6622     local new_index = i
6623
6624     local preceding_indentation = 0
6625     local current_trail = {}
6626
6627     local blank_starter = left_blank_starter(indent_table)
6628
6629     if current_line_indents == nil then
6630         current_line_indents = {}
6631     end
6632
6633     for index = 1, #indent_table.indents do
6634         local value = indent_table.indents[index]
6635         local pattern = decode_pattern(value.name)
6636
6637         -- match decoded pattern
6638         local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
6639         if new_indent_info == nil then
6640             local blankline_end = lpeg.match(Ct(parsers.blankline * Cg(Cp(), "pos")), s, new_index)
6641             if is_optional or not indent_table.ignore_blockquote_blank or not blankline_end then
6642                 return is_optional, new_index, current_trail, current_line_indents
6643             end
6644
6645             return traverse_indent(s, tonumber(blankline_end.pos), indent_table, is_optional, is_blank, new_index)
6646         end
6647
6648         local raw_last_trail = new_indent_info[1]
6649         local delimiter = new_indent_info[2]
6650         local raw_new_trail = new_indent_info[3]
6651         local next_index = new_indent_info[4]
6652
6653         local space_only = delimiter == " "
6654
6655         -- check previous trail
6656         if not space_only and next(current_trail) == nil then
6657             local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
6658             current_trail = {is_code=sp.is_code, remainder=sp.remainder, total_length=sp.total_length,
6659                             full_remainder=sp.full_remainder}
6660         end
6661
6662         if next(current_trail) ~= nil then
6663             if not space_only and current_trail.is_code then
6664                 return is_optional, new_index, current_trail, current_line_indents
6665             end
6666         end
6667     end
6668 end
```

```

6665     end
6666     if current_trail.internal_remainder ~= nil then
6667         raw_last_trail = current_trail.internal_remainder
6668     end
6669 end
6670
6671 local raw_last_trail_length = 0
6672 local delimiter_length = 0
6673
6674 if not space_only then
6675     delimiter_length = #delimiter
6676     raw_last_trail_length = #raw_last_trail
6677 end
6678
6679 local total_indent_level = preceding_indentation + raw_last_trail_length + delimi
6680
6681 local spacing_to_process
6682 local minimum = 0
6683 local left_strip_length = 0
6684
6685 if not space_only then
6686     spacing_to_process = raw_new_trail
6687     left_strip_length = 1
6688 else
6689     spacing_to_process = raw_last_trail
6690     minimum = value.length
6691 end
6692
6693 local sp = process_starter_spacing(total_indent_level, spacing_to_process, minimu
6694
6695 if space_only and not sp.is_minimum then
6696     return is_optional or (is_blank and blank_starter <= index), new_index, current
6697 end
6698
6699 local indent_length = raw_last_trail_length + delimiter_length + sp.left_total_st
6700
6701 -- update info for the next pattern
6702 if not space_only then
6703     preceding_indentation = preceding_indentation + indent_length
6704 else
6705     preceding_indentation = preceding_indentation + value.length
6706 end
6707
6708 current_trail = {is_code=sp.is_code, remainder=sp.remainder, internal_remainder=s
6709                 total_length=sp.total_length, full_remainder=sp.full_remainder}
6710
6711 current_line_indents[#current_line_indents + 1] = new_indent_info

```

```

6712     new_index = next_index
6713 end
6714
6715 return true, new_index, current_trail, current_line_indents
6716 end
6717

```

Check if a code trail is expected.

```

6718 local function check_trail(expect_code, is_code)
6719     return (expect_code and is_code) or (not expect_code and not is_code)
6720 end
6721

```

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`.

```

6722 local function check_trail_joined(s, i, indent_table, spacing, expect_code, omit_remainder)
6723     local is_code
6724     local remainder
6725
6726     if has_trail(indent_table) then
6727         local trail = indent_table.trail
6728         is_code = trail.is_code
6729         if is_code then
6730             remainder = trail.remainder
6731         else
6732             remainder = trail.full_remainder
6733         end
6734     else
6735         local sp = process_starter_spacing(0, spacing, 0, 0)
6736         is_code = sp.is_code
6737         if is_code then
6738             remainder = sp.remainder
6739         else
6740             remainder = sp.full_remainder
6741         end
6742     end
6743
6744     local result = check_trail(expect_code, is_code)
6745     if omit_remainder then
6746         return result
6747     end
6748     return result, remainder
6749 end
6750

```

Check if the current trail of the `indent_table` is of length between `min` and `max`.

```

6751 local function check_trail_length(s, i, indent_table, spacing, min, max) -- luacheck:

```

```

6752 local trail
6753
6754 if has_trail(indent_table) then
6755     trail = indent_table.trail
6756 else
6757     trail = process_starter_spacing(0, spacing, 0, 0)
6758 end
6759
6760 local total_length = trail.total_length
6761 if total_length == nil then
6762     return false
6763 end
6764
6765 return min <= total_length and total_length <= max
6766 end
6767

```

Check the indentation of the continuation line, optionally with the mode `is_optional` selected. Check blank line exclusively with `is_blank`.

```

6768 local function check_continuation_indentation(s, i, indent_table, is_optional, is_bla
6769     if not has_indents(indent_table) then
6770         return true
6771     end
6772
6773     local passes, new_index, current_trail, current_line_indents =
6774         traverse_indent(s, i, indent_table, is_optional, is_blank)
6775
6776     if passes then
6777         indent_table.current_line_indents = current_line_indents
6778         indent_table = add_trail(indent_table, current_trail)
6779         return new_index, indent_table
6780     end
6781     return false
6782 end
6783

```

Get name of the last indent from the `indent_table`.

```

6784 local function get_last_indent_name(indent_table)
6785     if has_indents(indent_table) then
6786         return indent_table.indents[#indent_table.indents].name
6787     end
6788 end
6789

```

Remove the remainder altogether if the last indent from the `indent_table` is blank-only.

```

6790 local function remove_remainder_if_blank(indent_table, remainder)
6791     if get_last_indent_name(indent_table) == "li" then

```

```

6792     return ""
6793 end
6794 return remainder
6795 end
6796

```

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.

```

6797 local function check_trail_type(s, i, trail, spacing, trail_type) -- luacheck: ignore s
6798     if trail == nil then
6799         trail = process_starter_spacing(0, spacing, 0, 0)
6800     end
6801
6802     if trail_type == "non-code" then
6803         return check_trail(false, trail.is_code)
6804     end
6805     if trail_type == "code" then
6806         return check_trail(true, trail.is_code)
6807     end
6808     if trail_type == "full-code" then
6809         if (trail.is_code) then
6810             return i, trail.remainder
6811         end
6812         return i, ""
6813     end
6814     if trail_type == "full-any" then
6815         return i, trail.internal_remainder
6816     end
6817 end
6818

```

Stores or restores an `is_freezing` trail from indent table `indent_table`.

```

6819 local function trail_freezing(s, i, indent_table, is_freezing) -- luacheck: ignore s
6820     if is_freezing then
6821         if indent_table.is_trail_frozen then
6822             indent_table.trail = indent_table.frozen_trail
6823         else
6824             indent_table.frozen_trail = indent_table.trail
6825             indent_table.is_trail_frozen = true
6826         end
6827     else
6828         indent_table.frozen_trail = nil
6829         indent_table.is_trail_frozen = false
6830     end
6831     return true, indent_table
6832 end
6833

```

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`.

```
6834 local function check_continuation_indentation_and_trail(s, i, indent_table, is_optional,
6835                                     reset_rem, omit_remainder)
6836   if not has_indents(indent_table) then
6837     local spacing, new_index = lpeg.match(C(parsers.spacechar^0) * Cp(), s, i)
6838     local result, remainder = check_trail_type(s, i, indent_table.trail, spacing, trail_type)
6839     if remainder == nil then
6840       if result then
6841         return new_index
6842       end
6843       return false
6844     end
6845     if result then
6846       return new_index, remainder
6847     end
6848     return false
6849   end
6850
6851   local passes, new_index, current_trail = traverse_indent(s, i, indent_table, is_optional)
6852
6853   if passes then
6854     local spacing
6855     if current_trail == nil then
6856       local newer_spacing, newer_index = lpeg.match(C(parsers.spacechar^0) * Cp(), s, i)
6857       current_trail = process_starter_spacing(0, newer_spacing, 0, 0)
6858       new_index = newer_index
6859       spacing = newer_spacing
6860     else
6861       spacing = current_trail.remainder
6862     end
6863     local result, remainder = check_trail_type(s, new_index, current_trail, spacing, trail_type)
6864     if remainder == nil or omit_remainder then
6865       if result then
6866         return new_index
6867       end
6868       return false
6869     end
6870
6871     if is_blank and reset_rem then
6872       remainder = remove_remainder_if_blank(indent_table, remainder)
6873     end
6874     if result then
6875       return new_index, remainder
6876     end
6877     return false
```

```

6878 end
6879 return false
6880 end
6881

```

The following patterns check whitespace indentation at the start of a block.

```

6882 parsers.check_trail = Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(false), che
6883
6884 parsers.check_trail_no_rem = Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(fals
6885
6886 parsers.check_code_trail = Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(true)
6887
6888 parsers.check_trail_length_range = function(min, max)
6889   return Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(min) * Cc(max), check_tr
6890 end
6891
6892 parsers.check_trail_length = function(n)
6893   return parsers.check_trail_length_range(n, n)
6894 end
6895

```

The following patterns handle trail backup, to prevent a failing pattern to modify it before passing it to the next.

```

6896 parsers.freeze_trail = Cg(Cmt(Cb("indent_info") * Cc(true), trail_freezing), "indent_
6897
6898 parsers.unfreeze_trail = Cg(Cmt(Cb("indent_info") * Cc(false), trail_freezing), "inde
6899

```

The following patterns check indentation in continuation lines as defined by the container start.

```

6900 parsers.check_minimal_indent = Cmt(Cb("indent_info") * Cc(false), check_continuation_
6901
6902 parsers.check_optional_indent = Cmt(Cb("indent_info") * Cc(true), check_continuation_
6903
6904 parsers.check_minimal_blank_indent = Cmt(Cb("indent_info") * Cc(false) * Cc(true), ch
6905

```

The following patterns check indentation in continuation lines as defined by the container start. Additionally the subsequent trail is also directly checked.

```

6906
6907 parsers.check_minimal_indent_and_trail = Cmt( Cb("indent_info")
6908           * Cc(false) * Cc(false) * Cc("non-
        code") * Cc(true),
6909           check_continuation_indentation_and_trail)
6910
6911 parsers.check_minimal_indent_and_code_trail = Cmt( Cb("indent_info")
6912           * Cc(false) * Cc(false) * Cc("code")
6913           check_continuation_indentation_and_t

```



```

6914
6915 parsers.check_minimal_blank_indent_and_full_code_trail = Cmt( Cb("indent_info")
6916                               * Cc(false) * Cc(true) *
        code") * Cc(true),
6917                               check_continuation_indentation_and_trailing_spaces)
6918
6919 parsers.check_minimal_indent_and_any_trail = Cmt( Cb("indent_info")
6920                               * Cc(false) * Cc(false) * Cc("full-
        any") * Cc(true) * Cc(false),
6921                               check_continuation_indentation_and_trailing_spaces)
6922
6923 parsers.check_minimal_blank_indent_and_any_trail = Cmt( Cb("indent_info")
6924                               * Cc(false) * Cc(true) * Cc("full-
        any") * Cc(true) * Cc(false),
6925                               check_continuation_indentation_and_trailing_spaces)
6926
6927 parsers.check_minimal_blank_indent_and_any_trail_no_rem = Cmt( Cb("indent_info")
6928                               * Cc(false) * Cc(true) * Cc("full-
        any") * Cc(true) * Cc(true),
6929                               check_continuation_indentation_and_trailing_spaces)
6930
6931 parsers.check_optional_indent_and_any_trail = Cmt( Cb("indent_info")
6932                               * Cc(true) * Cc(false) * Cc("full-
        any") * Cc(true) * Cc(false),
6933                               check_continuation_indentation_and_trailing_spaces)
6934
6935 parsers.check_optional_blank_indent_and_any_trail = Cmt( Cb("indent_info")
6936                               * Cc(true) * Cc(true) * Cc("full-
        any") * Cc(true) * Cc(false),
6937                               check_continuation_indentation_and_trailing_spaces)
6938

```

The following patterns specify behaviour around newlines.

```

6939
6940 parsers.spnlc_noexc = parsers.optionalspace
6941                       * (parsers.newline * parsers.check_minimal_indent_and_any_trail)^
        1
6942
6943 parsers.spnlc = parsers.optionalspace
6944                 * (V("EndlineNoSub"))^-1
6945
6946 parsers.spnlc_sep = parsers.optionalspace * V("EndlineNoSub")
6947                  + parsers.spacechar^1
6948
6949 parsers.only_blank = parsers.spacechar^0 * (parsers.newline + parsers.eof)
6950
6951 % \end{macrocode}
6952 % \begin{figure}

```

```

6953 % \hspace*{-0.1\textwidth}
6954 % \begin{minipage}{1.2\textwidth}
6955 % \centering
6956 % \begin{tikzpicture}[shorten >=1pt, line width=0.1mm, >={Stealth[length=2mm]}, node
6957 % \node[state, initial by diamond, accepting] (noop) {initial};
6958 % \node[state] (odd_backslash) [above right=of noop] {odd backslash};
6959 % \node[state] (even_backslash) [below right=of odd_backslash] {even backslash};
6960 % \node[state] (comment) [below=of noop] {comment};
6961 % \node[state] (leading_spaces) [below=of even_backslash, align=center] {leading tabs};
6962 % \node[state] (blank_line) [below right=of comment] {blank line};
6963 % \path[->]
6964 % (noop) edge [in=150, out=180, loop] node [align=center, yshift=-0.75cm] {match [\$^
6965 % edge [bend right=10] node [below right=-0.2cm] {match \textbackslash} (odd_b
6966 % edge [bend left=30] node [left, align=center] {match \%\\capture \textbacksl
6967 % (comment) edge [in=305, out=325, loop] node [xshift=-1.2cm] {match [\$^\\wedge$$\drsh
6968 % edge [bend left=10] node {match \$\drsh$} (leading_spaces)
6969 % (leading_spaces) edge [loop below] node {match [\textvisiblespace$\rightleftarrows$}
6970 % edge [bend right=90] node [right] {match \textbackslash} (odd_back
6971 % edge [bend left=10] node {match \%} (comment)
6972 % edge [bend right=10] node {\$\\epsilon$} (blank_line)
6973 % edge [bend left=10] node [align=center, right=0.3cm] {match [\$^\\we
6974 % (blank_line) edge [loop below] node {match [\textvisiblespace$\rightleftarrows$]} (
6975 % edge [bend left=90] node [align=center, below=1.2cm] {match \$\drsh$\\
6976 % (odd_backslash) edge [bend right=10] node [align=center, xshift=-0.3cm, yshift=0.2c
6977 % edge [bend right=10] node [align=center, above left=-
6978 % 0.3cm, xshift=0.1cm] {match [\$^\\wedge$\\textbackslash]\\for \%, capture \textbacksl
6979 % (even_backslash) edge [bend left=10] node {\$\\epsilon$} (noop);
6980 % \end{tikzpicture}
6981 % \caption{A pushdown automaton that recognizes \TeX{} comments}
6982 % \label{fig:commented_line}
6983 % \end{minipage}
6984 % \end{figure}
6985 % \begin{markdown}
6986 % The \luamdef{parsers.commented_line}`~1` parser recognizes the regular
6987 % language of \TeX{} comments, see an equivalent finite automaton in Figure
6988 % <#fig:commented_line>.
6989 %
6990 % \end{markdown}
6991 % \begin{macrocode}
6992 parsers.commented_line_letter = parsers.linechar
6993 + parsers.newline
6994 - parsers.backslash
6995 - parsers.percent
6996 parsers.commented_line = Cg(Cc(""), "backslashes")
6997 * ((#(parsers.commented_line_letter
6998 - parsers.newline)

```

```

6999         * Cb("backslashes")
7000         * Cs(parsers.commented_line_letter
7001             - parsers.newline)^1 -- initial
7002         * Cg(Cc(""), "backslashes"))
7003 + #(parsers.backslash * parsers.backslash)
7004 * Cg((parsers.backslash -- even backslash
7005     * parsers.backslash)^1, "backslashes")
7006 + (parsers.backslash
7007     * (#parsers.percent
7008     * Cb("backslashes")
7009     / function(backslashes)
7010         return string.rep("\\", #backslashes / 2)
7011     end
7012     * C(parsers.percent)
7013     + #parsers.commented_line_letter
7014     * Cb("backslashes")
7015     * Cc("\\")
7016     * C(parsers.commented_line_letter))
7017     * Cg(Cc(""), "backslashes"))^0
7018 * (#parsers.percent
7019 * Cb("backslashes")
7020 / function(backslashes)
7021     return string.rep("\\", #backslashes / 2)
7022 end
7023 * ((parsers.percent -- comment
7024     * parsers.line
7025     * #parsers.blankline) -- blank line
7026     / "\n"
7027     + parsers.percent -- comment
7028     * parsers.line
7029     * parsers.optionalspace) -- leading tabs and space
7030 + #(parsers.newline)
7031 * Cb("backslashes")
7032 * C(parsers.newline))
7033
7034 parsers.chunk = parsers.line * (parsers.optionallyindentedline
7035     - parsers.blankline)^0
7036
7037 parsers.attribute_key_char = parsers.alphanumeric + S("-_:.")
7038 parsers.attribute_raw_char = parsers.alphanumeric + S("-_")
7039 parsers.attribute_key = (parsers.attribute_key_char
7040     - parsers.dash - parsers.digit)
7041 * parsers.attribute_key_char^0
7042 parsers.attribute_value = ( (parsers.dquote / "\"")
7043     * (parsers.anyescaped - parsers.dquote)^0
7044     * (parsers.dquote / "\""))
7045 + ( (parsers.squote / "\"")

```

```

7046             * (parsers.anyescaped - parsers.squote)^0
7047             * (parsers.squote / ""))
7048         + ( parsers.anyescaped - parsers.dquote - parsers.rbra
7049           - parsers.space)^0
7050 parsers.attribute_identifier = parsers.attribute_key_char^1
7051 parsers.attribute_classname = parsers.letter
7052             * parsers.attribute_key_char^0
7053 parsers.attribute_raw       = parsers.attribute_raw_char^1
7054
7055 parsers.attribute = (parsers.dash * Cc(".unnumbered"))
7056                   + C( parsers.hash
7057                       * parsers.attribute_identifier)
7058                   + C( parsers.period
7059                       * parsers.attribute_classname)
7060                   + Cs( parsers.attribute_key
7061                       * parsers.optionalspace * parsers.equal * parsers.optionalspace
7062                       * parsers.attribute_value)
7063 parsers.attributes = parsers.lbrace
7064                   * parsers.optionalspace
7065                   * parsers.attribute
7066                   * (parsers.spacechar^1
7067                     * parsers.attribute)^0
7068                   * parsers.optionalspace
7069                   * parsers.rbrace
7070
7071
7072 parsers.raw_attribute = parsers.lbrace
7073                   * parsers.optionalspace
7074                   * parsers.equal
7075                   * C(parsers.attribute_raw)
7076                   * parsers.optionalspace
7077                   * parsers.rbrace
7078
7079 -- block followed by 0 or more optionally
7080 -- indented blocks with first line indented.
7081 parsers.indented_blocks = function(bl)
7082   return Cs( bl
7083             * (parsers.blankline^1 * parsers.indent * -parsers.blankline * bl)^0
7084             * (parsers.blankline^1 + parsers.eof) )
7085 end

```

### 3.1.5.2 Parsers Used for HTML Entities

```

7086 local function repeat_between(pattern, min, max)
7087   return -pattern^(max + 1) * pattern^min
7088 end
7089

```

```

7090 parsers.hexentity = parsers.ampersand * parsers.hash * C(S("Xx"))
7091                * C(repeat_between(parsers.hexdigit, 1, 6)) * parsers.semicolon
7092 parsers.decentity = parsers.ampersand * parsers.hash
7093                * C(repeat_between(parsers.digit, 1, 7)) * parsers.semicolon
7094 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
7095                * parsers.semicolon
7096
7097 parsers.html_entities = parsers.hexentity / entities.hex_entity_with_x_char
7098                + parsers.decentity / entities.dec_entity
7099                + parsers.tagentity / entities.char_entity

```

### 3.1.5.3 Parsers Used for Markdown Lists

```

7100 parsers.bullet = function(bullet_char, interrupting)
7101   local allowed_end
7102   if interrupting then
7103     allowed_end = C(parsers.spacechar^1) * #parsers.linechar
7104   else
7105     allowed_end = C(parsers.spacechar^1) + #(parsers.newline + parsers.eof)
7106   end
7107   return parsers.check_trail
7108         * Ct(C(bullet_char) * Cc(""))
7109         * allowed_end
7110 end
7111
7112 local function tickbox(interior)
7113   return parsers.optionalspace * parsers.lbracket
7114         * interior * parsers.rbracket * parsers.spacechar^1
7115 end
7116
7117 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
7118 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
7119 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
7120

```

### 3.1.5.4 Parsers Used for Markdown Code Spans

```

7121 parsers.openticks = Cg(parsers.backtick^1, "ticks")
7122
7123 local function captures_equal_length(_,i,a,b)
7124   return #a == #b and i
7125 end
7126
7127 parsers.closeticks = Cmt(C(parsers.backtick^1)
7128                * Cb("ticks"), captures_equal_length)
7129
7130 parsers.intickschar = (parsers.any - S("\n\r`"))
7131                + V("NoSoftLineBreakEndline")

```

```

7132             + (parsers.backtick^1 - parsers.closeticks)
7133
7134 local function process_inticks(s)
7135     s = s:gsub("\n", " ")
7136     s = s:gsub("^ (.*) $", "%1")
7137     return s
7138 end
7139
7140 parsers.inticks = parsers.openticks
7141                 * C(parsers.space^0)
7142                 * parsers.closeticks
7143                 + parsers.openticks
7144                 * Cs(Cs(parsers.intickschar^0) / process_inticks)
7145                 * parsers.closeticks
7146

```

### 3.1.5.5 Parsers Used for HTML

```

7147 -- case-insensitive match (we assume s is lowercase). must be single byte encoding
7148 parsers.keyword_exact = function(s)
7149     local parser = P(0)
7150     for i=1,#s do
7151         local c = s:sub(i,i)
7152         local m = c .. upper(c)
7153         parser = parser * S(m)
7154     end
7155     return parser
7156 end
7157
7158 parsers.special_block_keyword =
7159     parsers.keyword_exact("pre") +
7160     parsers.keyword_exact("script") +
7161     parsers.keyword_exact("style") +
7162     parsers.keyword_exact("textarea")
7163
7164 parsers.block_keyword =
7165     parsers.keyword_exact("address") +
7166     parsers.keyword_exact("article") +
7167     parsers.keyword_exact("aside") +
7168     parsers.keyword_exact("base") +
7169     parsers.keyword_exact("basefont") +
7170     parsers.keyword_exact("blockquote") +
7171     parsers.keyword_exact("body") +
7172     parsers.keyword_exact("caption") +
7173     parsers.keyword_exact("center") +
7174     parsers.keyword_exact("col") +
7175     parsers.keyword_exact("colgroup") +

```

7176 parsers.keyword\_exact("dd") +  
7177 parsers.keyword\_exact("details") +  
7178 parsers.keyword\_exact("dialog") +  
7179 parsers.keyword\_exact("dir") +  
7180 parsers.keyword\_exact("div") +  
7181 parsers.keyword\_exact("dl") +  
7182 parsers.keyword\_exact("dt") +  
7183 parsers.keyword\_exact("fieldset") +  
7184 parsers.keyword\_exact("figcaption") +  
7185 parsers.keyword\_exact("figure") +  
7186 parsers.keyword\_exact("footer") +  
7187 parsers.keyword\_exact("form") +  
7188 parsers.keyword\_exact("frame") +  
7189 parsers.keyword\_exact("frameset") +  
7190 parsers.keyword\_exact("h1") +  
7191 parsers.keyword\_exact("h2") +  
7192 parsers.keyword\_exact("h3") +  
7193 parsers.keyword\_exact("h4") +  
7194 parsers.keyword\_exact("h5") +  
7195 parsers.keyword\_exact("h6") +  
7196 parsers.keyword\_exact("head") +  
7197 parsers.keyword\_exact("header") +  
7198 parsers.keyword\_exact("hr") +  
7199 parsers.keyword\_exact("html") +  
7200 parsers.keyword\_exact("iframe") +  
7201 parsers.keyword\_exact("legend") +  
7202 parsers.keyword\_exact("li") +  
7203 parsers.keyword\_exact("link") +  
7204 parsers.keyword\_exact("main") +  
7205 parsers.keyword\_exact("menu") +  
7206 parsers.keyword\_exact("menuitem") +  
7207 parsers.keyword\_exact("nav") +  
7208 parsers.keyword\_exact("noframes") +  
7209 parsers.keyword\_exact("ol") +  
7210 parsers.keyword\_exact("optgroup") +  
7211 parsers.keyword\_exact("option") +  
7212 parsers.keyword\_exact("p") +  
7213 parsers.keyword\_exact("param") +  
7214 parsers.keyword\_exact("section") +  
7215 parsers.keyword\_exact("source") +  
7216 parsers.keyword\_exact("summary") +  
7217 parsers.keyword\_exact("table") +  
7218 parsers.keyword\_exact("tbody") +  
7219 parsers.keyword\_exact("td") +  
7220 parsers.keyword\_exact("tfoot") +  
7221 parsers.keyword\_exact("th") +  
7222 parsers.keyword\_exact("thead") +

```

7223     parsers.keyword_exact("title") +
7224     parsers.keyword_exact("tr") +
7225     parsers.keyword_exact("track") +
7226     parsers.keyword_exact("ul")
7227
7228 -- end conditions
7229 parsers.html_blankline_end_condition = parsers.linechar^0
7230                                     * ( parsers.newline
7231                                       * (parsers.check_minimal_blank_indent_and_any
7232                                         * #parsers.blankline
7233                                           + parsers.check_minimal_indent_and_any_trai
7234                                             * parsers.linechar^1)^0
7235                                       * (parsers.newline^-1 / ""))
7236
7237 local function remove_trailing_blank_lines(s)
7238     return s:gsub("[\n\r]+%s*$", "")
7239 end
7240
7241 parsers.html_until_end = function(end_marker)
7242     return Cs(Cs((parsers.newline
7243                 * (parsers.check_minimal_blank_indent_and_any_trail
7244                   * #parsers.blankline
7245                     + parsers.check_minimal_indent_and_any_trail)
7246                 + parsers.linechar - end_marker)^0
7247               * parsers.linechar^0 * parsers.newline^-1)
7248             / remove_trailing_blank_lines)
7249 end
7250
7251 -- attributes
7252 parsers.html_attribute_spacing = parsers.optionalspace
7253                               * V("NoSoftLineBreakEndline")
7254                               * parsers.optionalspace
7255                               + parsers.spacechar^1
7256
7257 parsers.html_attribute_name = (parsers.letter + parsers.colon + parsers.underscore)
7258                             * (parsers.alphanumeric + parsers.colon + parsers.undersco
7259                               + parsers.period + parsers.dash)^0
7260
7261 parsers.html_attribute_value = parsers.squote
7262                              * (parsers.linechar - parsers.squote)^0
7263                              * parsers.squote
7264                              + parsers.dquote
7265                              * (parsers.linechar - parsers.dquote)^0
7266                              * parsers.dquote
7267                              + ( parsers.any - parsers.spacechar - parsers.newline
7268                                - parsers.dquote - parsers.squote - parsers.backtick
7269                                - parsers.equal - parsers.less - parsers.more)^1

```



```

7270
7271 parsers.html_inline_attribute_value = parsers.quote
7272     * (V("NoSoftLineBreakEndline")
7273     + parsers.any
7274     - parsers.blankline^2
7275     - parsers.quote)^0
7276     * parsers.quote
7277     + parsers.dquote
7278     * (V("NoSoftLineBreakEndline")
7279     + parsers.any
7280     - parsers.blankline^2
7281     - parsers.dquote)^0
7282     * parsers.dquote
7283     + (parsers.any - parsers.spacechar - parsers.newline
7284     - parsers.dquote - parsers.quote - parsers.backslash
7285     - parsers.equal - parsers.less - parsers.more)^0
7286
7287 parsers.html_attribute_value_specification = parsers.optionalspace
7288     * parsers.equal
7289     * parsers.optionalspace
7290     * parsers.html_attribute_value
7291
7292 parsers.html_spnl = parsers.optionalspace
7293     * (V("NoSoftLineBreakEndline") * parsers.optionalspace)^-
7294     1
7295
7296 parsers.html_inline_attribute_value_specification = parsers.html_spnl
7297     * parsers.equal
7298     * parsers.html_spnl
7299     * parsers.html_inline_attribute_value_specification
7300
7301 parsers.html_attribute = parsers.html_attribute_spacing
7302     * parsers.html_attribute_name
7303     * parsers.html_inline_attribute_value_specification^-
7304     1
7305
7306 parsers.html_non_newline_attribute = parsers.spacechar^1
7307     * parsers.html_attribute_name
7308     * parsers.html_attribute_value_specification^-
7309     1
7310
7311 parsers.nested_breaking_blank = parsers.newline
7312     * parsers.check_minimal_blank_indent
7313     * parsers.blankline
7314
7315 parsers.html_comment_start = P("<!--")
7316
7317

```

```

7314 parsers.html_comment_end = P("-->")
7315
7316 parsers.html_comment = Cs( parsers.html_comment_start
7317     * parsers.html_until_end(parsers.html_comment_end))
7318
7319 parsers.html_inline_comment = (parsers.html_comment_start / "")
7320     * -P(">") * -P("->")
7321     * Cs((V("NoSoftLineBreakEndline") + parsers.any
7322         - parsers.nested_breaking_blank - parsers.html_commen
7323     * (parsers.html_comment_end / ""))
7324
7325 parsers.html_cdatasection_start = P("<![CDATA[")
7326
7327 parsers.html_cdatasection_end = P("]]>")
7328
7329 parsers.html_cdatasection = Cs( parsers.html_cdatasection_start
7330     * parsers.html_until_end(parsers.html_cdatasection_end))
7331
7332 parsers.html_inline_cdatasection = parsers.html_cdatasection_start
7333     * Cs(V("NoSoftLineBreakEndline") + parsers.any
7334         - parsers.nested_breaking_blank - parsers.html_
7335     * parsers.html_cdatasection_end)
7336
7337 parsers.html_declaration_start = P("<!") * parsers.letter
7338
7339 parsers.html_declaration_end = P(">")
7340
7341 parsers.html_declaration = Cs( parsers.html_declaration_start
7342     * parsers.html_until_end(parsers.html_declaration_end))
7343
7344 parsers.html_inline_declaration = parsers.html_declaration_start
7345     * Cs(V("NoSoftLineBreakEndline") + parsers.any
7346         - parsers.nested_breaking_blank - parsers.html_de
7347     * parsers.html_declaration_end)
7348
7349 parsers.html_instruction_start = P("<?")
7350
7351 parsers.html_instruction_end = P(">")
7352
7353 parsers.html_instruction = Cs( parsers.html_instruction_start
7354     * parsers.html_until_end(parsers.html_instruction_end))
7355
7356 parsers.html_inline_instruction = parsers.html_instruction_start
7357     * Cs(V("NoSoftLineBreakEndline") + parsers.any
7358         - parsers.nested_breaking_blank - parsers.html_in
7359     * parsers.html_instruction_end)
7360

```

```

7361 parsers.html_blankline = parsers.newline
7362                             * parsers.optionalspace
7363                             * parsers.newline
7364
7365 parsers.html_tag_start = parsers.less
7366
7367 parsers.html_tag_closing_start = parsers.less
7368                             * parsers.slash
7369
7370 parsers.html_tag_end = parsers.html_spnl
7371                             * parsers.more
7372
7373 parsers.html_empty_tag_end = parsers.html_spnl
7374                             * parsers.slash
7375                             * parsers.more
7376
7377 -- opening tags
7378 parsers.html_any_open_inline_tag = parsers.html_tag_start
7379                                 * parsers.keyword
7380                                 * parsers.html_attribute^0
7381                                 * parsers.html_tag_end
7382
7383 parsers.html_any_open_tag = parsers.html_tag_start
7384                             * parsers.keyword
7385                             * parsers.html_non_newline_attribute^0
7386                             * parsers.html_tag_end
7387
7388 parsers.html_open_tag = parsers.html_tag_start
7389                             * parsers.block_keyword
7390                             * parsers.html_attribute^0
7391                             * parsers.html_tag_end
7392
7393 parsers.html_open_special_tag = parsers.html_tag_start
7394                                 * parsers.special_block_keyword
7395                                 * parsers.html_attribute^0
7396                                 * parsers.html_tag_end
7397
7398 -- incomplete tags
7399 parsers.incomplete_tag_following = parsers.spacechar
7400                                 + parsers.more
7401                                 + parsers.slash * parsers.more
7402                                 + #(parsers.newline + parsers.eof)
7403
7404 parsers.incomplete_special_tag_following = parsers.spacechar
7405                                             + parsers.more
7406                                             + #(parsers.newline + parsers.eof)
7407

```

```

7408 parsers.html_incomplete_open_tag = parsers.html_tag_start
7409                                     * parsers.block_keyword
7410                                     * parsers.incomplete_tag_following
7411
7412 parsers.html_incomplete_open_special_tag = parsers.html_tag_start
7413                                         * parsers.special_block_keyword
7414                                         * parsers.incomplete_special_tag_following
7415
7416 parsers.html_incomplete_close_tag = parsers.html_tag_closing_start
7417                                     * parsers.block_keyword
7418                                     * parsers.incomplete_tag_following
7419
7420 parsers.html_incomplete_close_special_tag = parsers.html_tag_closing_start
7421                                         * parsers.special_block_keyword
7422                                         * parsers.incomplete_tag_following
7423
7424 -- closing tags
7425 parsers.html_close_tag = parsers.html_tag_closing_start
7426                         * parsers.block_keyword
7427                         * parsers.html_tag_end
7428
7429 parsers.html_any_close_tag = parsers.html_tag_closing_start
7430                             * parsers.keyword
7431                             * parsers.html_tag_end
7432
7433 parsers.html_close_special_tag = parsers.html_tag_closing_start
7434                                 * parsers.special_block_keyword
7435                                 * parsers.html_tag_end
7436
7437 -- empty tags
7438 parsers.html_any_empty_inline_tag = parsers.html_tag_start
7439                                   * parsers.keyword
7440                                   * parsers.html_attribute^0
7441                                   * parsers.html_empty_tag_end
7442
7443 parsers.html_any_empty_tag = parsers.html_tag_start
7444                             * parsers.keyword
7445                             * parsers.html_non_newline_attribute^0
7446                             * parsers.optionalspace
7447                             * parsers.slash
7448                             * parsers.more
7449
7450 parsers.html_empty_tag = parsers.html_tag_start
7451                         * parsers.block_keyword
7452                         * parsers.html_attribute^0
7453                         * parsers.html_empty_tag_end
7454

```

```

7455 parsers.html_empty_special_tag = parsers.html_tag_start
7456                                 * parsers.special_block_keyword
7457                                 * parsers.html_attribute^0
7458                                 * parsers.html_empty_tag_end
7459
7460 parsers.html_incomplete_blocks = parsers.html_incomplete_open_tag
7461                                 + parsers.html_incomplete_open_special_tag
7462                                 + parsers.html_incomplete_close_tag
7463
7464 -- parse special html blocks
7465 parsers.html_blankline_ending_special_block_opening = (parsers.html_close_special_tag
7466                                                         + parsers.html_empty_special_tag
7467                                                         * #(parsers.optionalspace
7468                                                         * (parsers.newline + parsers.e
7469
7470 parsers.html_blankline_ending_special_block = parsers.html_blankline_ending_special_b
7471                                               * parsers.html_blankline_end_condition
7472
7473 parsers.html_special_block_opening = parsers.html_incomplete_open_special_tag
7474                                     - parsers.html_empty_special_tag
7475
7476 parsers.html_closing_special_block = parsers.html_special_block_opening
7477                                     * parsers.html_until_end(parsers.html_close_speci
7478
7479 parsers.html_special_block = parsers.html_blankline_ending_special_block
7480                               + parsers.html_closing_special_block
7481
7482 -- parse html blocks
7483 parsers.html_block_opening = parsers.html_incomplete_open_tag
7484                               + parsers.html_incomplete_close_tag
7485
7486 parsers.html_block = parsers.html_block_opening
7487                       * parsers.html_blankline_end_condition
7488
7489 -- parse any html blocks
7490 parsers.html_any_block_opening = (parsers.html_any_open_tag
7491                                   + parsers.html_any_close_tag
7492                                   + parsers.html_any_empty_tag)
7493                                   * #(parsers.optionalspace * (parsers.newline + parser
7494
7495 parsers.html_any_block = parsers.html_any_block_opening
7496                           * parsers.html_blankline_end_condition
7497
7498 parsers.html_inline_comment_full = parsers.html_comment_start
7499                                   * -P(">") * -P("->")
7500                                   * Cs((V("NoSoftLineBreakEndline") + parsers.any - P
")

```

```

7501             - parsers.nested_breaking_blank - parsers.html_
7502             * parsers.html_comment_end
7503
7504 parsers.html_inline_tags = parsers.html_inline_comment_full
7505             + parsers.html_any_empty_inline_tag
7506             + parsers.html_inline_instruction
7507             + parsers.html_inline_cdatasection
7508             + parsers.html_inline_declaration
7509             + parsers.html_any_open_inline_tag
7510             + parsers.html_any_close_tag
7511

```

### 3.1.5.6 Parsers Used for Markdown Tags and Links

```

7512 parsers.urlchar = parsers.anyescaped
7513             - parsers.newline
7514             - parsers.more
7515
7516 parsers.auto_link_scheme_part = parsers.alphanumeric
7517             + parsers.plus
7518             + parsers.period
7519             + parsers.dash
7520
7521 parsers.auto_link_scheme = parsers.letter
7522             * parsers.auto_link_scheme_part
7523             * parsers.auto_link_scheme_part^-30
7524
7525 parsers.absolute_uri = parsers.auto_link_scheme * parsers.colon
7526             * (parsers.any - parsers.spacing - parsers.less - parsers.more)
7527
7528 parsers.printable_characters = S(" !#$%&'*/+=?^_`{|}~-")
7529
7530 parsers.email_address_local_part_char = parsers.alphanumeric
7531             + parsers.printable_characters
7532
7533 parsers.email_address_local_part = parsers.email_address_local_part_char^1
7534
7535 parsers.email_address_dns_label = parsers.alphanumeric
7536             * (parsers.alphanumeric + parsers.dash)^-
7537             62
7538             * B(parsers.alphanumeric)
7539
7539 parsers.email_address_domain = parsers.email_address_dns_label
7540             * (parsers.period * parsers.email_address_dns_label)^0
7541
7542 parsers.email_address = parsers.email_address_local_part
7543             * parsers.at

```

```

7544             * parsers.email_address_domain
7545
7546 parsers.auto_link_url = parsers.less
7547             * C(parsers.absolute_uri)
7548             * parsers.more
7549
7550 parsers.auto_link_email = parsers.less
7551             * C(parsers.email_address)
7552             * parsers.more
7553
7554 parsers.auto_link_relative_reference = parsers.less
7555             * C(parsers.urlchar^1)
7556             * parsers.more
7557
7558 parsers.autolink = parsers.auto_link_url
7559             + parsers.auto_link_email
7560
7561 -- content in balanced brackets, parentheses, or quotes:
7562 parsers.bracketed = P{ parsers.lbracket
7563             * (( parsers.backslash / "\"" * parsers.rbracket
7564             + parsers.any - (parsers.lbracket
7565             + parsers.rbracket
7566             + parsers.blankline^2)
7567             ) + V(1))^0
7568             * parsers.rbracket }
7569
7570 parsers.inparens = P{ parsers.lparent
7571             * ((parsers.anyescaped - (parsers.lparent
7572             + parsers.rparent
7573             + parsers.blankline^2)
7574             ) + V(1))^0
7575             * parsers.rparent }
7576
7577 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
7578             * ((parsers.anyescaped - (parsers.squote
7579             + parsers.blankline^2)
7580             ) + V(1))^0
7581             * parsers.squote }
7582
7583 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
7584             * ((parsers.anyescaped - (parsers.dquote
7585             + parsers.blankline^2)
7586             ) + V(1))^0
7587             * parsers.dquote }
7588
7589 parsers.link_text = parsers.lbracket
7590             * Cs((parsers.alphanumeric^1

```

```

7591         + parsers.bracketed
7592         + parsers.inticks
7593         + parsers.autolink
7594         + V("InlineHtml")
7595         + ( parsers.backslash * parsers.backslash)
7596         + ( parsers.backslash * (parsers.lbracket + parsers.rbracket)
7597           + V("NoSoftLineBreakSpace")
7598           + V("NoSoftLineBreakEndline")
7599         + (parsers.any
7600           - (parsers.newline + parsers.lbracket + parsers.rbracket
7601             * parsers.rbracket
7602
7603 parsers.link_label_body = -#(parsers.sp * parsers.rbracket)
7604                       * #((parsers.any - parsers.rbracket)^-999 * parsers.rbracket)
7605                       * Cs((parsers.alphanumeric^1
7606                           + parsers.inticks
7607                           + parsers.autolink
7608                           + V("InlineHtml")
7609                           + ( parsers.backslash * parsers.backslash)
7610                           + ( parsers.backslash * (parsers.lbracket + parsers.rbracket)
7611                             + V("NoSoftLineBreakSpace")
7612                             + V("NoSoftLineBreakEndline")
7613                           + (parsers.any
7614                             - (parsers.newline + parsers.lbracket + parsers.rbracket
7615
7616 parsers.link_label    = parsers.lbracket
7617                       * parsers.link_label_body
7618                       * parsers.rbracket
7619
7620 parsers.inparens_url = P{ parsers.lparent
7621                       * ((parsers.anyescaped - (parsers.lparent
7622                                     + parsers.rparent
7623                                     + parsers.spacing)
7624                           ) + V(1))^0
7625                       * parsers.rparent }
7626
7627 -- url for markdown links, allowing nested brackets:
7628 parsers.url          = parsers.less * Cs((parsers.anyescaped
7629                                     - parsers.newline
7630                                     - parsers.less
7631                                     - parsers.more)^0)
7632                       * parsers.more
7633 + -parsers.less
7634 * Cs((parsers.inparens_url + (parsers.anyescaped
7635                                     - parsers.spacing
7636                                     - parsers.lparent
7637                                     - parsers.rparent))^1)

```



```

7638
7639 -- quoted text:
7640 parsers.title_s      = parsers.squote
7641                      * Cs((parsers.html_entities
7642                          + V("NoSoftLineBreakSpace")
7643                          + V("NoSoftLineBreakEndline")
7644                          + (parsers.anyescaped - parsers.newline - parsers.squote - p
7645                      * parsers.squote
7646
7647 parsers.title_d      = parsers.dquote
7648                      * Cs((parsers.html_entities
7649                          + V("NoSoftLineBreakSpace")
7650                          + V("NoSoftLineBreakEndline")
7651                          + (parsers.anyescaped - parsers.newline - parsers.dquote - p
7652                      * parsers.dquote
7653
7654 parsers.title_p      = parsers.lparent
7655                      * Cs((parsers.html_entities
7656                          + V("NoSoftLineBreakSpace")
7657                          + V("NoSoftLineBreakEndline")
7658                          + (parsers.anyescaped - parsers.newline - parsers.lparent -
7659                          - parsers.blankline^2))^0)
7660                      * parsers.rparent
7661
7662 parsers.title        = parsers.title_d + parsers.title_s + parsers.title_p
7663
7664 parsers.optionaltitle
7665                      = parsers.spnlc * parsers.title * parsers.spacechar^0
7666                      + Cc("")
7667

```

### 3.1.5.7 Helpers for Links and Link Reference Definitions

```

7668 -- parse a reference definition: [foo]: /bar "title"
7669 parsers.define_reference_parser = (parsers.check_trail / "") * parsers.link_label * p
7670                                * parsers.spnlc * parsers.url
7671                                * ( parsers.spnlc_sep * parsers.title * parsers.only_
7672                                + Cc("") * parsers.only_blank)

```

### 3.1.5.8 Inline Elements

```

7673 parsers.Inline      = V("Inline")
7674
7675 -- parse many p between starter and ender
7676 parsers.between = function(p, starter, ender)
7677   local ender2 = B(parsers.nonspacechar) * ender
7678   return (starter * #parsers.nonspacechar * Ct(p * (p - ender2)^0) * ender2)
7679 end

```

7680

### 3.1.5.9 Block Elements

```
7681 parsers.lineof = function(c)
7682     return (parsers.check_trail_no_rem * (P(c) * parsers.optionalspace)^3
7683           * (parsers.newline + parsers.eof))
7684 end
7685
7686 parsers.thematic_break_lines = parsers.lineof(parsers.asterisk)
7687                               + parsers.lineof(parsers.dash)
7688                               + parsers.lineof(parsers.underscore)
```

### 3.1.5.10 Headings

```
7689 -- parse Atx heading start and return level
7690 parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
7691                       * -parsers.hash / length
7692
7693 -- parse setext header ending and return level
7694 parsers.heading_level = parsers.nonindentspace * parsers.equal^1 * parsers.optionalsp
7695                       + parsers.nonindentspace * parsers.dash^1 * parsers.optionalspa
7696
7697 local function strip_atx_end(s)
7698     return s:gsub("%s+#+%s*\n$", "")
7699 end
7700
7701 parsers.atx_heading = parsers.check_trail_no_rem
7702                     * Cg(parsers.heading_start, "level")
7703                     * (C( parsers.optionalspace
7704                       * parsers.hash^0
7705                       * parsers.optionalspace
7706                       * parsers.newline)
7707                     + parsers.spacechar^1
7708                     * C(parsers.line))
```

## 3.1.6 Markdown Reader

This section documents the `reader` object, which implements the routines for parsing the markdown input. The object corresponds to the markdown reader object that was located in the `lunamark/reader/markdown.lua` file in the Lunamark Lua module.

The `reader.new` method creates and returns a new `TeX` reader object associated with the Lua interface options (see Section 2.1.3) `options` and with a writer object `writer`. When `options` are unspecified, it is assumed that an empty table was passed to the method.

The objects produced by the `reader.new` method expose instance methods and variables of their own. As a convention, I will refer to these *<member>*s as `reader-><member>`.

```
7709 M.reader = {}
7710 function M.reader.new(writer, options)
7711   local self = {}
```

Make the `writer` and `options` parameters available as `reader->writer` and `reader->options`, respectively, so that they are accessible from extensions.

```
7712   self.writer = writer
7713   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.

```
7714   self.parsers = {}
7715   (function(parsers)
7716     setmetatable(self.parsers, {
7717       __index = function (_, key)
7718         return parsers[key]
7719       end
7720     })
7721   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.

```
7722   local parsers = self.parsers
```

### 3.1.6.1 Top-Level Helper Functions

Define `reader->normalize_tag` as a function that normalizes a markdown reference tag by lowercasing it, and by collapsing any adjacent whitespace characters.

```
7723   function self.normalize_tag(tag)
7724     tag = util.ropetostring(tag)
7725     tag = tag:gsub("[\n\r\t]+", " ")
7726     tag = tag:gsub("^ ", ""):gsub(" $", "")
7727     tag = uni_algos.case.casefold(tag, true, false)
7728     return tag
7729   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.

```
7730   local function iterlines(s, f)
7731     local rope = lpeg.match(Ct((parsers.line / f)^1), s)
7732     return util.ropetostring(rope)
7733   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.

```
7734  if options.preserveTabs then
7735      self.expandtabs = function(s) return s end
7736  else
7737      self.expandtabs = function(s)
7738          if s:find("\t") then
7739              return iterlines(s, util.expand_tabs_in_line)
7740          else
7741              return s
7742          end
7743      end
7744  end
```

### 3.1.6.2 High-Level Parser Functions

Create a `reader->parser_functions` hash table that stores high-level parser functions. Define `reader->create_parser` as a function that will create a high-level parser function `reader->parser_functions.name`, that matches input using grammar `grammar`. If `toplevel` is true, the input is expected to come straight from the user, not from a recursive call, and will be preprocessed.

```
7745  self.parser_functions = {}
7746  self.create_parser = function(name, grammar, toplevel)
7747      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.

```
7748      if toplevel and options.stripIndent then
7749          local min_prefix_length, min_prefix = nil, ''
7750          str = iterlines(str, function(line)
7751              if lpeg.match(parsers.nonemptyline, line) == nil then
7752                  return line
7753              end
7754              line = util.expand_tabs_in_line(line)
7755              local prefix = lpeg.match(C(parsers.optionalspace), line)
7756              local prefix_length = #prefix
7757              local is_shorter = min_prefix_length == nil
7758              is_shorter = is_shorter or prefix_length < min_prefix_length
7759              if is_shorter then
7760                  min_prefix_length, min_prefix = prefix_length, prefix
7761              end
7762              return line
7763          end)
7764          str = str:gsub('^' .. min_prefix, '')
7765      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.

```
7766     if toplevel and (options.texComments or options.hybrid) then
7767         str = lpeg.match(Ct(parsers.commented_line^1), str)
7768         str = util.rope_to_string(str)
7769     end
7770     local res = lpeg.match(grammar(), str)
7771     if res == nil then
7772         error(format("%s failed on:\n%s", name, str:sub(1,20)))
7773     else
7774         return res
7775     end
7776 end
7777 end
7778
7779 self.create_parser("parse_blocks",
7780     function()
7781         return parsers.blocks
7782     end, true)
7783
7784 self.create_parser("parse_blocks_nested",
7785     function()
7786         return parsers.blocks_nested
7787     end, false)
7788
7789 self.create_parser("parse_inlines",
7790     function()
7791         return parsers.inlines
7792     end, false)
7793
7794 self.create_parser("parse_inlines_no_inline_note",
7795     function()
7796         return parsers.inlines_no_inline_note
7797     end, false)
7798
7799 self.create_parser("parse_inlines_no_html",
7800     function()
7801         return parsers.inlines_no_html
7802     end, false)
7803
7804 self.create_parser("parse_inlines_nbsp",
7805     function()
7806         return parsers.inlines_nbsp
7807     end, false)
7808 self.create_parser("parse_inlines_no_link_or_emphasis",
7809     function()
```

```

7810         return parsers.inlines_no_link_or_emphasis
7811     end, false)

```

### 3.1.6.3 Parsers Used for Indentation (local)

The following patterns represent basic building blocks of indented content.

```

7812 parsers.minimally_indented_blankline = parsers.check_minimal_indent * (parsers.blankline)
7813
7814 parsers.minimally_indented_block = parsers.check_minimal_indent * V("Block")
7815
7816 parsers.minimally_indented_block_or_paragraph = parsers.check_minimal_indent * V("Block")
7817
7818 parsers.minimally_indented_paragraph = parsers.check_minimal_indent * V("Paragraph")
7819
7820 parsers.minimally_indented_plain = parsers.check_minimal_indent * V("Plain")
7821
7822 parsers.minimally_indented_par_or_plain = parsers.minimally_indented_paragraph
7823         + parsers.minimally_indented_plain
7824
7825 parsers.minimally_indented_par_or_plain_no_blank = parsers.minimally_indented_paragraph
7826         - parsers.minimally_indented_blankline
7827
7828 parsers.minimally_indented_ref = parsers.check_minimal_indent * V("Reference")
7829
7830 parsers.minimally_indented_blank = parsers.check_minimal_indent * V("Blank")
7831
7832 parsers.conditionally_indented_blankline = parsers.check_minimal_blank_indent * (parsers.blankline)
7833
7834 parsers.minimally_indented_ref_or_block = parsers.minimally_indented_ref
7835         + parsers.minimally_indented_block
7836         - parsers.minimally_indented_blankline
7837
7838 parsers.minimally_indented_ref_or_block_or_paragraph = parsers.minimally_indented_ref
7839         + parsers.minimally_indented_block
7840         - parsers.minimally_indented_blankline
7841

```

The following pattern parses the properly indented content that follows the initial container start.

```

7842
7843 parsers.separator_loop = function(separated_block, paragraph, block_separator, paragraph_separator)
7844     return separated_block
7845         + block_separator
7846         * paragraph
7847         * separated_block
7848         + paragraph_separator
7849         * paragraph
7850 end

```

```

7851
7852 parsers.create_loop_body_pair = function(separated_block, paragraph, block_separato
7853     return {
7854         block = parsers.separator_loop(separated_block, paragraph, block_separator, blo
7855         par = parsers.separator_loop(separated_block, paragraph, block_separator, para
7856     }
7857 end
7858
7859 parsers.block_sep_group = function(blank)
7860     return blank^0 * parsers.eof
7861         + ( blank^2 / writer.paragraphsep
7862           + blank^0 / writer.interblocksep
7863         )
7864 end
7865
7866 parsers.par_sep_group = function(blank)
7867     return blank^0 * parsers.eof
7868         + blank^0 / writer.paragraphsep
7869 end
7870
7871 parsers.sep_group_no_output = function(blank)
7872     return blank^0 * parsers.eof
7873         + blank^0
7874 end
7875
7876 parsers.content_blank = parsers.minimally_indented_blankline
7877
7878 parsers.ref_or_block_separated = parsers.sep_group_no_output(parsers.content_blank
7879                             * ( parsers.minimally_indented_ref
7880                               - parsers.content_blank)
7881                             + parsers.block_sep_group(parsers.content_blank)
7882                             * ( parsers.minimally_indented_block
7883                               - parsers.content_blank)
7884
7885 parsers.loop_body_pair =
7886     parsers.create_loop_body_pair(parsers.ref_or_block_separated,
7887                                   parsers.minimally_indented_par_or_plain_no_blank,
7888                                   parsers.block_sep_group(parsers.content_blank),
7889                                   parsers.par_sep_group(parsers.content_blank))
7890
7891 parsers.content_loop = ( V("Block")
7892                         * parsers.loop_body_pair.block^0
7893                         + (V("Paragraph") + V("Plain")))
7894                         * parsers.ref_or_block_separated
7895                         * parsers.loop_body_pair.block^0
7896                         + (V("Paragraph") + V("Plain")))
7897                         * parsers.loop_body_pair.par^0)

```

```

7898             * parsers.content_blank^0
7899
7900 parsers.indented_content = function()
7901   return Ct( (V("Reference") + (parsers.blankline / ""))
7902             * parsers.content_blank^0
7903             * parsers.check_minimal_indent
7904             * parsers.content_loop
7905             + (V("Reference") + (parsers.blankline / ""))
7906             * parsers.content_blank^0
7907             + parsers.content_loop)
7908 end
7909
7910 parsers.add_indent = function(pattern, name, breakable)
7911   return Cg(Cmt( Cb("indent_info")
7912                * Ct(pattern)
7913                * (#parsers.linechar * Cc(false) + Cc(true)) -- check if starter is
7914                * Cc(name)
7915                * Cc(breakable),
7916                process_starter_indent), "indent_info")
7917 end
7918

```

### 3.1.6.4 Parsers Used for Markdown Lists (local)

```

7919 if options.hashEnumerators then
7920   parsers.dig = parsers.digit + parsers.hash
7921 else
7922   parsers.dig = parsers.digit
7923 end
7924
7925 parsers.enumerator = function(delimiter_type, interrupting)
7926   local delimiter_range
7927   local allowed_end
7928   if interrupting then
7929     delimiter_range = P("1")
7930     allowed_end = C(parsers.spacechar^1) * #parsers.linechar
7931   else
7932     delimiter_range = parsers.dig * parsers.dig^-8
7933     allowed_end = C(parsers.spacechar^1) + #(parsers.newline + parsers.eof)
7934   end
7935
7936   return parsers.check_trail
7937     * Ct(C(delimiter_range) * C(delimiter_type))
7938     * allowed_end
7939 end
7940
7941 parsers.starter = parsers.bullet(parsers.dash)

```



```

7942         + parsers.bullet(parsers.asterisk)
7943         + parsers.bullet(parsers.plus)
7944         + parsers.enumerator(parsers.period)
7945         + parsers.enumerator(parsers.rparent)
7946

```

### 3.1.6.5 Parsers Used for Blockquotes (local)

```

7947 parsers.blockquote_start = parsers.check_trail * C(parsers.more) * C(parsers.space)
7948
7949 parsers.blockquote_body = parsers.add_indent(parsers.blockquote_start, "bq", true)
7950                         * parsers.indented_content()
7951                         * remove_indent("bq")
7952
7953 if not options.breakableBlockquotes then
7954     parsers.blockquote_body = parsers.add_indent(parsers.blockquote_start, "bq", false)
7955     * parsers.indented_content()
7956     * remove_indent("bq")
7957 end

```

### 3.1.6.6 Helpers for Emphasis and Strong Emphasis (local)

Parse the content of a table `content_part` with links, images and emphasis disabled.

```

7958 local function parse_content_part(content_part)
7959     local rope = util.rope_to_string(content_part)
7960     local parsed = self.parser_functions.parse_inlines_no_link_or_emphasis(rope)
7961     parsed.indent_info = nil
7962     return parsed
7963 end
7964

```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

7965 local function collect_emphasis_content(t, opening_index, closing_index)
7966     local content = {}
7967
7968     local content_part = {}
7969     for i = opening_index, closing_index do
7970         local value = t[i]
7971
7972         if value.rendered ~= nil then
7973             content[#content + 1] = parse_content_part(content_part)
7974             content_part = {}
7975             content[#content + 1] = value.rendered
7976             value.rendered = nil
7977         else
7978             if value.type == "delimiter" and value.element == "emphasis" then

```

```

7979         if value.is_active then
7980             content_part[#content_part + 1] = string.rep(value.character, value.curren
7981         end
7982     else
7983         content_part[#content_part + 1] = value.content
7984     end
7985     value.content = ''
7986     value.is_active = false
7987 end
7988 end
7989
7990 if next(content_part) ~= nil then
7991     content[#content + 1] = parse_content_part(content_part)
7992 end
7993
7994 return content
7995 end
7996

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as emphasis.

```

7997 local function fill_emph(t, opening_index, closing_index)
7998     local content = collect_emphasis_content(t, opening_index + 1, closing_index - 1)
7999     t[opening_index + 1].is_active = true
8000     t[opening_index + 1].rendered = writer.emphasis(content)
8001 end
8002

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as strong emphasis.

```

8003 local function fill_strong(t, opening_index, closing_index)
8004     local content = collect_emphasis_content(t, opening_index + 1, closing_index - 1)
8005     t[opening_index + 1].is_active = true
8006     t[opening_index + 1].rendered = writer.strong(content)
8007 end
8008

```

Check whether the opening delimiter `opening_delimiter` and closing delimiter `closing_delimiter` break rule three together.

```

8009 local function breaks_three_rule(opening_delimiter, closing_delimiter)
8010     return (opening_delimiter.is_closing or closing_delimiter.is_opening) and
8011         ((opening_delimiter.original_count + closing_delimiter.original_count) % 3 == 0)
8012         (opening_delimiter.original_count % 3 ~= 0 or closing_delimiter.original_count
8013     end
8014

```

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`.

```

8015 local function find_emphasis_opener(t, bottom_index, latest_index, character, closing_delimiter)
8016     for i = latest_index, bottom_index, -1 do
8017         local value = t[i]
8018         if value.is_active and
8019            value.is_opening and
8020            value.type == "delimiter" and
8021            value.element == "emphasis" and
8022            (value.character == character) and
8023            (value.current_count > 0) then
8024             if not breaks_three_rule(value, closing_delimiter) then
8025                 return i
8026             end
8027         end
8028     end
8029 end
8030

```

Iterate over the delimiters in the delimiter table `t`, producing emphasis or strong emphasis macros.

```

8031 local function process_emphasis(t, opening_index, closing_index)
8032     for i = opening_index, closing_index do
8033         local value = t[i]
8034         if value.type == "delimiter" and value.element == "emphasis" then
8035             local delimiter_length = string.len(value.content)
8036             value.character = string.sub(value.content, 1, 1)
8037             value.current_count = delimiter_length
8038             value.original_count = delimiter_length
8039         end
8040     end
8041
8042     local openers_bottom = {
8043         ['*'] = {
8044             [true] = {opening_index, opening_index, opening_index},
8045             [false] = {opening_index, opening_index, opening_index}
8046         },
8047         ['_'] = {
8048             [true] = {opening_index, opening_index, opening_index},
8049             [false] = {opening_index, opening_index, opening_index}
8050         }
8051     }
8052
8053     local current_position = opening_index
8054     local max_position = closing_index
8055

```

```

8056 while current_position <= max_position do
8057     local value = t[current_position]
8058
8059     if value.type ~= "delimiter" or
8060        value.element ~= "emphasis" or
8061        not value.is_active or
8062        not value.is_closing or
8063        (value.current_count <= 0) then
8064         current_position = current_position + 1
8065         goto continue
8066     end
8067
8068     local character = value.character
8069     local is_opening = value.is_opening
8070     local closing_length_modulo_three = value.original_count % 3
8071
8072     local current_openers_bottom = openers_bottom[character][is_opening][closing_le
8073
8074     local opener_position = find_emphasis_opener(t, current_openers_bottom, current
8075
8076     if (opener_position == nil) then
8077         openers_bottom[character][is_opening][closing_length_modulo_three + 1] = curr
8078         current_position = current_position + 1
8079         goto continue
8080     end
8081
8082     local opening_delimiter = t[opener_position]
8083
8084     local current_opening_count = opening_delimiter.current_count
8085     local current_closing_count = t[current_position].current_count
8086
8087     if (current_opening_count >= 2) and (current_closing_count >= 2) then
8088         opening_delimiter.current_count = current_opening_count - 2
8089         t[current_position].current_count = current_closing_count - 2
8090         fill_strong(t, opener_position, current_position)
8091     else
8092         opening_delimiter.current_count = current_opening_count - 1
8093         t[current_position].current_count = current_closing_count - 1
8094         fill_emph(t, opener_position, current_position)
8095     end
8096
8097     ::continue::
8098 end
8099 end
8100
8101 local cont = lpeg.R("\128\191") -- continuation byte
8102

```

Match a UTF-8 character of byte length `n`.

```
8103 local function utf8_by_byte_count(n)
8104   if (n == 1) then
8105     return lpeg.R("\0\127")
8106   end
8107   if (n == 2) then
8108     return lpeg.R("\194\223") * cont
8109   end
8110   if (n == 3) then
8111     return lpeg.R("\224\239") * cont * cont
8112   end
8113   if (n == 4) then
8114     return lpeg.R("\240\244") * cont * cont * cont
8115   end
8116 end
```

Check if there is a character of a type `chartype` between the start position `start_pos` and end position `end_pos` in a string `s` relative to current index `i`.

```
8117 local function check_unicode_type(s, i, start_pos, end_pos, chartype)
8118   local c
8119   local char_length
8120   for pos = start_pos, end_pos, 1 do
8121     if (start_pos < 0) then
8122       char_length = -pos
8123     else
8124       char_length = pos + 1
8125     end
8126
8127     if (chartype == "punctuation") then
8128       if lpeg.match(parsers.punctuation[char_length], s, i+pos) then
8129         return i
8130       end
8131     else
8132       c = lpeg.match({ C(utf8_by_byte_count(char_length)) }, s, i+pos)
8133       if (c ~= nil) and (unicode.utf8.match(c, chartype)) then
8134         return i
8135       end
8136     end
8137   end
8138 end
8139
8140 local function check_preceding_unicode_punctuation(s, i)
8141   return check_unicode_type(s, i, -4, -1, "punctuation")
8142 end
8143
8144 local function check_preceding_unicode_whitespace(s, i)
8145   return check_unicode_type(s, i, -4, -1, "%s")
```

```

8146 end
8147
8148 local function check_following_unicode_punctuation(s, i)
8149     return check_unicode_type(s, i, 0, 3, "punctuation")
8150 end
8151
8152 local function check_following_unicode_whitespace(s, i)
8153     return check_unicode_type(s, i, 0, 3, "%s")
8154 end
8155
8156 parsers.unicode_preceding_punctuation = B(parsers.escapable)
8157     + Cmt(parsers.succeed, check_preceding_unicode)
8158
8159 parsers.unicode_preceding_whitespace = Cmt(parsers.succeed, check_preceding_unicode)
8160
8161 parsers.unicode_following_punctuation = #parsers.escapable
8162     + Cmt(parsers.succeed, check_following_unicode)
8163
8164 parsers.unicode_following_whitespace = Cmt(parsers.succeed, check_following_unicode)
8165
8166 parsers.delimiter_run = function(character)
8167     return (B(parsers.backslash * character) + ~B(character))
8168         * character~1
8169         * ~#character
8170 end
8171
8172 parsers.left_flanking_delimiter_run = function(character)
8173     return (B( parsers.any)
8174         * (parsers.unicode_preceding_punctuation + parsers.unicode_preceding_whitespace)
8175         + ~B(parsers.any))
8176     * parsers.delimiter_run(character)
8177     * parsers.unicode_following_punctuation
8178     + parsers.delimiter_run(character)
8179     * ~#(parsers.unicode_following_punctuation + parsers.unicode_following_whitespace)
8180     + parsers.eof)
8181 end
8182
8183 parsers.right_flanking_delimiter_run = function(character)
8184     return parsers.unicode_preceding_punctuation
8185     * parsers.delimiter_run(character)
8186     * (parsers.unicode_following_punctuation + parsers.unicode_following_whitespace)
8187     + parsers.eof)
8188     + (B(parsers.any)
8189     * ~(parsers.unicode_preceding_punctuation + parsers.unicode_preceding_whitespace)
8190     * parsers.delimiter_run(character)
8191 end
8192

```

```

8193 if options.underscores then
8194     parsers.emph_start = parsers.left_flanking_delimiter_run(parsers.asterisk)
8195                       + (-#parsers.right_flanking_delimiter_run(parsers.underscore)
8196                         + (parsers.unicode_preceding_punctuation
8197                           * #parsers.right_flanking_delimiter_run(parsers.underscore)
8198                           * parsers.left_flanking_delimiter_run(parsers.underscore)
8199
8200     parsers.emph_end = parsers.right_flanking_delimiter_run(parsers.asterisk)
8201                     + (-#parsers.left_flanking_delimiter_run(parsers.underscore)
8202                       + #(parsers.left_flanking_delimiter_run(parsers.underscore)
8203                         * parsers.unicode_following_punctuation))
8204                     * parsers.right_flanking_delimiter_run(parsers.underscore)
8205 else
8206     parsers.emph_start = parsers.left_flanking_delimiter_run(parsers.asterisk)
8207
8208     parsers.emph_end = parsers.right_flanking_delimiter_run(parsers.asterisk)
8209 end
8210
8211 parsers.emph_capturing_open_and_close = #parsers.emph_start * #parsers.emph_end
8212                                       * Ct( Cg(Cc("delimiter"), "type")
8213                                             * Cg(Cc("emphasis"), "element")
8214                                             * Cg(C(parsers.emph_start), "content")
8215                                             * Cg(Cc(true), "is_opening")
8216                                             * Cg(Cc(true), "is_closing"))
8217
8218 parsers.emph_capturing_open = Ct( Cg(Cc("delimiter"), "type")
8219                                  * Cg(Cc("emphasis"), "element")
8220                                  * Cg(C(parsers.emph_start), "content")
8221                                  * Cg(Cc(true), "is_opening")
8222                                  * Cg(Cc(false), "is_closing"))
8223
8224 parsers.emph_capturing_close = Ct( Cg(Cc("delimiter"), "type")
8225                                   * Cg(Cc("emphasis"), "element")
8226                                   * Cg(C(parsers.emph_end), "content")
8227                                   * Cg(Cc(false), "is_opening")
8228                                   * Cg(Cc(true), "is_closing"))
8229
8230 parsers.emph_open_or_close = parsers.emph_capturing_open_and_close
8231                             + parsers.emph_capturing_open
8232                             + parsers.emph_capturing_close
8233
8234 parsers.emph_open = parsers.emph_capturing_open_and_close
8235                  + parsers.emph_capturing_open
8236
8237 parsers.emph_close = parsers.emph_capturing_open_and_close
8238                   + parsers.emph_capturing_close
8239

```

### 3.1.6.7 Helpers for Links and Link Reference Definitions (local)

```
8240 -- List of references defined in the document
8241 local references
8242
8243 -- List of note references defined in the document
8244 parsers.rawnotes = {}
8245
```

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.

```
8246 function self.register_link(_, tag, url, title,
8247                             attributes)
8248     local normalized_tag = self.normalize_tag(tag)
8249     if references[normalized_tag] == nil then
8250         references[normalized_tag] = {
8251             url = url,
8252             title = title,
8253             attributes = attributes
8254         }
8255     end
8256     return ""
8257 end
8258
```

The `reader->lookup_reference` method looks up a reference with link label `tag`.

```
8259 function self.lookup_reference(tag)
8260     return references[self.normalize_tag(tag)]
8261 end
8262
```

The `reader->lookup_note_reference` method looks up a note reference with label `tag`.

```
8263 function self.lookup_note_reference(tag)
8264     return parsers.rawnotes[self.normalize_tag(tag)]
8265 end
8266
8267 parsers.title_s_direct_ref = parsers.squote
8268                             * Cs((parsers.html_entities
8269                                 + (parsers.anyescaped - parsers.squote - parsers.bl
8270                                 * parsers.squote
8271
8272 parsers.title_d_direct_ref = parsers.dquote
8273                             * Cs((parsers.html_entities
8274                                 + (parsers.anyescaped - parsers.dquote - parsers.bl
8275                             * parsers.dquote
8276
8277 parsers.title_p_direct_ref = parsers.lparent
```



```

8278         * Cs((parsers.html_entities
8279             + (parsers.anyescaped - parsers.lparent - parsers.r
8280             * parsers.rparent
8281
8282     parsers.title_direct_ref = parsers.title_s_direct_ref
8283         + parsers.title_d_direct_ref
8284         + parsers.title_p_direct_ref
8285
8286     parsers.inline_direct_ref_inside = parsers.lparent * parsers.spnl
8287         * Cg(parsers.url + Cc(""), "url")
8288         * parsers.spnl
8289         * Cg(parsers.title_direct_ref + Cc(""), "title")
8290         * parsers.spnl * parsers.rparent
8291
8292     parsers.inline_direct_ref = parsers.lparent * parsers.spnlc
8293         * Cg(parsers.url + Cc(""), "url")
8294         * parsers.spnlc
8295         * Cg(parsers.title + Cc(""), "title")
8296         * parsers.spnlc * parsers.rparent
8297
8298     parsers.empty_link = parsers.lbracket
8299         * parsers.rbracket
8300
8301     parsers.inline_link = parsers.link_text
8302         * parsers.inline_direct_ref
8303
8304     parsers.full_link = parsers.link_text
8305         * parsers.link_label
8306
8307     parsers.shortcut_link = parsers.link_label
8308         * -(parsers.empty_link + parsers.link_label)
8309
8310     parsers.collapsed_link = parsers.link_label
8311         * parsers.empty_link
8312
8313     parsers.image_opening = #(parsers.exclamation * parsers.inline_link)
8314         * Cg(Cc("inline"), "link_type")
8315         + #(parsers.exclamation * parsers.full_link)
8316         * Cg(Cc("full"), "link_type")
8317         + #(parsers.exclamation * parsers.collapsed_link)
8318         * Cg(Cc("collapsed"), "link_type")
8319         + #(parsers.exclamation * parsers.shortcut_link)
8320         * Cg(Cc("shortcut"), "link_type")
8321         + #(parsers.exclamation * parsers.empty_link)
8322         * Cg(Cc("empty"), "link_type")
8323
8324     parsers.link_opening = #parsers.inline_link

```

```

8325         * Cg(Cc("inline"), "link_type")
8326     + #parsers.full_link
8327     * Cg(Cc("full"), "link_type")
8328     + #parsers.collapsed_link
8329     * Cg(Cc("collapsed"), "link_type")
8330     + #parsers.shortcut_link
8331     * Cg(Cc("shortcut"), "link_type")
8332     + #parsers.empty_link
8333     * Cg(Cc("empty_link"), "link_type")
8334     + #parsers.link_text
8335     * Cg(Cc("link_text"), "link_type")
8336
8337     parsers.note_opening = #(parsers.circumflex * parsers.shortcut_link)
8338     * Cg(Cc("note_inline"), "link_type")
8339
8340     parsers.raw_note_opening = #( parsers.lbracket
8341     * parsers.circumflex
8342     * parsers.link_label_body
8343     * parsers.rbracket)
8344     * Cg(Cc("raw_note"), "link_type")
8345
8346     local inline_note_element = Cg(Cc("note"), "element")
8347     * parsers.note_opening
8348     * Cg(parsers.circumflex * parsers.lbracket, "content")
8349
8350     local image_element = Cg(Cc("image"), "element")
8351     * parsers.image_opening
8352     * Cg(parsers.exclamation * parsers.lbracket, "content")
8353
8354     local note_element = Cg(Cc("note"), "element")
8355     * parsers.raw_note_opening
8356     * Cg(parsers.lbracket * parsers.circumflex, "content")
8357
8358     local link_element = Cg(Cc("link"), "element")
8359     * parsers.link_opening
8360     * Cg(parsers.lbracket, "content")
8361
8362     local opening_elements = parsers.fail
8363
8364     if options.inlineNotes then
8365         opening_elements = opening_elements + inline_note_element
8366     end
8367
8368     opening_elements = opening_elements + image_element
8369
8370     if options.notes then
8371         opening_elements = opening_elements + note_element

```

```

8372 end
8373
8374 opening_elements = opening_elements + link_element
8375
8376 parsers.link_image_opening = Ct( Cg(Cc("delimiter"), "type")
8377     * Cg(Cc(true), "is_opening")
8378     * Cg(Cc(false), "is_closing")
8379     * opening_elements)
8380
8381 parsers.link_image_closing = Ct( Cg(Cc("delimiter"), "type")
8382     * Cg(Cc("link"), "element")
8383     * Cg(Cc(false), "is_opening")
8384     * Cg(Cc(true), "is_closing")
8385     * ( Cg(Cc(true), "is_direct")
8386     * Cg(parsers.rbracket * #parsers.inline_direct_re
8387     + Cg(Cc(false), "is_direct")
8388     * Cg(parsers.rbracket, "content")))
8389
8390 parsers.link_image_open_or_close = parsers.link_image_opening
8391     + parsers.link_image_closing
8392
8393 if options.html then
8394     parsers.link_emph_precedence = parsers.inticks
8395     + parsers.autolink
8396     + parsers.html_inline_tags
8397 else
8398     parsers.link_emph_precedence = parsers.inticks
8399     + parsers.autolink
8400 end
8401
8402 parsers.link_and_emph_endline = parsers.newline
8403     * ((parsers.check_minimal_indent
8404     * -V("EndlineExceptions")
8405     + parsers.check_optional_indent
8406     * -V("EndlineExceptions")
8407     * -parsers.starter) / "")
8408     * parsers.spacechar^0 / "\n"
8409
8410 parsers.link_and_emph_content = Ct( Cg(Cc("content"), "type")
8411     * Cg(Cs(( parsers.link_emph_precedence
8412     + parsers.backslash * parsers.any
8413     + parsers.link_and_emph_endline
8414     + (parsers.linechar
8415     - parsers.blankline^2
8416     - parsers.link_image_open_or_close
8417     - parsers.emph_open_or_close))^0), "con
8418

```

```

8419 parsers.link_and_emph_table = (parsers.link_image_opening + parsers.emph_open)
8420                               * parsers.link_and_emph_content
8421                               * ((parsers.link_image_open_or_close + parsers.emph_ope
8422                               * parsers.link_and_emph_content)^1
8423

```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

8424 local function collect_link_content(t, opening_index, closing_index)
8425     local content = {}
8426     for i = opening_index, closing_index do
8427         content[#content + 1] = t[i].content
8428     end
8429     return util.rope_to_string(content)
8430 end
8431

```

Look for the closest potential link opener in the delimiter table `t` in the range from `bottom_index` to `latest_index`.

```

8432 local function find_link_opener(t, bottom_index, latest_index)
8433     for i = latest_index, bottom_index, -1 do
8434         local value = t[i]
8435         if value.type == "delimiter" and
8436            value.is_opening and
8437            (value.element == "link" or value.element == "image" or value.element == "no
8438            and not value.removed then
8439             if value.is_active then
8440                 return i
8441             end
8442             value.removed = true
8443             return nil
8444         end
8445     end
8446 end
8447

```

Find the position of a delimiter that closes a full link after an an index `latest_index` in the delimiter table `t`.

```

8448 local function find_next_link_closing_index(t, latest_index)
8449     for i = latest_index, #t do
8450         local value = t[i]
8451         if value.is_closing and
8452            value.element == "link" and
8453            not value.removed then
8454             return i
8455         end
8456     end
8457 end

```

8458

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.

```
8459 local function disable_previous_link_openers(t, opening_index)
8460   if t[opening_index].element == "image" then
8461     return
8462   end
8463
8464   for i = opening_index, 1, -1 do
8465     local value = t[i]
8466     if value.is_active and
8467        value.type == "delimiter" and
8468        value.is_opening and
8469        value.element == "link" then
8470       value.is_active = false
8471     end
8472   end
8473 end
8474
```

Disable the delimiters between the `opening_index` and `closing_index` in the delimiter table `t` by marking them inactive with the `is_active` property.

```
8475 local function disable_range(t, opening_index, closing_index)
8476   for i = opening_index, closing_index do
8477     local value = t[i]
8478     if value.is_active then
8479       value.is_active = false
8480       if value.type == "delimiter" then
8481         value.removed = true
8482       end
8483     end
8484   end
8485 end
8486
```

Clear the parsed content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
8487 local function delete_parsed_content_in_range(t, opening_index, closing_index)
8488   for i = opening_index, closing_index do
8489     t[i].rendered = nil
8490   end
8491 end
8492
```

Clear the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
8493 local function empty_content_in_range(t, opening_index, closing_index)
```

```

8494     for i = opening_index, closing_index do
8495         t[i].content = ''
8496     end
8497 end
8498

```

Join the attributes from the link reference definition `reference_attributes` with the link's own attributes `own_attributes`.

```

8499 local function join_attributes(reference_attributes, own_attributes)
8500     local merged_attributes = {}
8501     for _, attribute in ipairs(reference_attributes or {}) do
8502         table.insert(merged_attributes, attribute)
8503     end
8504     for _, attribute in ipairs(own_attributes or {}) do
8505         table.insert(merged_attributes, attribute)
8506     end
8507     if next(merged_attributes) == nil then
8508         merged_attributes = nil
8509     end
8510     return merged_attributes
8511 end
8512

```

Parse content between two delimiters in the delimiter table `t`. Produce the respective link and image macros.

```

8513 local function render_link_or_image(t, opening_index, closing_index, content_end_index)
8514     process_emphasis(t, opening_index, content_end_index)
8515     local mapped = collect_emphasis_content(t, opening_index + 1, content_end_index - 1)
8516
8517     local rendered = {}
8518     if (t[opening_index].element == "link") then
8519         rendered = writer.link(mapped, reference.url, reference.title, reference.attributes)
8520     end
8521
8522     if (t[opening_index].element == "image") then
8523         rendered = writer.image(mapped, reference.url, reference.title, reference.attributes)
8524     end
8525
8526     if (t[opening_index].element == "note") then
8527         if (t[opening_index].link_type == "note_inline") then
8528             rendered = writer.note(mapped)
8529         end
8530         if (t[opening_index].link_type == "raw_note") then
8531             rendered = writer.note(reference)
8532         end
8533     end
8534
8535     t[opening_index].rendered = rendered

```

```

8536     delete_parsed_content_in_range(t, opening_index + 1, closing_index)
8537     empty_content_in_range(t, opening_index, closing_index)
8538     disable_previous_link_openers(t, opening_index)
8539     disable_range(t, opening_index, closing_index)
8540 end
8541

```

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.

```

8542 local function resolve_inline_following_content(t, closing_index, match_reference,
8543     local content = ""
8544     for i = closing_index + 1, #t do
8545         content = content .. t[i].content
8546     end
8547
8548     local matching_content = parsers.succeed
8549
8550     if match_reference then
8551         matching_content = matching_content * parsers.inline_direct_ref_inside
8552     end
8553
8554     if match_link_attributes then
8555         matching_content = matching_content * Cg(Ct(parsers.attributes^-
1), "attributes")
8556     end
8557
8558     local matched = lpeg.match(Ct(matching_content * Cg(Cp(), "end_position")), content)
8559
8560     local matched_count = matched.end_position - 1
8561     for i = closing_index + 1, #t do
8562         local value = t[i]
8563
8564         local chars_left = matched_count
8565         matched_count = matched_count - #value.content
8566
8567         if matched_count <= 0 then
8568             value.content = value.content:sub(chars_left + 1)
8569             break
8570         end
8571
8572         value.content = ''
8573         value.is_active = false
8574     end
8575
8576     local attributes = matched.attributes
8577     if attributes == nil or next(attributes) == nil then

```

```

8578     attributes = nil
8579     end
8580
8581     return {
8582       url = matched.url or "",
8583       title = matched.title or "",
8584       attributes = attributes
8585     }
8586   end
8587

```

Resolve an inline link `[a](b "c")` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Here, compared to other types of links, no reference definition is needed.

```

8588   local function resolve_inline_link(t, opening_index, closing_index)
8589     local inline_content = resolve_inline_following_content(t, closing_index, true, t)
8590     render_link_or_image(t, opening_index, closing_index, closing_index, inline_content)
8591   end
8592

```

Resolve an inline note `^[a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`.

```

8593   local function resolve_note_inline_link(t, opening_index, closing_index)
8594     local inline_content = resolve_inline_following_content(t, closing_index, false, t)
8595     render_link_or_image(t, opening_index, closing_index, closing_index, inline_content)
8596   end
8597

```

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.

```

8598   local function resolve_shortcut_link(t, opening_index, closing_index)
8599     local content = collect_link_content(t, opening_index + 1, closing_index - 1)
8600     local r = self.lookup_reference(content)
8601
8602     if r then
8603       local inline_content = resolve_inline_following_content(t, closing_index, false, t)
8604       r.attributes = join_attributes(r.attributes, inline_content.attributes)
8605       render_link_or_image(t, opening_index, closing_index, closing_index, r)
8606     end
8607   end
8608

```

Resolve a note `[^a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `a` is not found in the rawnotes.

```

8609   local function resolve_raw_note_link(t, opening_index, closing_index)
8610     local content = collect_link_content(t, opening_index + 1, closing_index - 1)
8611     local r = self.lookup_note_reference(content)

```



```

8612
8613     if r then
8614         local parsed_ref = self.parser_functions.parse_blocks_nested(r)
8615         render_link_or_image(t, opening_index, closing_index, closing_index, parsed_ref)
8616     end
8617 end
8618

```

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.

```

8619 local function resolve_full_link(t, opening_index, closing_index)
8620     local next_link_closing_index = find_next_link_closing_index(t, closing_index + 4)
8621     local next_link_content = collect_link_content(t, closing_index + 3, next_link_closing_index)
8622     local r = self.lookup_reference(next_link_content)
8623
8624     if r then
8625         local inline_content = resolve_inline_following_content(t, next_link_closing_index,
8626                                                                 t.match_link_attributes)
8627         r.attributes = join_attributes(r.attributes, inline_content.attributes)
8628         render_link_or_image(t, opening_index, next_link_closing_index, closing_index, r)
8629     end
8630 end
8631

```

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.

```

8632 local function resolve_collapsed_link(t, opening_index, closing_index)
8633     local next_link_closing_index = find_next_link_closing_index(t, closing_index + 4)
8634     local content = collect_link_content(t, opening_index + 1, closing_index - 1)
8635     local r = self.lookup_reference(content)
8636
8637     if r then
8638         local inline_content = resolve_inline_following_content(t, closing_index, false)
8639         r.attributes = join_attributes(r.attributes, inline_content.attributes)
8640         render_link_or_image(t, opening_index, next_link_closing_index, closing_index, r)
8641     end
8642 end
8643

```

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.

```

8644 local function process_links_and_emphasis(t)
8645     for _,value in ipairs(t) do
8646         value.is_active = true
8647     end

```

```

8648
8649   for i,value in ipairs(t) do
8650       if not value.is_closing or
8651           value.type ~= "delimiter" or
8652           not (value.element == "link" or value.element == "image" or value.element ==
8653               goto continue
8654           end
8655
8656           local opener_position = find_link_opener(t, 1, i - 1)
8657           if (opener_position == nil) then
8658               goto continue
8659           end
8660
8661           local opening_delimiter = t[opener_position]
8662           opening_delimiter.removed = true
8663
8664           local link_type = opening_delimiter.link_type
8665
8666           if (link_type == "inline") then
8667               resolve_inline_link(t, opener_position, i)
8668           end
8669           if (link_type == "shortcut") then
8670               resolve_shortcut_link(t, opener_position, i)
8671           end
8672           if (link_type == "full") then
8673               resolve_full_link(t, opener_position, i)
8674           end
8675           if (link_type == "collapsed") then
8676               resolve_collapsed_link(t, opener_position, i)
8677           end
8678           if (link_type == "note_inline") then
8679               resolve_note_inline_link(t, opener_position, i)
8680           end
8681           if (link_type == "raw_note") then
8682               resolve_raw_note_link(t, opener_position, i)
8683           end
8684
8685           ::continue::
8686       end
8687
8688       t[#t].content = t[#t].content:gsub("%s*$", "")
8689
8690       process_emphasis(t, 1, #t)
8691       local final_result = collect_emphasis_content(t, 1, #t)
8692       return final_result
8693   end
8694

```

```

8695 function self.defer_link_and_emphasis_processing(delimiter_table)
8696     return writer.defer_call(function()
8697         return process_links_and_emphasis(delimiter_table)
8698     end)
8699 end
8700

```

### 3.1.6.8 Inline Elements (local)

```

8701 parsers.Str      = (parsers.normalchar * (parsers.normalchar + parsers.at)^0)
8702                 / writer.string
8703
8704 parsers.Symbol   = (parsers.backtick^1 + V("SpecialChar"))
8705                 / writer.string
8706
8707 parsers.Ellipsis = P("...") / writer.ellipsis
8708
8709 parsers.Smart    = parsers.Ellipsis
8710
8711 parsers.Code     = parsers.inticks / writer.code
8712
8713 if options.blankBeforeBlockquote then
8714     parsers.bqstart = parsers.fail
8715 else
8716     parsers.bqstart = parsers.blockquote_start
8717 end
8718
8719 if options.blankBeforeHeading then
8720     parsers.headerstart = parsers.fail
8721 else
8722     parsers.headerstart = parsers.atx_heading
8723 end
8724
8725 if options.blankBeforeList then
8726     parsers.interrupting_bullets = parsers.fail
8727     parsers.interrupting_enumerators = parsers.fail
8728 else
8729     parsers.interrupting_bullets = parsers.bullet(parsers.dash, true)
8730                                 + parsers.bullet(parsers.asterisk, true)
8731                                 + parsers.bullet(parsers.plus, true)
8732
8733     parsers.interrupting_enumerators = parsers.enumerator(parsers.period, true)
8734                                       + parsers.enumerator(parsers.rparent, true)
8735 end
8736
8737 if options.html then
8738     parsers.html_interrupting = parsers.check_trail

```

```

8739         * ( parsers.html_incomplete_open_tag
8740           + parsers.html_incomplete_close_tag
8741           + parsers.html_incomplete_open_special_tag
8742           + parsers.html_comment_start
8743           + parsers.html_cdatasection_start
8744           + parsers.html_declaration_start
8745           + parsers.html_instruction_start
8746           - parsers.html_close_special_tag
8747           - parsers.html_empty_special_tag)
8748     else
8749       parsers.html_interrupting = parsers.fail
8750     end
8751
8752     parsers.EndlineExceptions
8753       = parsers.blankline -- paragraph break
8754       + parsers.eof      -- end of document
8755       + parsers.bqstart
8756       + parsers.thematic_break_lines
8757       + parsers.interrupting_bullets
8758       + parsers.interrupting_enumerators
8759       + parsers.headerstart
8760       + parsers.html_interrupting
8761
8762     parsers.NoSoftLineBreakEndlineExceptions = parsers.EndlineExceptions
8763
8764     parsers.endline = parsers.newline
8765       * (parsers.check_minimal_indent
8766         * -V("EndlineExceptions")
8767         + parsers.check_optional_indent
8768         * -V("EndlineExceptions")
8769         * -parsers.starter)
8770       * parsers.spacechar^0
8771
8772     parsers.Endline = parsers.endline
8773       / writer.soft_line_break
8774
8775     parsers.EndlineNoSub = parsers.endline
8776
8777     parsers.NoSoftLineBreakEndline
8778       = parsers.newline
8779       * (parsers.check_minimal_indent
8780         * -V("NoSoftLineBreakEndlineExceptions")
8781         + parsers.check_optional_indent
8782         * -V("NoSoftLineBreakEndlineExceptions")
8783         * -parsers.starter)
8784       * parsers.spacechar^0
8785       / writer.space

```

```

8786
8787 parsers.EndlineBreak = parsers.backslash * parsers.Endline
8788                        / writer.hard_line_break
8789
8790 parsers.OptionalIndent
8791     = parsers.spacechar^1 / writer.space
8792
8793 parsers.Space          = parsers.spacechar^2 * parsers.Endline
8794                        / writer.hard_line_break
8795                        + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / self.
8796                        + parsers.spacechar^1 * parsers.Endline
8797                        / writer.soft_line_break
8798                        + parsers.spacechar^1 * -parsers.newline / self.expandtabs
8799
8800 parsers.NoSoftLineBreakSpace
8801     = parsers.spacechar^2 * parsers.Endline
8802                        / writer.hard_line_break
8803                        + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / self.
8804                        + parsers.spacechar^1 * parsers.Endline
8805                        / writer.soft_line_break
8806                        + parsers.spacechar^1 * -parsers.newline / self.expandtabs
8807
8808 parsers.NonbreakingEndline
8809     = parsers.endline
8810     / writer.soft_line_break
8811
8812 parsers.NonbreakingSpace
8813     = parsers.spacechar^2 * parsers.Endline
8814                        / writer.hard_line_break
8815     + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
8816     + parsers.spacechar^1 * parsers.Endline
8817     * parsers.optionalspace
8818     / writer.soft_line_break
8819     + parsers.spacechar^1 * parsers.optionalspace
8820     / writer.nbsp
8821

```

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.

```

8822 function self.auto_link_url(url, attributes)
8823   return writer.link(writer.escape(url),
8824                     url, nil, attributes)
8825 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.

```
8826 function self.auto_link_email(email, attributes)
8827   return writer.link(writer.escape(email),
8828                     "mailto:".email,
8829                     nil, attributes)
8830 end
8831
8832 parsers.AutoLinkUrl = parsers.auto_link_url
8833                   / self.auto_link_url
8834
8835 parsers.AutoLinkEmail
8836                   = parsers.auto_link_email
8837                   / self.auto_link_email
8838
8839 parsers.AutoLinkRelativeReference
8840                   = parsers.auto_link_relative_reference
8841                   / self.auto_link_url
8842
8843 parsers.LinkAndEmph = Ct(parsers.link_and_emph_table)
8844                   / self.defer_link_and_emphasis_processing
8845
8846 parsers.EscapedChar = parsers.backslash * C(parsers.escapable) / writer.string
8847
8848 parsers.InlineHtml = Cs(parsers.html_inline_comment) / writer.inline_html_comment
8849                   + Cs(parsers.html_any_empty_inline_tag
8850                       + parsers.html_inline_instruction
8851                       + parsers.html_inline_cdatasection
8852                       + parsers.html_inline_declaration
8853                       + parsers.html_any_open_inline_tag
8854                       + parsers.html_any_close_tag)
8855                   / writer.inline_html_tag
8856
8857 parsers.HtmlEntity = parsers.html_entities / writer.string
```

### 3.1.6.9 Block Elements (local)

```
8858 parsers.DisplayHtml = Cs(parsers.check_trail
8859                         * ( parsers.html_comment
8860                           + parsers.html_special_block
8861                           + parsers.html_block
8862                           + parsers.html_any_block
8863                           + parsers.html_instruction
8864                           + parsers.html_cdatasection
8865                           + parsers.html_declaration))
8866                   / writer.block_html_element
8867
```

```

8868 parsers.indented_non_blank_line = parsers.indentedline - parsers.blankline
8869
8870 parsers.Verbatim = Cs(
8871     parsers.check_code_trail
8872     * (parsers.line - parsers.blankline)
8873     * ((parsers.check_minimal_blank_indent_and_full_code_trail * pa
8874     * ((parsers.check_minimal_indent / "") * parsers.check_code_t
8875     * (parsers.line - parsers.blankline))^1)^0
8876     ) / self.expandtabs / writer.verbatim
8877
8878 parsers.Blockquote = parsers.blockquote_body
8879     / writer.blockquote
8880
8881 parsers.ThematicBreak = parsers.thematic_break_lines
8882     / writer.thematic_break
8883
8884 parsers.Reference = parsers.define_reference_parser
8885     / self.register_link
8886
8887 parsers.Paragraph = parsers.freeze_trail
8888     * (Ct((parsers.Inline)^1)
8889     * (parsers.newline + parsers.eof)
8890     * parsers.unfreeze_trail
8891     / writer.paragraph)
8892
8893 parsers.Plain = parsers.nonindentspace * Ct(parsers.Inline^1)
8894     / writer.plain

```

### 3.1.6.10 Lists (local)

```

8895
8896 if options.taskLists then
8897     parsers.tickbox = ( parsers.ticked_box
8898     + parsers.halfticked_box
8899     + parsers.unticked_box
8900     ) / writer.tickbox
8901 else
8902     parsers.tickbox = parsers.fail
8903 end
8904
8905 parsers.list_blank = parsers.conditionally_indented_blankline
8906
8907 parsers.ref_or_block_list_separated = parsers.sep_group_no_output(parsers.list_bla
8908     * parsers.minimally_indented_ref
8909     + parsers.block_sep_group(parsers.list_blank)
8910     * parsers.minimally_indented_block
8911

```

```

8912 parsers.ref_or_block_non_separated = parsers.minimally_indented_ref
8913 + (parsers.succeed / writer.interblocksep)
8914 * parsers.minimally_indented_block
8915 - parsers.minimally_indented_blankline
8916
8917 parsers.tight_list_loop_body_pair =
8918   parsers.create_loop_body_pair(parsers.ref_or_block_non_separated,
8919     parsers.minimally_indented_par_or_plain_no_blank,
8920     (parsers.succeed / writer.interblocksep),
8921     (parsers.succeed / writer.paragraphsep))
8922
8923 parsers.loose_list_loop_body_pair =
8924   parsers.create_loop_body_pair(parsers.ref_or_block_list_separated,
8925     parsers.minimally_indented_par_or_plain,
8926     parsers.block_sep_group(parsers.list_blank),
8927     parsers.par_sep_group(parsers.list_blank))
8928
8929 parsers.tight_list_content_loop = V("Block")
8930 * parsers.tight_list_loop_body_pair.block^0
8931 + (V("Paragraph") + V("Plain"))
8932 * parsers.ref_or_block_non_separated
8933 * parsers.tight_list_loop_body_pair.block^0
8934 + (V("Paragraph") + V("Plain"))
8935 * parsers.tight_list_loop_body_pair.par^0
8936
8937 parsers.loose_list_content_loop = V("Block")
8938 * parsers.loose_list_loop_body_pair.block^0
8939 + (V("Paragraph") + V("Plain"))
8940 * parsers.ref_or_block_list_separated
8941 * parsers.loose_list_loop_body_pair.block^0
8942 + (V("Paragraph") + V("Plain"))
8943 * parsers.loose_list_loop_body_pair.par^0
8944
8945 parsers.list_item_tightness_condition = -( parsers.list_blank^0
8946   * parsers.minimally_indented_ref_or_block
8947   * remove_indent("li")
8948   + remove_indent("li")
8949   * parsers.fail
8950
8951 parsers.indented_content_tight = Ct( (parsers.blankline / "")
8952   * #parsers.list_blank
8953   * remove_indent("li")
8954   + ( (V("Reference") + (parsers.blankline / ""))
8955     * parsers.check_minimal_indent
8956     * parsers.tight_list_content_loop
8957     + (V("Reference") + (parsers.blankline / ""))
8958     + (parsers.tickbox^-1 / writer.escape)

```



```

8959         * parsers.tight_list_content_loop
8960     )
8961     * parsers.list_item_tightness_condition
8962 )
8963
8964 parsers.indented_content_loose = Ct( (parsers.blankline / "")
8965     * #parsers.list_blank
8966     + ( (V("Reference")) + (parsers.blankline / ""))
8967     * parsers.check_minimal_indent
8968     * parsers.loose_list_content_loop
8969     + (V("Reference")) + (parsers.blankline / ""))
8970     + (parsers.checkbox^-1 / writer.escape)
8971     * parsers.loose_list_content_loop
8972 )
8973 )
8974
8975 parsers.TightListItem = function(starter)
8976     return -parsers.ThematicBreak
8977         * parsers.add_indent(starter, "li")
8978         * parsers.indented_content_tight
8979 end
8980
8981 parsers.LooseListItem = function(starter)
8982     return -parsers.ThematicBreak
8983         * parsers.add_indent(starter, "li")
8984         * parsers.indented_content_loose
8985         * remove_indent("li")
8986 end
8987
8988 parsers.BulletListOfType = function(bullet_type)
8989     local bullet = parsers.bullet(bullet_type)
8990     return ( Ct( parsers.TightListItem(bullet)
8991         * ( (parsers.check_minimal_indent / "")
8992         * parsers.TightListItem(bullet)
8993         )^0
8994     )
8995     * Cc(true)
8996     * -#( (parsers.list_blank^0 / "")
8997         * parsers.check_minimal_indent
8998         * (bullet - parsers.ThematicBreak)
8999     )
9000     + Ct( parsers.LooseListItem(bullet)
9001         * ( (parsers.list_blank^0 / "")
9002         * (parsers.check_minimal_indent / "")
9003         * parsers.LooseListItem(bullet)
9004         )^0
9005     )

```

```

9006             * Cc(false)
9007         ) / writer.bulletlist
9008     end
9009
9010     parsers.BulletList = parsers.BulletListOfType(parsers.dash)
9011                     + parsers.BulletListOfType(parsers.asterisk)
9012                     + parsers.BulletListOfType(parsers.plus)
9013
9014     local function ordered_list(items,tight,starter)
9015         local startnum = starter[2][1]
9016         if options.startNumber then
9017             startnum = tonumber(startnum) or 1 -- fallback for '#'
9018             if startnum ~= nil then
9019                 startnum = math.floor(startnum)
9020             end
9021         else
9022             startnum = nil
9023         end
9024         return writer.orderedlist(items,tight,startnum)
9025     end
9026
9027     parsers.OrderedListOfType = function(delimiter_type)
9028         local enumerator = parsers.enumerator(delimiter_type)
9029         return Cg(enumerator, "listtype")
9030             * (Ct( parsers.TightListItem(Cb("listtype"))
9031                 * ((parsers.check_minimal_indent / "") * parsers.TightListItem(enumerat
9032                 * Cc(true)
9033                 * -#((parsers.list_blank^0 / ""))
9034                 * parsers.check_minimal_indent * enumerator)
9035         + Ct( parsers.LooseListItem(Cb("listtype"))
9036             * ((parsers.list_blank^0 / ""))
9037             * (parsers.check_minimal_indent / "") * parsers.LooseListItem(enumerat
9038             * Cc(false)
9039         ) * Ct(Cb("listtype")) / ordered_list
9040     end
9041
9042     parsers.OrderedList = parsers.OrderedListOfType(parsers.period)
9043                     + parsers.OrderedListOfType(parsers.rparent)

```

### 3.1.6.11 Blank (local)

```

9044     parsers.Blank           = parsers.blankline / ""
9045                             + V("Reference")

```

### 3.1.6.12 Headings (local)

```

9046     function parsers.parse_heading_text(s)
9047         local inlines = self.parser_functions.parse_inlines(s)

```

```

9048     local flatten_inlines = self.writer.flatten_inlines
9049     self.writer.flatten_inlines = true
9050     local flat_text = self.parser_functions.parse_inlines(s)
9051     flat_text = util.ropo_to_string(flat_text)
9052     self.writer.flatten_inlines = flatten_inlines
9053     return {flat_text, inlines}
9054 end
9055
9056 -- parse atx header
9057 parsers.AtxHeading = parsers.check_trail_no_rem
9058     * Cg(parsers.heading_start, "level")
9059     * ((C( parsers.optionalspace
9060         * parsers.hash^0
9061         * parsers.optionalspace
9062         * parsers.newline)
9063       + parsers.spacechar^1
9064       * C(parsers.line))
9065     / strip_atx_end
9066     / parsers.parse_heading_text)
9067     * Cb("level")
9068     / writer.heading
9069
9070 parsers.heading_line = parsers.linechar^1
9071     - parsers.thematic_break_lines
9072
9073 parsers.heading_text = parsers.heading_line
9074     * ((V("Endline") / "\n") * (parsers.heading_line - parsers.heading_line)
9075     * parsers.newline^-1)
9076
9077 parsers.SetextHeading = parsers.freeze_trail * parsers.check_trail_no_rem
9078     * #(parsers.heading_text
9079     * parsers.check_minimal_indent * parsers.check_trail * parsers.heading_line)
9080     * Cs(parsers.heading_text)
9081     / parsers.parse_heading_text
9082     * parsers.check_minimal_indent_and_trail * parsers.heading_line
9083     * parsers.newline
9084     * parsers.unfreeze_trail
9085     / writer.heading
9086
9087 parsers.Heading = parsers.AtxHeading + parsers.SetextHeading

```

### 3.1.6.13 Syntax Specification

Define `reader->finalize_grammar` as a function that constructs the PEG grammar of markdown, applies syntax extensions `extensions` and returns a conversion function that takes a markdown string and turns it into a plain  $\text{T}_{\text{E}}\text{X}$  output.

```

9088     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.

```

9089     local walkable_syntax = (function(global_walkable_syntax)
9090       local local_walkable_syntax = {}
9091       for lhs, rule in pairs(global_walkable_syntax) do
9092         local_walkable_syntax[lhs] = util.table_copy(rule)
9093       end
9094       return local_walkable_syntax
9095     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.

```

9096     local current_extension_name = nil
9097     self.insert_pattern = function(selector, pattern, pattern_name)
9098       assert(pattern_name == nil or type(pattern_name) == "string")
9099       local _, _, lhs, pos, rhs = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
9100       assert(lhs ~= nil,
9101         [[Expected selector in form "LHS (before|after|instead of) RHS", not "]]
9102         .. selector .. ["])
9103       assert(walkable_syntax[lhs] ~= nil,
9104         [[Rule ]] .. lhs .. [[ -> ... does not exist in markdown grammar]])
9105       assert(pos == "before" or pos == "after" or pos == "instead of",
9106         [[Expected positional specifier "before", "after", or "instead of", not "]]
9107         .. pos .. ["])
9108       local rule = walkable_syntax[lhs]
9109       local index = nil
9110       for current_index, current_rhs in ipairs(rule) do
9111         if type(current_rhs) == "string" and current_rhs == rhs then
9112           index = current_index
9113           if pos == "after" then
9114             index = index + 1
9115           end
9116           break
9117         end
9118       end
9119       assert(index ~= nil,
9120         [[Rule ]] .. lhs .. [[ -> ]] .. rhs
9121         .. [[ does not exist in markdown grammar]])
9122       local accountable_pattern
9123       if current_extension_name then
9124         accountable_pattern = { pattern, current_extension_name, pattern_name }
9125       else
9126         assert(type(pattern) == "string",

```

```

9127         [[reader->insert_pattern() was called outside an extension with ]]
9128         .. [[a PEG pattern instead of a rule name]])
9129         accountable_pattern = pattern
9130     end
9131     if pos == "instead of" then
9132         rule[index] = accountable_pattern
9133     else
9134         table.insert(rule, index, accountable_pattern)
9135     end
9136 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.

```

9137     local syntax =
9138         { "Blocks",
9139
9140           Blocks = V("InitializeState")
9141                 * ( V("ExpectedJekyllData")
9142                   * (V("Blank")^0 / writer.interblocksep)
9143                   )^-1
9144                 * 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.

```

9145                                     * ( V("Block")
9146                                       * ( V("Blank")^0 * parsers.eof
9147                                         + ( V("Blank")^2 / writer.paragraphsep
9148                                           + V("Blank")^0 / writer.interblocksep
9149                                         )
9150                                       )
9151                                       + ( V("Paragraph") + V("Plain") )
9152                                       * ( V("Blank")^0 * parsers.eof
9153                                         + ( V("Blank")^2 / writer.paragraphsep
9154                                           + V("Blank")^0 / writer.interblocksep
9155                                         )
9156                                       )
9157                                       * V("Block")
9158                                       * ( V("Blank")^0 * parsers.eof
9159                                         + ( V("Blank")^2 / writer.paragraphsep
9160                                           + V("Blank")^0 / writer.interblocksep
9161                                         )
9162                                       )
9163                                       + ( V("Paragraph") + V("Plain") )
9164                                       * ( V("Blank")^0 * parsers.eof
9165                                         + V("Blank")^0 / writer.paragraphsep
9166                                       )
9167                                     )^0,

```

```

9168
9169     ExpectedJekyllData      = parsers.fail,
9170
9171     Blank                    = parsers.Blank,
9172     Reference                 = parsers.Reference,
9173
9174     Blockquote                = parsers.Blockquote,
9175     Verbatim                  = parsers.Verbatim,
9176     ThematicBreak             = parsers.ThematicBreak,
9177     BulletList                 = parsers.BulletList,
9178     OrderedList               = parsers.OrderedList,
9179     DisplayHtml               = parsers.DisplayHtml,
9180     Heading                   = parsers.Heading,
9181     Paragraph                 = parsers.Paragraph,
9182     Plain                     = parsers.Plain,
9183
9184     EndlineExceptions         = parsers.EndlineExceptions,
9185     NoSoftLineBreakEndlineExceptions
9186                             = parsers.NoSoftLineBreakEndlineExceptions,
9187
9188     Str                        = parsers.Str,
9189     Space                     = parsers.Space,
9190     NoSoftLineBreakSpace     = parsers.NoSoftLineBreakSpace,
9191     OptionalIndent           = parsers.OptionalIndent,
9192     Endline                   = parsers.Endline,
9193     EndlineNoSub              = parsers.EndlineNoSub,
9194     NoSoftLineBreakEndline
9195                             = parsers.NoSoftLineBreakEndline,
9196     EndlineBreak              = parsers.EndlineBreak,
9197     LinkAndEmph               = parsers.LinkAndEmph,
9198     Code                      = parsers.Code,
9199     AutoLinkUrl               = parsers.AutoLinkUrl,
9200     AutoLinkEmail             = parsers.AutoLinkEmail,
9201     AutoLinkRelativeReference
9202                             = parsers.AutoLinkRelativeReference,
9203     InlineHtml                = parsers.InlineHtml,
9204     HtmlEntity                = parsers.HtmlEntity,
9205     EscapedChar               = parsers.EscapedChar,
9206     Smart                     = parsers.Smart,
9207     Symbol                    = parsers.Symbol,
9208     SpecialChar               = parsers.fail,
9209     InitializeState           = parsers.succeed,
9210 }

```

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].

```
9211 self.update_rule = function(rule_name, get_pattern)
9212     assert(current_extension_name ~= nil)
9213     assert(syntax[rule_name] ~= nil,
9214         [[Rule ]] .. rule_name .. [[ -> ... does not exist in markdown grammar]])
9215     local previous_pattern
9216     local extension_name
9217     if walkable_syntax[rule_name] then
9218         local previous_accountable_pattern = walkable_syntax[rule_name][1]
9219         previous_pattern = previous_accountable_pattern[1]
9220         extension_name = previous_accountable_pattern[2] .. ", " .. current_extension_name
9221     else
9222         previous_pattern = nil
9223         extension_name = current_extension_name
9224     end
9225     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
```

```
9226     if type(get_pattern) == "function" then
9227         pattern = get_pattern(previous_pattern)
9228     else
9229         assert(previous_pattern == nil,
9230             [[Rule ]] .. rule_name ..
9231             [[ has already been updated by ]] .. extension_name)
9232         pattern = get_pattern
9233     end
9234     local accountable_pattern = { pattern, extension_name, rule_name }
9235     walkable_syntax[rule_name] = { accountable_pattern }
9236 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.

```
9237     local special_characters = {}
9238     self.add_special_character = function(c)
9239         table.insert(special_characters, c)
```

```

9240     syntax.SpecialChar = S(table.concat(special_characters, ""))
9241     end
9242
9243     self.add_special_character("*")
9244     self.add_special_character("[")
9245     self.add_special_character("]")
9246     self.add_special_character("<")
9247     self.add_special_character("!")
9248     self.add_special_character("\\")

```

Add method `reader->initialize_named_group` that defines named groups with a default capture value.

```

9249     self.initialize_named_group = function(name, value)
9250         local pattern = Ct("")
9251         if value ~= nil then
9252             pattern = pattern / value
9253         end
9254         syntax.InitializeState = syntax.InitializeState
9255             * Cg(pattern, name)
9256     end

```

Add a named group for indentation.

```

9257     self.initialize_named_group("indent_info")

```

Apply syntax extensions.

```

9258     for _, extension in ipairs(extensions) do
9259         current_extension_name = extension.name
9260         extension.extend_writer(writer)
9261         extension.extend_reader(self)
9262     end
9263     current_extension_name = nil

```

If the `debugExtensions` option is enabled, serialize `walkable_syntax` to a JSON for debugging purposes.

```

9264     if options.debugExtensions then
9265         local sorted_lhs = {}
9266         for lhs, _ in pairs(walkable_syntax) do
9267             table.insert(sorted_lhs, lhs)
9268         end
9269         table.sort(sorted_lhs)
9270
9271         local output_lines = {"{"}
9272         for lhs_index, lhs in ipairs(sorted_lhs) do
9273             local encoded_lhs = util.encode_json_string(lhs)
9274             table.insert(output_lines, [[    ]] .. encoded_lhs .. [[: ]])
9275             local rule = walkable_syntax[lhs]
9276             for rhs_index, rhs in ipairs(rule) do
9277                 local human_readable_rhs

```



```

9278         if type(rhs) == "string" then
9279             human_readable_rhs = rhs
9280         else
9281             local pattern_name
9282             if rhs[3] then
9283                 pattern_name = rhs[3]
9284             else
9285                 pattern_name = "Anonymous Pattern"
9286             end
9287             local extension_name = rhs[2]
9288             human_readable_rhs = pattern_name .. [[ (] .. extension_name .. [[]]
9289         end
9290         local encoded_rhs = util.encode_json_string(human_readable_rhs)
9291         local output_line = [[          ]] .. encoded_rhs
9292         if rhs_index < #rule then
9293             output_line = output_line .. ", "
9294         end
9295         table.insert(output_lines, output_line)
9296     end
9297     local output_line = "    ]"
9298     if lhs_index < #sorted_lhs then
9299         output_line = output_line .. ", "
9300     end
9301     table.insert(output_lines, output_line)
9302 end
9303 table.insert(output_lines, "}")
9304
9305 local output = table.concat(output_lines, "\n")
9306 local output_filename = options.debugExtensionsFileName
9307 local output_file = assert(io.open(output_filename, "w"),
9308     [[Could not open file ]] .. output_filename .. [[ for writing]])
9309 assert(output_file:write(output))
9310 assert(output_file:close())
9311 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](#).

```

9312     for lhs, rule in pairs(walkable_syntax) do
9313         syntax[lhs] = parsers.fail
9314         for _, rhs in ipairs(rule) do
9315             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.

```

9316         if type(rhs) == "string" then
9317             pattern = V(rhs)
9318         else
9319             pattern = rhs[1]
9320             if type(pattern) == "string" then
9321                 pattern = V(pattern)
9322             end
9323         end
9324         syntax[lhs] = syntax[lhs] + pattern
9325     end
9326 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.

```

9327     if options.underscores then
9328         self.add_special_character("_")
9329     end
9330
9331     if not options.codeSpans then
9332         syntax.Code = parsers.fail
9333     else
9334         self.add_special_character("`")
9335     end
9336
9337     if not options.html then
9338         syntax.DisplayHtml = parsers.fail
9339         syntax.InlineHtml = parsers.fail
9340         syntax.HtmlEntity = parsers.fail
9341     else
9342         self.add_special_character("&")
9343     end
9344
9345     if options.preserveTabs then
9346         options.stripIndent = false
9347     end
9348
9349     if not options.smartEllipses then
9350         syntax.Smart = parsers.fail
9351     else
9352         self.add_special_character(".")
9353     end
9354
9355     if not options.relativeReferences then
9356         syntax.AutoLinkRelativeReference = parsers.fail
9357     end
9358

```

```

9359     if options.contentLevel == "inline" then
9360         syntax[1] = "Inlines"
9361         syntax.Inlines = V("InitializeState")
9362             * parsers.Inline^0
9363             * ( parsers.spacing^0
9364                 * parsers.eof / "" )
9365         syntax.Space = parsers.Space + parsers.blankline / writer.space
9366     end
9367
9368     local blocks_nested_t = util.table_copy(syntax)
9369     blocks_nested_t.ExpectedJekyllData = parsers.fail
9370     parsers.blocks_nested = Ct(blocks_nested_t)
9371
9372     parsers.blocks = Ct(syntax)
9373
9374     local inlines_t = util.table_copy(syntax)
9375     inlines_t[1] = "Inlines"
9376     inlines_t.Inlines = V("InitializeState")
9377         * parsers.Inline^0
9378         * ( parsers.spacing^0
9379             * parsers.eof / "" )
9380     parsers.inlines = Ct(inlines_t)
9381
9382     local inlines_no_inline_note_t = util.table_copy(inlines_t)
9383     inlines_no_inline_note_t.InlineNote = parsers.fail
9384     parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
9385
9386     local inlines_no_html_t = util.table_copy(inlines_t)
9387     inlines_no_html_t.DisplayHtml = parsers.fail
9388     inlines_no_html_t.InlineHtml = parsers.fail
9389     inlines_no_html_t.HtmlEntity = parsers.fail
9390     parsers.inlines_no_html = Ct(inlines_no_html_t)
9391
9392     local inlines_nbsp_t = util.table_copy(inlines_t)
9393     inlines_nbsp_t.Endline = parsers.NonbreakingEndline
9394     inlines_nbsp_t.Space = parsers.NonbreakingSpace
9395     parsers.inlines_nbsp = Ct(inlines_nbsp_t)
9396
9397     local inlines_no_link_or_emphasis_t = util.table_copy(inlines_t)
9398     inlines_no_link_or_emphasis_t.LinkAndEmph = parsers.fail
9399     inlines_no_link_or_emphasis_t.EndlineExceptions = parsers.EndlineExceptions - par
9400     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..

```
9401     return function(input)
```

Unicode-normalize the input.

```

9402     if options.unicodeNormalization then
9403         local form = options.unicodeNormalizationForm
9404         if form == "nfc" then
9405             input = uni_algos.normalize.NFC(input)
9406         elseif form == "nfd" then
9407             input = uni_algos.normalize.NFD(input)
9408         elseif form == "nfkc" then
9409             input = uni_algos.normalize.NFKC(input)
9410         elseif form == "nfkd" then
9411             input = uni_algos.normalize.NFKD(input)
9412         else
9413             error(format("Unknown normalization form %s", form))
9414         end
9415     end

```

Since the Lua converter expects UNIX line endings, normalize the input. Also add a line ending at the end of the file in case the input file has none.

```

9416     input = input:gsub("\r\n?", "\n")
9417     if input:sub(-1) ~= "\n" then
9418         input = input .. "\n"
9419     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.

```

9420     references = {}
9421     local opt_string = {}
9422     for k, _ in pairs(defaultOptions) do
9423         local v = options[k]
9424         if type(v) == "table" then
9425             for _, i in ipairs(v) do
9426                 opt_string[#opt_string+1] = k .. "=" .. tostring(i)
9427             end
9428         elseif k ~= "cacheDir" then
9429             opt_string[#opt_string+1] = k .. "=" .. tostring(v)
9430         end
9431     end
9432     table.sort(opt_string)
9433     local salt = table.concat(opt_string, ",") .. "," .. metadata.version
9434     local output
9435     local function convert(input)
9436         local document = self.parser_functions.parse_blocks(input)
9437         local output = util.rope_to_string(writer.document(document))

```

Remove block element / paragraph separators immediately followed by the output of `writer->undosep`, possibly interleaved by section ends. Then, remove any leftover output of `writer->undosep`.

```

9438     local undosep_start, undosep_end

```

```

9439     local potential_secend_start, secend_start
9440     local potential_sep_start, sep_start
9441     while true do
9442         -- find a `writer->undosep`
9443         undosep_start, undosep_end = output:find(writer.undosep_text, 1, true)
9444         if undosep_start == nil then break end
9445         -- skip any preceding section ends
9446         secend_start = undosep_start
9447         while true do
9448             potential_secend_start = secend_start - #writer.secend_text
9449             if potential_secend_start < 1
9450                 or output:sub(potential_secend_start, secend_start - 1) ~= writer.sece
9451                 break
9452             end
9453             secend_start = potential_secend_start
9454         end
9455         -- find an immediately preceding block element / paragraph separator
9456         sep_start = secend_start
9457         potential_sep_start = sep_start - #writer.interblocksep_text
9458         if potential_sep_start >= 1
9459             and output:sub(potential_sep_start, sep_start - 1) == writer.interblocks
9460             sep_start = potential_sep_start
9461         else
9462             potential_sep_start = sep_start - #writer.paragraphsep_text
9463             if potential_sep_start >= 1
9464                 and output:sub(potential_sep_start, sep_start - 1) == writer.paragraph
9465             sep_start = potential_sep_start
9466         end
9467     end
9468     -- remove `writer->undosep` and immediately preceding block element / parag
9469     output = output:sub(1, sep_start - 1)
9470         .. output:sub(secend_start, undosep_start - 1)
9471         .. output:sub(undosep_end + 1)
9472     end
9473     return output
9474 end

```

If we cache markdown documents, produce the cache file and transform its filename to plain  $\text{T}_{\text{E}}\text{X}$  output via the `writer->pack` method.

```

9475     if options.eagerCache or options.finalizeCache then
9476         local name = util.cache(options.cacheDir, input, salt, convert,
9477             ".md" .. writer.suffix)
9478         output = writer.pack(name)

```

Otherwise, return the result of the conversion directly.

```

9479     else
9480         output = convert(input)
9481     end

```

If the `finalizeCache` option is enabled, populate the frozen cache in the file `frozenCacheFileName` with an entry for markdown document number `frozenCacheCounter`.

```
9482     if options.finalizeCache then
9483         local file, mode
9484         if options.frozenCacheCounter > 0 then
9485             mode = "a"
9486         else
9487             mode = "w"
9488         end
9489         file = assert(io.open(options.frozenCacheFileName, mode),
9490             [[Could not open file ]] .. options.frozenCacheFileName
9491             .. [[ for writing]])
9492         assert(file:write([[\\expandafter\\global\\expandafter\\def\\csname ]]
9493             .. [[markdownFrozenCache]] .. options.frozenCacheCounter
9494             .. [[\\endcsname{}}] .. output .. [{}]] .. "\\n"))
9495         assert(file:close())
9496     end
9497     return output
9498 end
9499 end
9500 return self
9501 end
```

### 3.1.7 Built-In Syntax Extensions

Create `extensions` hash table that contains built-in syntax extensions. Syntax extensions are functions that produce objects with two methods: `extend_writer` and `extend_reader`. The `extend_writer` object takes a `writer` object as the only parameter and mutates it. Similarly, `extend_reader` takes a `reader` object as the only parameter and mutates it.

```
9502 M.extensions = {}
```

#### 3.1.7.1 Bracketed Spans

The `extensions.bracketed_spans` function implements the Pandoc bracketed span syntax extension.

```
9503 M.extensions.bracketed_spans = function()
9504     return {
9505         name = "built-in bracketed_spans syntax extension",
9506         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.

```
9507         function self.span(s, attr)
9508             if self.flatten_inlines then return s end
```

```

9509         return {"\\markdownRendererBracketedSpanAttributeContextBegin",
9510                 self.attributes(attr),
9511                 s,
9512                 "\\markdownRendererBracketedSpanAttributeContextEnd{}}"}
9513     end
9514 end, extend_reader = function(self)
9515     local parsers = self.parsers
9516     local writer = self.writer
9517
9518     local span_label = parsers.lbracket
9519                     * (Cs((parsers.alphanumeric^1
9520                         + parsers.inticks
9521                         + parsers.autolink
9522                         + V("InlineHtml")
9523                         + ( parsers.backslash * parsers.backslash)
9524                         + ( parsers.backslash * (parsers.lbracket + parsers.rbracket)
9525                         + V("Space") + V("Endline")
9526                         + (parsers.any
9527                           - (parsers.newline + parsers.lbracket + parsers.rbracket
9528                             + parsers.blankline^2))))^1)
9529                     / self.parser_functions.parse_inlines)
9530     * parsers.rbracket
9531
9532     local Span = span_label
9533                 * Ct(parsers.attributes)
9534                 / writer.span
9535
9536     self.insert_pattern("Inline before LinkAndEmph",
9537                       Span, "Span")
9538 end
9539 }
9540 end

```

### 3.1.7.2 Citations

The `extensions.citations` function implements the Pandoc citation syntax extension. When the `citation_nbsps` parameter is enabled, the syntax extension will replace regular spaces with non-breaking spaces inside the prenotes and postnotes of citations.

```

9541 M.extensions.citations = function(citation_nbsps)
9542     return {
9543         name = "built-in citations syntax extension",
9544         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.

```
9545     function self.citations(text_cites, cites)
9546         local buffer = {}
9547         if self.flatten_inlines then
9548             for _,cite in ipairs(cites) do
9549                 if cite.prenote then
9550                     table.insert(buffer, {cite.prenote, " "})
9551                 end
9552                 table.insert(buffer, cite.name)
9553                 if cite.postnote then
9554                     table.insert(buffer, {" ", cite.postnote})
9555                 end
9556             end
9557         else
9558             table.insert(buffer, {"\\markdownRenderer", text_cites and "TextCite" or "C",
9559                 "{", #cites, "}")
9560             for _,cite in ipairs(cites) do
9561                 table.insert(buffer, {cite.suppress_author and "-" or "+", "{",
9562                     cite.prenote or "", "}{", cite.postnote or "", "}{", cite.name, "}")
9563             end
9564         end
9565         return buffer
9566     end
9567 end, extend_reader = function(self)
9568     local parsers = self.parsers
9569     local writer = self.writer
9570
9571     local citation_chars
9572         = parsers.alphanumeric
9573         + S("#$%&-+<>-/_")
9574
9575     local citation_name
9576         = Cs(parsers.dash^-1) * parsers.at
9577         * Cs(citation_chars
9578             * (((citation_chars + parsers.internal_punctuation
```



```

9579         - parsers.comma - parsers.semicolon)
9580     * -#((parsers.internal_punctuation - parsers.comma
9581         - parsers.semicolon)^0
9582         * -(citation_chars + parsers.internal_punctuation
9583           - parsers.comma - parsers.semicolon)))^0
9584     * citation_chars)^-1)
9585
9586     local citation_body_prenote
9587         = Cs((parsers.alphanumeric^1
9588             + parsers.bracketed
9589             + parsers.inticks
9590             + parsers.autolink
9591             + V("InlineHtml")
9592             + V("Space") + V("Endline")
9593             + (parsers.anyescaped
9594               - (parsers.newline + parsers.rbracket + parsers.blankline^
9595                 - (parsers.spnl * parsers.dash^-1 * parsers.at))^1)
9596
9597     local citation_body_postnote
9598         = Cs((parsers.alphanumeric^1
9599             + parsers.bracketed
9600             + parsers.inticks
9601             + parsers.autolink
9602             + V("InlineHtml")
9603             + V("Space") + V("Endline")
9604             + (parsers.anyescaped
9605               - (parsers.newline + parsers.rbracket + parsers.semicolon
9606                 + parsers.blankline^2))
9607             - (parsers.spnl * parsers.rbracket))^1)
9608
9609     local citation_body_chunk
9610         = ( citation_body_prenote
9611           * parsers.spnlc_sep
9612           + Cc("")
9613           * parsers.spnlc
9614         )
9615     * citation_name
9616     * (parsers.internal_punctuation - parsers.semicolon)^-
1
9617     * ( parsers.spnlc
9618       * citation_body_postnote
9619       + Cc("")
9620       * parsers.spnlc
9621     )
9622
9623     local citation_body
9624         = citation_body_chunk

```

```

9625         * ( parsers.semicolon
9626         * parsers.spnlc
9627         * citation_body_chunk
9628         )^0
9629
9630     local citation_headless_body_postnote
9631         = Cs((parsers.alphanumeric^1
9632             + parsers.bracketed
9633             + parsers.inticks
9634             + parsers.autolink
9635             + V("InlineHtml")
9636             + V("Space") + V("Endline")
9637             + (parsers.anyescaped
9638               - (parsers.newline + parsers.rbracket + parsers.at
9639                 + parsers.semicolon + parsers.blankline^2))
9640             - (parsers.spnl * parsers.rbracket))^0)
9641
9642     local citation_headless_body
9643         = citation_headless_body_postnote
9644         * ( parsers.semicolon
9645         * parsers.spnlc
9646         * citation_body_chunk
9647         )^0
9648
9649     local citations
9650         = function(text_cites, raw_cites)
9651         local function normalize(str)
9652             if str == "" then
9653                 str = nil
9654             else
9655                 str = (citation_nbsps and
9656                     self.parser_functions.parse_inlines_nbsp or
9657                     self.parser_functions.parse_inlines)(str)
9658             end
9659             return str
9660         end
9661
9662         local cites = {}
9663         for i = 1,#raw_cites,4 do
9664             cites[#cites+1] = {
9665                 prenote = normalize(raw_cites[i]),
9666                 suppress_author = raw_cites[i+1] == "-",
9667                 name = writer.identifier(raw_cites[i+2]),
9668                 postnote = normalize(raw_cites[i+3]),
9669             }
9670         end
9671         return writer.citations(text_cites, cites)

```

```

9672     end
9673
9674     local TextCitations
9675         = Ct((parsers.spnlc
9676             * Cc("")
9677             * citation_name
9678             * ((parsers.spnlc
9679                 * parsers.lbracket
9680                 * citation_headless_body
9681                 * parsers.rbracket) + Cc("")))~1)
9682     / function(raw_cites)
9683         return citations(true, raw_cites)
9684     end
9685
9686     local ParenthesizedCitations
9687         = Ct((parsers.spnlc
9688             * parsers.lbracket
9689             * citation_body
9690             * parsers.rbracket)^1)
9691     / function(raw_cites)
9692         return citations(false, raw_cites)
9693     end
9694
9695     local Citations = TextCitations + ParenthesizedCitations
9696
9697     self.insert_pattern("Inline before LinkAndEmph",
9698                       Citations, "Citations")
9699
9700     self.add_special_character("@")
9701     self.add_special_character("-")
9702 end
9703 }
9704 end

```

### 3.1.7.3 Content Blocks

The `extensions.content_blocks` function implements the iA Writer content blocks syntax extension. The `language_map` parameter specifies the filename of the JSON file that maps filename extensions to programming language names.

```
9705 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.

```
9706     local languages_json = (function()
9707         local base, prev, curr
```

```

9708     for _, pathname in ipairs{kpse.lookup(language_map, { all=true })} do
9709         local file = io.open(pathname, "r")
9710         if not file then goto continue end
9711         local input = assert(file:read("*a"))
9712         assert(file:close())
9713         local json = input:gsub('[^\n]-':'', '%1=')
9714         curr = load("_ENV = {}; return "..json")()
9715         if type(curr) == "table" then
9716             if base == nil then
9717                 base = curr
9718             else
9719                 setmetatable(prev, { __index = curr })
9720             end
9721             prev = curr
9722         end
9723         ::continue::
9724     end
9725     return base or {}
9726 end)()
9727
9728 return {
9729     name = "built-in content_blocks syntax extension",
9730     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.

```

9731     function self.contentblock(src,suf,type,tit)
9732         if not self.is_writing then return "" end
9733         src = src..".."..suf
9734         suf = suf:lower()
9735         if type == "onlineimage" then
9736             return {"\\markdownRendererContentBlockOnlineImage{"..suf.."},"",
9737                 {"",self.string(src),"},",
9738                 {"",self.uri(src),"},",
9739                 {"",self.string(tit or ""),"},"}
9740         elseif languages_json[suf] then
9741             return {"\\markdownRendererContentBlockCode{"..suf.."},"",
9742                 {"",self.string(languages_json[suf]),"},",
9743                 {"",self.string(src),"},",
9744                 {"",self.uri(src),"},",
9745                 {"",self.string(tit or ""),"},"}
9746         else
9747             return {"\\markdownRendererContentBlock{"..suf.."},"",
9748                 {"",self.string(src),"},",
9749                 {"",self.uri(src),"},",

```

```

9750             {"",self.string(tit or ""),"}"}
9751         end
9752     end
9753 end, extend_reader = function(self)
9754     local parsers = self.parsers
9755     local writer = self.writer
9756
9757     local contentblock_tail
9758         = parsers.optionaltitle
9759         * (parsers.newline + parsers.eof)
9760
9761     -- case insensitive online image suffix:
9762     local onlineimagesuffix
9763         = (function(...)
9764             local parser = nil
9765             for _, suffix in ipairs({...}) do
9766                 local pattern=nil
9767                 for i=1,#suffix do
9768                     local char=suffix:sub(i,i)
9769                     char = S(char:lower()..char:upper())
9770                     if pattern == nil then
9771                         pattern = char
9772                     else
9773                         pattern = pattern * char
9774                     end
9775                 end
9776                 if parser == nil then
9777                     parser = pattern
9778                 else
9779                     parser = parser + pattern
9780                 end
9781             end
9782             return parser
9783         end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
9784
9785     -- online image url for iA Writer content blocks with mandatory suffix,
9786     -- allowing nested brackets:
9787     local onlineimageurl
9788         = (parsers.less
9789         * Cs((parsers.anyescaped
9790             - parsers.more
9791             - parsers.spacing
9792             - #(parsers.period
9793                 * onlineimagesuffix
9794                 * parsers.more
9795                 * contentblock_tail))^0)
9796         * parsers.period

```

```

9797         * Cs(onlineimagesuffix)
9798         * parsers.more
9799         + (Cs((parsers.inparens
9800             + (parsers.anyescaped
9801                 - parsers.spacing
9802                 - parsers.rparent
9803                 - #(parsers.period
9804                     * onlineimagesuffix
9805                     * contentblock_tail)))^0)
9806         * parsers.period
9807         * Cs(onlineimagesuffix))
9808     ) * Cc("onlineimage")
9809
9810     -- filename for iA Writer content blocks with mandatory suffix:
9811     local localfilepath
9812         = parsers.slash
9813         * Cs((parsers.anyescaped
9814             - parsers.tab
9815             - parsers.newline
9816             - #(parsers.period
9817                 * parsers.alphanumeric^1
9818                 * contentblock_tail))^1)
9819         * parsers.period
9820         * Cs(parsers.alphanumeric^1)
9821         * Cc("localfile")
9822
9823     local ContentBlock
9824         = parsers.check_trail_no_rem
9825         * (localfilepath + onlineimageurl)
9826         * contentblock_tail
9827         / writer.contentblock
9828
9829     self.insert_pattern("Block before Blockquote",
9830                       ContentBlock, "ContentBlock")
9831 end
9832 }
9833 end

```

### 3.1.7.4 Definition Lists

The `extensions.definition_lists` function implements the Pandoc definition list syntax extension. If the `tight_lists` parameter is `true`, tight lists will produce special right item renderers.

```

9834 M.extensions.definition_lists = function(tight_lists)
9835   return {
9836     name = "built-in definition_lists syntax extension",
9837     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.

```

9838     local function dlitem(term, defs)
9839         local retVal = {"\\markdownRendererDlItem{" ,term,"}"}
9840         for _, def in ipairs(defs) do
9841             retVal[#retVal+1] = {"\\markdownRendererDlDefinitionBegin ",def,
9842                 "\\markdownRendererDlDefinitionEnd "}
9843         end
9844         retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
9845         return retVal
9846     end
9847
9848     function self.definitionlist(items,tight)
9849         if not self.is_writing then return "" end
9850         local buffer = {}
9851         for _,item in ipairs(items) do
9852             buffer[#buffer + 1] = dlitem(item.term, item.definitions)
9853         end
9854         if tight and tight_lists then
9855             return {"\\markdownRendererDlBeginTight\\n", buffer,
9856                 "\\n\\markdownRendererDlEndTight"}
9857         else
9858             return {"\\markdownRendererDlBegin\\n", buffer,
9859                 "\\n\\markdownRendererDlEnd"}
9860         end
9861     end
9862     end, extend_reader = function(self)
9863         local parsers = self.parsers
9864         local writer = self.writer
9865
9866         local defstartchar = S("~:")
9867
9868         local defstart  = parsers.check_trail_length(0) * defstartchar * #parsers.spaci
9869                               * (parsers.tab + parsers.space^-
9870
9871                               3)
9872                               + parsers.check_trail_length(1) * defstartchar * #parsers.spaci
9873                               * (parsers.tab + parsers.space^-
9874
9875                               2)
9876                               + parsers.check_trail_length(2) * defstartchar * #parsers.spaci
9877                               * (parsers.tab + parsers.space^-
9878
9879                               1)
9880                               + parsers.check_trail_length(3) * defstartchar * #parsers.spaci
9881
9882         local indented_line = (parsers.check_minimal_indent / "") * parsers.check_code_

```

```

9877
9878     local blank = parsers.check_minimal_blank_indent_and_any_trail * parsers.option
9879
9880     local dlchunk = Cs(parsers.line * (indented_line - blank)^0)
9881
9882     local indented_blocks = function(bl)
9883         return Cs( bl
9884             * (blank^1 * (parsers.check_minimal_indent / ""))
9885             * parsers.check_code_trail * -parsers.blankline * bl)^0
9886             * (blank^1 + parsers.eof))
9887     end
9888
9889     local function definition_list_item(term, defs, _)
9890         return { term = self.parser_functions.parse_inlines(term),
9891             definitions = defs }
9892     end
9893
9894     local DefinitionListItemLoose
9895         = C(parsers.line) * blank^0
9896         * Ct((parsers.check_minimal_indent * (defstart
9897             * indented_blocks(dlchunk)
9898             / self.parser_functions.parse_blocks_nested))^1)
9899         * Cc(false) / definition_list_item
9900
9901     local DefinitionListItemTight
9902         = C(parsers.line)
9903         * Ct((parsers.check_minimal_indent * (defstart * dlchunk
9904             / self.parser_functions.parse_blocks_nested))^1)
9905         * Cc(true) / definition_list_item
9906
9907     local DefinitionList
9908         = ( Ct(DefinitionListItemLoose^1) * Cc(false)
9909             + Ct(DefinitionListItemTight^1)
9910             * (blank^0
9911                 * -DefinitionListItemLoose * Cc(true))
9912             ) / writer.definitionlist
9913
9914     self.insert_pattern("Block after Heading",
9915         DefinitionList, "DefinitionList")
9916 end
9917 }
9918 end

```

### 3.1.7.5 Fancy Lists

The `extensions.fancy_lists` function implements the Pandoc fancy list syntax extension.



```

9919 M.extensions.fancy_lists = function()
9920   return {
9921     name = "built-in fancy_lists syntax extension",
9922     extend_writer = function(self)
9923       local options = self.options
9924

```

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.

```

9925     function self.fancylist(items,tight,startnum,numstyle,numdelim)
9926       if not self.is_writing then return "" end
9927       local buffer = {}
9928       local num = startnum
9929       for _,item in ipairs(items) do
9930         if item ~= "" then
9931           buffer[#buffer + 1] = self.fancyitem(item,num)
9932         end
9933         if num ~= nil and item ~= "" then
9934           num = num + 1
9935         end
9936       end
9937       local contents = util.intersperse(buffer,"\n")
9938       if tight and options.tightLists then
9939         return {"\markdownRendererFancyOlBeginTight{",
9940               numstyle,"}{",numdelim,"}",contents,
9941               "\n\markdownRendererFancyOlEndTight "}
9942       else

```

```

9943         return {"\markdownRendererFancyOlBegin{",
9944             numstyle,"}{",numdelim,"}",contents,
9945             "\n\markdownRendererFancyOlEnd "}
9946     end
9947 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.

```

9948     function self.fancyitem(s,num)
9949         if num ~= nil then
9950             return {"\markdownRendererFancyOlItemWithNumber{",num,"}",s,
9951                 "\n\markdownRendererFancyOlItemEnd "}
9952         else
9953             return {"\markdownRendererFancyOlItem ",s,"\n\markdownRendererFancyOlItemEnd "}
9954         end
9955     end
9956 end, extend_reader = function(self)
9957     local parsers = self.parsers
9958     local options = self.options
9959     local writer = self.writer
9960
9961     local function combine_markers_and_delims(markers, delims)
9962         local markers_table = {}
9963         for _,marker in ipairs(markers) do
9964             local start_marker
9965             local continuation_marker
9966             if type(marker) == "table" then
9967                 start_marker = marker[1]
9968                 continuation_marker = marker[2]
9969             else
9970                 start_marker = marker
9971                 continuation_marker = marker
9972             end
9973             for _,delim in ipairs(delims) do
9974                 table.insert(markers_table, {start_marker, continuation_marker, delim})
9975             end
9976         end
9977         return markers_table
9978     end
9979
9980     local function join_table_with_func(func, markers_table)
9981         local pattern = func(table.unpack(markers_table[1]))
9982         for i = 2, #markers_table do
9983             pattern = pattern + func(table.unpack(markers_table[i]))
9984         end
9985         return pattern

```

```

9986     end
9987
9988     local lowercase_letter_marker = R("az")
9989     local uppercase_letter_marker = R("AZ")
9990
9991     local roman_marker = function(chars)
9992         local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])
9993         local l, x, v, i = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])
9994         return m^-3
9995             * (c*m + c*d + d^-1 * c^-3)
9996             * (x*c + x*l + l^-1 * x^-3)
9997             * (i*x + i*v + v^-1 * i^-3)
9998     end
9999
10000     local lowercase_roman_marker = roman_marker({"m", "d", "c", "l", "x", "v", "i"})
10001     local uppercase_roman_marker = roman_marker({"M", "D", "C", "L", "X", "V", "I"})
10002
10003     local lowercase_opening_roman_marker = P("i")
10004     local uppercase_opening_roman_marker = P("I")
10005
10006     local digit_marker = parsers.dig * parsers.dig^-8
10007
10008     local markers = {
10009         {lowercase_opening_roman_marker, lowercase_roman_marker},
10010         {uppercase_opening_roman_marker, uppercase_roman_marker},
10011         lowercase_letter_marker,
10012         uppercase_letter_marker,
10013         lowercase_roman_marker,
10014         uppercase_roman_marker,
10015         digit_marker
10016     }
10017
10018     local delims = {
10019         parsers.period,
10020         parsers.rparent
10021     }
10022
10023     local markers_table = combine_markers_and_delims(markers, delims)
10024
10025     local function enumerator(start_marker, _, delimiter_type, interrupting)
10026         local delimiter_range
10027         local allowed_end
10028         if interrupting then
10029             delimiter_range = P("1")
10030             allowed_end = C(parsers.spacechar^1) * #parsers.linechar
10031         else
10032             delimiter_range = start_marker

```

```

10033         allowed_end = C(parsers.spacechar~1) + #(parsers.newline + parsers.eof)
10034     end
10035
10036     return parsers.check_trail
10037         * Ct(C(delimiter_range) * C(delimiter_type))
10038         * allowed_end
10039 end
10040
10041 local starter = join_table_with_func(enumerator, markers_table)
10042
10043 local TightListItem = function(starter)
10044     return parsers.add_indent(starter, "li")
10045         * parsers.indented_content_tight
10046 end
10047
10048 local LooseListItem = function(starter)
10049     return parsers.add_indent(starter, "li")
10050         * parsers.indented_content_loose
10051         * remove_indent("li")
10052 end
10053
10054 local function roman2number(roman)
10055     local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100, ["L"] = 50, ["X"] =
10056     local numeral = 0
10057
10058     local i = 1
10059     local len = string.len(roman)
10060     while i < len do
10061         local z1, z2 = romans[ string.sub(roman, i, i) ], romans[ string.sub(roman,
10062         if z1 < z2 then
10063             numeral = numeral + (z2 - z1)
10064             i = i + 2
10065         else
10066             numeral = numeral + z1
10067             i = i + 1
10068         end
10069     end
10070     if i <= len then numeral = numeral + romans[ string.sub(roman,i,i) ] end
10071     return numeral
10072 end
10073
10074 local function sniffstyle(numstr, delimend)
10075     local numdelim
10076     if delimend == ")" then
10077         numdelim = "OneParen"
10078     elseif delimend == "." then
10079         numdelim = "Period"

```

```

10080     else
10081         numdelim = "Default"
10082     end
10083
10084     local num
10085     num = numstr:match("^([I])$")
10086     if num then
10087         return roman2number(num), "UpperRoman", numdelim
10088     end
10089     num = numstr:match("^([i])$")
10090     if num then
10091         return roman2number(string.upper(num)), "LowerRoman", numdelim
10092     end
10093     num = numstr:match("^([A-Z])$")
10094     if num then
10095         return string.byte(num) - string.byte("A") + 1, "UpperAlpha", numdelim
10096     end
10097     num = numstr:match("^([a-z])$")
10098     if num then
10099         return string.byte(num) - string.byte("a") + 1, "LowerAlpha", numdelim
10100     end
10101     num = numstr:match("^([IVXLCDM]+)")
10102     if num then
10103         return roman2number(num), "UpperRoman", numdelim
10104     end
10105     num = numstr:match("^([ivxlcdm]+)")
10106     if num then
10107         return roman2number(string.upper(num)), "LowerRoman", numdelim
10108     end
10109     return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
10110 end
10111
10112 local function fancylist(items,tight,start)
10113     local startnum, numstyle, numdelim = sniffstyle(start[2][1], start[2][2])
10114     return writer.fancylist(items,tight,
10115                             options.startNumber and startnum or 1,
10116                             numstyle or "Decimal",
10117                             numdelim or "Default")
10118 end
10119
10120 local FancyListOfType = function(start_marker, continuation_marker, delimiter_t
10121     local enumerator_start = enumerator(start_marker, continuation_marker, delimi
10122     local enumerator_cont = enumerator(continuation_marker, continuation_marker,
10123     return Cg(enumerator_start, "listtype")
10124         * (Ct( TightListItem(Cb("listtype"))
10125             * ((parsers.check_minimal_indent / "") * TightListItem(enumerator_co
10126         * Cc(true)

```

```

10127         * -#((parsers.conditionally_indented_blankline~0 / "")
10128         * parsers.check_minimal_indent * enumerator_cont)
10129     + Ct( LooseListItem(Cb("listtype"))
10130         * ((parsers.conditionally_indented_blankline~0 / "")
10131         * (parsers.check_minimal_indent / "") * LooseListItem(enumerator_c
10132         * Cc(false)
10133         ) * Ct(Cb("listtype"))) / fancylist
10134     end
10135
10136     local FancyList = join_table_with_func(FancyListOfType, markers_table)
10137
10138     local Endline   = parsers.newline
10139                     * (parsers.check_minimal_indent
10140                     * -parsers.EndlineExceptions
10141                     + parsers.check_optional_indent
10142                     * -parsers.EndlineExceptions
10143                     * -starter)
10144                     * parsers.spacechar~0
10145                     / writer.soft_line_break
10146
10147     self.update_rule("OrderedList", FancyList)
10148     self.update_rule("Endline", Endline)
10149 end
10150 }
10151 end

```

### 3.1.7.6 Fenced Code

The `extensions.fenced_code` function implements the commonmark fenced code block syntax extension. When the `blank_before_code_fence` parameter is `true`, the syntax extension requires a blank line between a paragraph and the following fenced code block.

When the `allow_attributes` option is `true`, the syntax extension permits attributes following the infostring. When the `allow_raw_blocks` option is `true`, the syntax extension permits the specification of raw blocks using the Pandoc raw attribute syntax extension.

```

10152 M.extensions.fenced_code = function(blank_before_code_fence,
10153                                     allow_attributes,
10154                                     allow_raw_blocks)
10155   return {
10156     name = "built-in fenced_code syntax extension",
10157     extend_writer = function(self)
10158       local options = self.options
10159

```

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.

```

10160     function self.fencedCode(s, i, attr)
10161         if not self.is_writing then return "" end
10162         s = s:gsub("\n$", "")
10163         local buf = {}
10164         if attr ~= nil then
10165             table.insert(buf, {"\\markdownRendererFencedCodeAttributeContextBegin",
10166                             self.attributes(attr)})
10167         end
10168         local name = util.cache_verbatim(options.cacheDir, s)
10169         table.insert(buf, {"\\markdownRendererInputFencedCode{",
10170                             name,"}{" ,self.string(i),"}{" ,self.infostring(i),"}")
10171         if attr ~= nil then
10172             table.insert(buf, "\\markdownRendererFencedCodeAttributeContextEnd{")
10173         end
10174         return buf
10175     end
10176

```

Define `writer->rawBlock` as a function that will transform an input raw block `s` with the raw attribute `attr` to the output format.

```

10177     if allow_raw_blocks then
10178         function self.rawBlock(s, attr)
10179             if not self.is_writing then return "" end
10180             s = s:gsub("\n$", "")
10181             local name = util.cache_verbatim(options.cacheDir, s)
10182             return {"\\markdownRendererInputRawBlock{",
10183                     name,"}{" , self.string(attr),"}"}
10184         end
10185     end
10186     end, extend_reader = function(self)
10187         local parsers = self.parsers
10188         local writer = self.writer
10189
10190         local function captures_geq_length(_,i,a,b)
10191             return #a >= #b and i
10192         end
10193
10194         local function strip_enclosing_whitespaces(str)
10195             return str:gsub("^%s*(.)%s*$", "%1")
10196         end
10197
10198         local tilde_infostring = Cs(Cs((V("HtmlEntity")
10199                                     + parsers.anyescaped
10200                                     - parsers.newline)^0)
10201                                     / strip_enclosing_whitespaces)
10202
10203         local backtick_infostring = Cs(Cs((V("HtmlEntity")

```

```

10204             + (-#(parsers.backslash * parsers.backtick) *
10205                - parsers.newline
10206                - parsers.backtick)^0)
10207             / strip_enclosing_whitespaces)
10208
10209     local fenceindent
10210
10211     local function has_trail(indent_table)
10212         return indent_table ~= nil and
10213            indent_table.trail ~= nil and
10214            next(indent_table.trail) ~= nil
10215     end
10216
10217     local function has_indents(indent_table)
10218         return indent_table ~= nil and
10219            indent_table.indents ~= nil and
10220            next(indent_table.indents) ~= nil
10221     end
10222
10223     local function get_last_indent_name(indent_table)
10224         if has_indents(indent_table) then
10225             return indent_table.indents[#indent_table.indents].name
10226         end
10227     end
10228
10229     local function count_fenced_start_indent(_, _, indent_table, trail)
10230         local last_indent_name = get_last_indent_name(indent_table)
10231         fenceindent = 0
10232         if last_indent_name ~= "li" then
10233             fenceindent = #trail
10234         end
10235         return true
10236     end
10237
10238     local fencehead = function(char, infostring)
10239         return
10240             * Cmt(Cb("indent_info") * parsers.check_trail, count_fence
10241             * Cg(char^3, "fencelength")
10242             * parsers.optionalspace
10243             * infostring
10244             * (parsers.newline + parsers.eof)
10245     end
10246
10247     local fencetail = function(char)
10248         return
10249             parsers.check_trail_no_rem
10250             * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
10251             * parsers.optionalspace * (parsers.newline + parsers.eof)
10252             + parsers.eof

```



```

10251     end
10252
10253     local function process_fenced_line(s, i, indent_table, line_content, is_blank)
10254         local remainder = ""
10255         if has_trail(indent_table) then
10256             remainder = indent_table.trail.internal_remainder
10257         end
10258
10259         if is_blank and get_last_indent_name(indent_table) == "li" then
10260             remainder = ""
10261         end
10262
10263         local str = remainder .. line_content
10264         local index = 1
10265         local remaining = fenceindent
10266
10267         while true do
10268             local c = str:sub(index, index)
10269             if c == " " and remaining > 0 then
10270                 remaining = remaining - 1
10271                 index = index + 1
10272             elseif c == "\t" and remaining > 3 then
10273                 remaining = remaining - 4
10274                 index = index + 1
10275             else
10276                 break
10277             end
10278         end
10279
10280         return true, str:sub(index)
10281     end
10282
10283     local fencedline = function(char)
10284         return Cmt(Cb("indent_info") * C(parsers.line - fencetail(char)) * Cc(false),
10285     end
10286
10287     local blankfencedline = Cmt(Cb("indent_info") * C(parsers.blankline) * Cc(true))
10288
10289     local TildeFencedCode
10290         = fencehead(parsers.tilde, tilde_infostring)
10291         * Cs(((parsers.check_minimal_blank_indent / "") * blankfencedline
10292             + (parsers.check_minimal_indent / "") * fencedline(parsers.tilde))
10293         * ((parsers.check_minimal_indent / "") * fencetail(parsers.tilde) + pars
10294
10295     local BacktickFencedCode
10296         = fencehead(parsers.backtick, backtick_infostring)
10297         * Cs(((parsers.check_minimal_blank_indent / "") * blankfencedline

```

```

10298         + (parsers.check_minimal_indent / "") * fencedline(parsers.backtick
10299         * ((parsers.check_minimal_indent / "") * fencetail(parsers.backtick) + p
10300
10301     local infostring_with_attributes
10302         = Ct(C((parsers.linechar
10303             - ( parsers.optionalspace
10304               * parsers.attributes))^0)
10305           * parsers.optionalspace
10306           * Ct(parsers.attributes))
10307
10308     local FencedCode
10309         = ((TildeFencedCode + BacktickFencedCode)
10310       / function(infostring, code)
10311         local expanded_code = self.expandtabs(code)
10312
10313         if allow_raw_blocks then
10314             local raw_attr = lpeg.match(parsers.raw_attribute,
10315                                       infostring)
10316
10317             if raw_attr then
10318                 return writer.rawBlock(expanded_code, raw_attr)
10319             end
10320         end
10321
10322         local attr = nil
10323         if allow_attributes then
10324             local match = lpeg.match(infostring_with_attributes,
10325                                     infostring)
10326
10327             if match then
10328                 infostring, attr = table.unpack(match)
10329             end
10330         end
10331         return writer.fencedCode(expanded_code, infostring, attr)
10332     end)
10333
10334     self.insert_pattern("Block after Verbatim",
10335                       FencedCode, "FencedCode")
10336
10337     local fencestart
10338     if blank_before_code_fence then
10339         fencestart = parsers.fail
10340     else
10341         fencestart = fencehead(parsers.backtick, backtick_infostring)
10342         + fencehead(parsers.tilde, tilde_infostring)
10343     end
10344
10345     self.update_rule("EndlineExceptions", function(previous_pattern)
10346         if previous_pattern == nil then

```

```

10345         previous_pattern = parsers.EndlineExceptions
10346     end
10347     return previous_pattern + fencestart
10348 end)
10349
10350     self.add_special_character("`")
10351     self.add_special_character("~")
10352 end
10353 }
10354 end

```

### 3.1.7.7 Fenced Divs

The `extensions.fenced_divs` function implements the Pandoc fenced div syntax extension. When the `blank_before_div_fence` parameter is `true`, the syntax extension requires a blank line between a paragraph and the following fenced code block.

```

10355 M.extensions.fenced_divs = function(blank_before_div_fence)
10356   return {
10357     name = "built-in fenced_divs syntax extension",
10358     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.

```

10359     function self.div_begin(attributes)
10360       local start_output = {"\markdownRendererFencedDivAttributeContextBegin\n",
10361                             self.attributes(attributes)}
10362       local end_output = {"\markdownRendererFencedDivAttributeContextEnd{}}
10363       return self.push_attributes("div", attributes, start_output, end_output)
10364     end

```

Define `writer->div_end` as a function that will produce the end of a fenced div in the output format.

```

10365     function self.div_end()
10366       return self.pop_attributes("div")
10367     end
10368 end, extend_reader = function(self)
10369   local parsers = self.parsers
10370   local writer = self.writer

```

Define basic patterns for matching the opening and the closing tag of a div.

```

10371     local fenced_div_infostring
10372         = C((parsers.linechar
10373             - ( parsers.spacechar^1
10374               * parsers.colon^1))^1)
10375
10376     local fenced_div_begin = parsers.nonindentospace
10377         * parsers.colon^3

```

```

10378         * parsers.optionalspace
10379         * fenced_div_infostring
10380         * ( parsers.spacechar^1
10381           * parsers.colon^1)^0
10382         * parsers.optionalspace
10383         * (parsers.newline + parsers.eof)
10384
10385     local fenced_div_end = parsers.nonindentSPACE
10386         * parsers.colon^3
10387         * parsers.optionalspace
10388         * (parsers.newline + parsers.eof)

```

Initialize a named group named `fenced_div_level` for tracking how deep we are nested in divs and the named group `fenced_div_num_opening_indents` for tracking the indent of the starting div fence. The former named group is immutable and should roll back properly when we fail to match a fenced div. The latter is mutable and may contain items from unsuccessful matches on top. However, we always know how many items at the head of the latter we can trust by consulting the former.

```

10389     self.initialize_named_group("fenced_div_level", "0")
10390     self.initialize_named_group("fenced_div_num_opening_indents")
10391
10392     local function increment_div_level()
10393         local function push_indent_table(s, i, indent_table, -- luacheck: ignore s i
10394                                     fenced_div_num_opening_indents, fenced_div_l
10395             fenced_div_level = tonumber(fenced_div_level) + 1
10396             local num_opening_indents = 0
10397             if indent_table.indents ~= nil then
10398                 num_opening_indents = #indent_table.indents
10399             end
10400             fenced_div_num_opening_indents[fenced_div_level] = num_opening_indents
10401             return true, fenced_div_num_opening_indents
10402         end
10403
10404         local function increment_level(s, i, fenced_div_level) -- luacheck: ignore s
10405             fenced_div_level = tonumber(fenced_div_level) + 1
10406             return true, tostring(fenced_div_level)
10407         end
10408
10409         return Cg( Cmt( Cb("indent_info")
10410                       * Cb("fenced_div_num_opening_indents")
10411                       * Cb("fenced_div_level"), push_indent_table)
10412                 , "fenced_div_num_opening_indents")
10413             * Cg( Cmt( Cb("fenced_div_level"), increment_level)
10414                 , "fenced_div_level")
10415         end
10416
10417     local function decrement_div_level()

```

```

10418     local function pop_indent_table(s, i, fenced_div_indent_table, fenced_div_level)
10419         fenced_div_level = tonumber(fenced_div_level)
10420         fenced_div_indent_table[fenced_div_level] = nil
10421         return true, tostring(fenced_div_level - 1)
10422     end
10423
10424     return Cg( Cmt( Cb("fenced_div_num_opening_indents")
10425         * Cb("fenced_div_level"), pop_indent_table)
10426         , "fenced_div_level")
10427 end
10428
10429
10430 local non_fenced_div_block = parsers.check_minimal_indent * V("Block")
10431                             - parsers.check_minimal_indent_and_trail * fenced_d
10432
10433 local non_fenced_div_paragraph = parsers.check_minimal_indent * V("Paragraph")
10434                                 - parsers.check_minimal_indent_and_trail * fenced
10435
10436 local blank = parsers.minimally_indented_blank
10437
10438 local block_separated = parsers.block_sep_group(blank)
10439                         * non_fenced_div_block
10440
10441 local loop_body_pair = parsers.create_loop_body_pair(block_separated,
10442                                                       non_fenced_div_paragraph,
10443                                                       parsers.block_sep_group(t
10444                                                       parsers.par_sep_group(bla
10445
10446 local content_loop = ( non_fenced_div_block
10447                       * loop_body_pair.block^0
10448                       + non_fenced_div_paragraph
10449                       * block_separated
10450                       * loop_body_pair.block^0
10451                       + non_fenced_div_paragraph
10452                       * loop_body_pair.par^0)
10453                       * blank^0
10454
10455 local FencedDiv = fenced_div_begin
10456                 / function (infostring)
10457                     local attr = lpeg.match(Ct(parsers.attributes), infostring)
10458                     if attr == nil then
10459                         attr = {"." .. infostring}
10460                     end
10461                     return attr
10462                 end
10463                 / writer.div_begin
10464                 * increment_div_level()

```

```

10465         * parsers.skipblanklines
10466         * Ct(content_loop)
10467         * parsers.minimally_indented_blank^0
10468         * parsers.check_minimal_indent_and_trail * fenced_div_end
10469         * decrement_div_level()
10470         * (Cc("") / writer.div_end)
10471
10472     self.insert_pattern("Block after Verbatim",
10473                       FencedDiv, "FencedDiv")
10474
10475     self.add_special_character(":")
10476

```

If the `blank_before_div_fence` parameter is `false`, we will have the closing div at the beginning of a line break the current paragraph if we are currently nested in a div and the indentation matches the opening div fence.

```

10477     local function is_inside_div()
10478         local function check_div_level(s, i, fenced_div_level) -- luacheck: ignore s
10479             fenced_div_level = tonumber(fenced_div_level)
10480             return fenced_div_level > 0
10481         end
10482
10483         return Cmt(Cb("fenced_div_level"), check_div_level)
10484     end
10485
10486     local function check_indent()
10487         local function compare_indent(s, i, indent_table, -- luacheck: ignore s i
10488                                     fenced_div_num_opening_indents, fenced_div_level)
10489             fenced_div_level = tonumber(fenced_div_level)
10490             local num_current_indents = (indent_table.current_line_indents ~= nil and
10491                                         #indent_table.current_line_indents) or 0
10492             local num_opening_indents = fenced_div_num_opening_indents[fenced_div_level]
10493             return num_current_indents == num_opening_indents
10494         end
10495
10496         return Cmt( Cb("indent_info")
10497                   * Cb("fenced_div_num_opening_indents")
10498                   * Cb("fenced_div_level"), compare_indent)
10499     end
10500
10501     local fencestart = is_inside_div()
10502                     * fenced_div_end
10503                     * check_indent()
10504
10505     if not blank_before_div_fence then
10506         self.update_rule("EndlineExceptions", function(previous_pattern)
10507             if previous_pattern == nil then

```

```

10508         previous_pattern = parsers.EndlineExceptions
10509     end
10510     return previous_pattern + fencestart
10511 end)
10512 end
10513 end
10514 }
10515 end

```

### 3.1.7.8 Header Attributes

The `extensions.header_attributes` function implements the Pandoc header attribute syntax extension.

```

10516 M.extensions.header_attributes = function()
10517   return {
10518     name = "built-in header_attributes syntax extension",
10519     extend_writer = function()
10520     end, extend_reader = function(self)
10521       local parsers = self.parsers
10522       local writer = self.writer
10523
10524       local function strip_atx_end(s)
10525         return s:gsub("%s+##%s*$", "")
10526       end
10527
10528       local AtxHeading = Cg(parsers.heading_start, "level")
10529         * parsers.optionalspace
10530         * (C(((parsers.linechar
10531           - (parsers.attributes
10532             * parsers.optionalspace
10533             * parsers.newline))
10534           * (parsers.linechar
10535             - parsers.lbrace)^0)^1)
10536         / strip_atx_end
10537         / parsers.parse_heading_text)
10538         * Cg(Ct(parsers.newline
10539           + (parsers.attributes
10540             * parsers.optionalspace
10541             * parsers.newline)), "attributes")
10542         * Cb("level")
10543         * Cb("attributes")
10544         / writer.heading
10545
10546       local function strip_trailing_spaces(s)
10547         return s:gsub("%s*$", "")
10548       end
10549

```

```

10550     local heading_line = (parsers.linechar
10551                          - (parsers.attributes
10552                             * parsers.optionalspace
10553                             * parsers.newline))^1
10554     - parsers.thematic_break_lines
10555
10556     local heading_text = heading_line
10557                       * ((V("Endline") / "\n") * (heading_line - parsers.heading_
10558                       * parsers.newline)^-1
10559
10560     local SetextHeading = parsers.freeze_trail * parsers.check_trail_no_rem
10561                       * #(heading_text
10562                          * (parsers.attributes
10563                             * parsers.optionalspace
10564                             * parsers.newline)^-1
10565                          * parsers.check_minimal_indent * parsers.check_trail *
10566                          * Cs(heading_text) / strip_trailing_spaces
10567                          / parsers.parse_heading_text
10568                          * Cg(Ct((parsers.attributes
10569                             * parsers.optionalspace
10570                             * parsers.newline)^-1), "attributes")
10571                          * parsers.check_minimal_indent_and_trail * parsers.heading_
10572                          * Cb("attributes")
10573                          * parsers.newline
10574                          * parsers.unfreeze_trail
10575                          / writer.heading
10576
10577     local Heading = AtxHeading + SetextHeading
10578     self.update_rule("Heading", Heading)
10579 end
10580 }
10581 end

```

### 3.1.7.9 Inline Code Attributes

The `extensions.inline_code_attributes` function implements the Pandoc inline code attribute syntax extension.

```

10582 M.extensions.inline_code_attributes = function()
10583   return {
10584     name = "built-in inline_code_attributes syntax extension",
10585     extend_writer = function()
10586     end, extend_reader = function(self)
10587       local writer = self.writer
10588
10589       local CodeWithAttributes = parsers.inticks
10590                             * Ct(parsers.attributes)
10591                             / writer.code

```



```

10592
10593     self.insert_pattern("Inline before Code",
10594                          CodeWithAttributes,
10595                          "CodeWithAttributes")
10596   end
10597 }
10598 end

```

### 3.1.7.10 Line Blocks

The `extensions.line_blocks` function implements the Pandoc line block syntax extension.

```

10599 M.extensions.line_blocks = function()
10600   return {
10601     name = "built-in line_blocks syntax extension",
10602     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.

```

10603     function self.lineblock(lines)
10604       if not self.is_writing then return "" end
10605       local buffer = {}
10606       for i = 1, #lines - 1 do
10607         buffer[#buffer + 1] = { lines[i], self.hard_line_break }
10608       end
10609       buffer[#buffer + 1] = lines[#lines]
10610
10611       return {"\\markdownRendererLineBlockBegin\n"
10612              ,buffer,
10613              "\n\\markdownRendererLineBlockEnd "}
10614     end
10615   end, extend_reader = function(self)
10616     local parsers = self.parsers
10617     local writer = self.writer
10618
10619     local LineBlock = Ct(
10620       (Cs(
10621         ( (parsers.pipe * parsers.space)/""
10622          * ((parsers.space)/entities.char_entity("nbsp"))^0
10623          * parsers.linechar^0 * (parsers.newline/"")
10624          * (-parsers.pipe
10625           * (parsers.space^1/" ")
10626           * parsers.linechar^1
10627           * (parsers.newline/"")
10628           )^0
10629          * (parsers.blankline/"")^0
10630         ) / self.parser_functions.parse_inlines)^1) / writer.lineblock
10631

```

```

10632     self.insert_pattern("Block after Blockquote",
10633                          LineBlock, "LineBlock")
10634   end
10635 }
10636 end

```

### 3.1.7.11 Marked text

The `extensions.mark` function implements the Pandoc mark syntax extension.

```

10637 M.extensions.mark = function()
10638   return {
10639     name = "built-in mark syntax extension",
10640     extend_writer = function(self)

```

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

```

10641     function self.mark(s)
10642       if self.flatten_inlines then return s end
10643       return {"\\markdownRendererMark{" , s, "}"}
10644     end
10645   end, extend_reader = function(self)
10646     local parsers = self.parsers
10647     local writer = self.writer
10648
10649     local doubleequals = P("==")
10650
10651     local Mark = parsers.between(V("Inline"), doubleequals, doubleequals)
10652       / function (inlines) return writer.mark(inlines) end
10653
10654     self.add_special_character("=")
10655     self.insert_pattern("Inline before LinkAndEmph",
10656                        Mark, "Mark")
10657   end
10658 }
10659 end

```

### 3.1.7.12 Link Attributes

The `extensions.link_attributes` function implements the Pandoc link attribute syntax extension.

```

10660 M.extensions.link_attributes = function()
10661   return {
10662     name = "built-in link_attributes syntax extension",
10663     extend_writer = function()
10664     end, extend_reader = function(self)
10665       local parsers = self.parsers
10666       local options = self.options
10667

```

The following patterns define link reference definitions with attributes.

```
10668     local define_reference_parser = (parsers.check_trail / "") * parsers.link_label
10669                                     * parsers.spnlc * parsers.url
10670                                     * ( parsers.spnlc_sep * parsers.title * (parsers.
10671                                       * parsers.only_blank
10672                                       + parsers.spnlc_sep * parsers.title * parsers.c
10673                                       + Cc("")) * (parsers.spnlc * Ct(parsers.attribut
10674                                       + Cc("")) * parsers.only_blank)
10675
10676     local ReferenceWithAttributes = define_reference_parser
10677                                     / self.register_link
10678
10679     self.update_rule("Reference", ReferenceWithAttributes)
10680
```

The following patterns define direct and indirect links with attributes.

```
10681
10682     local LinkWithAttributesAndEmph = Ct(parsers.link_and_emph_table * Cg(Cc(true),
10683                                     / self.defer_link_and_emphasis_processing
10684
10685     self.update_rule("LinkAndEmph", LinkWithAttributesAndEmph)
10686
```

The following patterns define autolinks with attributes.

```
10687     local AutoLinkUrlWithAttributes
10688                                     = parsers.auto_link_url
10689                                     * Ct(parsers.attributes)
10690                                     / self.auto_link_url
10691
10692     self.insert_pattern("Inline before AutoLinkUrl",
10693                       AutoLinkUrlWithAttributes,
10694                       "AutoLinkUrlWithAttributes")
10695
10696     local AutoLinkEmailWithAttributes
10697                                     = parsers.auto_link_email
10698                                     * Ct(parsers.attributes)
10699                                     / self.auto_link_email
10700
10701     self.insert_pattern("Inline before AutoLinkEmail",
10702                       AutoLinkEmailWithAttributes,
10703                       "AutoLinkEmailWithAttributes")
10704
10705     if options.relativeReferences then
10706
10707         local AutoLinkRelativeReferenceWithAttributes
10708                                     = parsers.auto_link_relative_reference
10709                                     * Ct(parsers.attributes)
10710                                     / self.auto_link_url
```

```

10711
10712     self.insert_pattern(
10713         "Inline before AutoLinkRelativeReference",
10714         AutoLinkRelativeReferenceWithAttributes,
10715         "AutoLinkRelativeReferenceWithAttributes")
10716
10717     end
10718
10719 end
10720 }
10721 end

```

### 3.1.7.13 Notes

The `extensions.notes` function implements the Pandoc note and inline note syntax extensions. When the `note` parameter is `true`, the Pandoc note syntax extension will be enabled. When the `inline_notes` parameter is `true`, the Pandoc inline note syntax extension will be enabled.

```

10722 M.extensions.notes = function(notes, inline_notes)
10723     assert(notes or inline_notes)
10724     return {
10725         name = "built-in notes syntax extension",
10726         extend_writer = function(self)

```

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

```

10727     function self.note(s)
10728         if self.flatten_inlines then return "" end
10729         return {"\\markdownRendererNote{",s,""}
10730     end
10731 end, extend_reader = function(self)
10732     local parsers = self.parsers
10733     local writer = self.writer
10734
10735     local rawnotes = parsers.rawnotes
10736
10737     if inline_notes then
10738         local InlineNote
10739             = parsers.circumflex
10740             * (parsers.link_label / self.parser_functions.parse_inlines_no_in
10741             / writer.note
10742
10743         self.insert_pattern("Inline after LinkAndEmph",
10744             InlineNote, "InlineNote")
10745     end
10746     if notes then
10747         local function strip_first_char(s)

```

```

10748         return s:sub(2)
10749     end
10750
10751     local RawNoteRef
10752         = #(parsers.lbracket * parsers.circumflex)
10753         * parsers.link_label / strip_first_char
10754
10755     -- like indirect_link
10756     local function lookup_note(ref)
10757         return writer.defer_call(function()
10758             local found = rawnotes[self.normalize_tag(ref)]
10759             if found then
10760                 return writer.note(
10761                     self.parser_functions.parse_blocks_nested(found))
10762             else
10763                 return {"[",
10764                     self.parser_functions.parse_inlines("^" .. ref), "]" }
10765             end
10766         end)
10767     end
10768
10769     local function register_note(ref,rawnote)
10770         local normalized_tag = self.normalize_tag(ref)
10771         if rawnotes[normalized_tag] == nil then
10772             rawnotes[normalized_tag] = rawnote
10773         end
10774         return ""
10775     end
10776
10777     local NoteRef = RawNoteRef / lookup_note
10778
10779     local optionally_indented_line = parsers.check_optional_indent_and_any_trail
10780
10781     local blank = parsers.check_optional_blank_indent_and_any_trail * parsers.opt
10782
10783     local chunk = Cs(parsers.line * (optionally_indented_line - blank)^0)
10784
10785     local indented_blocks = function(bl)
10786         return Cs( bl
10787             * (blank^1 * (parsers.check_optional_indent / ""))
10788             * parsers.check_code_trail * -parsers.blankline * bl)^0)
10789     end
10790
10791     local NoteBlock
10792         = parsers.check_trail_no_rem * RawNoteRef * parsers.colon
10793         * parsers.spnlc * indented_blocks(chunk)
10794         / register_note

```

```

10795
10796     local Reference = NoteBlock + parsers.Reference
10797
10798     self.update_rule("Reference", Reference)
10799     self.insert_pattern("Inline before LinkAndEmph",
10800                         NoteRef, "NoteRef")
10801     end
10802
10803     self.add_special_character("^")
10804 end
10805 }
10806 end

```

### 3.1.7.14 Pipe Tables

The `extensions.pipe_table` function implements the PHP Markdown table syntax extension (also known as pipe tables in Pandoc). When the `table_captions` parameter is `true`, the function also implements the Pandoc table caption syntax extension for table captions. When the `table_attributes` parameter is also `true`, the function also allows attributes to be attached to the (possibly empty) table captions.

```

10807 M.extensions.pipe_tables = function(table_captions, table_attributes)
10808
10809     local function make_pipe_table_rectangular(rows)
10810         local num_columns = #rows[2]
10811         local rectangular_rows = {}
10812         for i = 1, #rows do
10813             local row = rows[i]
10814             local rectangular_row = {}
10815             for j = 1, num_columns do
10816                 rectangular_row[j] = row[j] or ""
10817             end
10818             table.insert(rectangular_rows, rectangular_row)
10819         end
10820         return rectangular_rows
10821     end
10822
10823     local function pipe_table_row(allow_empty_first_column
10824                                   , nonempty_column
10825                                   , column_separator
10826                                   , column)
10827         local row_beginning
10828         if allow_empty_first_column then
10829             row_beginning = -- empty first column
10830                             #(parsers.spacechar^4
10831                               * column_separator)
10832             * parsers.optionalspace

```

```

10833         * column
10834         * parsers.optionalspace
10835         -- non-empty first column
10836         + parsers.nonindentSPACE
10837         * nonempty_column~-1
10838         * parsers.optionalspace
10839     else
10840         row_beginning = parsers.nonindentSPACE
10841         * nonempty_column~-1
10842         * parsers.optionalspace
10843     end
10844
10845     return Ct(row_beginning
10846         * (-- single column with no leading pipes
10847           #(column_separator
10848             * parsers.optionalspace
10849             * parsers.newline)
10850           * column_separator
10851           * parsers.optionalspace
10852           -- single column with leading pipes or
10853           -- more than a single column
10854           + (column_separator
10855             * parsers.optionalspace
10856             * column
10857             * parsers.optionalspace)^1
10858           * (column_separator
10859             * parsers.optionalspace)^-1))
10860     end
10861
10862     return {
10863         name = "built-in pipe_tables syntax extension",
10864         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.

```

10865     function self.table(rows, caption, attributes)
10866         if not self.is_writing then return "" end
10867         local buffer = {}
10868         if attributes ~= nil then
10869             table.insert(buffer,
10870                 "\\markdownRendererTableAttributeContextBegin\n")
10871             table.insert(buffer, self.attributes(attributes))
10872         end
10873         table.insert(buffer,
10874             {"\\markdownRendererTable{",
10875                 caption or "", "}{" , #rows - 1, "}{" ,

```

```

10876             #rows[1], "}")
10877     local temp = rows[2] -- put alignments on the first row
10878     rows[2] = rows[1]
10879     rows[1] = temp
10880     for i, row in ipairs(rows) do
10881         table.insert(buffer, "{")
10882         for _, column in ipairs(row) do
10883             if i > 1 then -- do not use braces for alignments
10884                 table.insert(buffer, "{")
10885             end
10886             table.insert(buffer, column)
10887             if i > 1 then
10888                 table.insert(buffer, "}")
10889             end
10890         end
10891         table.insert(buffer, "}")
10892     end
10893     if attributes ~= nil then
10894         table.insert(buffer,
10895             "\\markdownRendererTableAttributeContextEnd{")
10896     end
10897     return buffer
10898 end
10899 end, extend_reader = function(self)
10900     local parsers = self.parsers
10901     local writer = self.writer
10902
10903     local table_hline_separator = parsers.pipe + parsers.plus
10904
10905     local table_hline_column = (parsers.dash
10906         - #(parsers.dash
10907             * (parsers.spacechar
10908                 + table_hline_separator
10909                 + parsers.newline)))^1
10910     * (parsers.colon * Cc("r")
10911         + parsers.dash * Cc("d"))
10912     + parsers.colon
10913     * (parsers.dash
10914         - #(parsers.dash
10915             * (parsers.spacechar
10916                 + table_hline_separator
10917                 + parsers.newline)))^1
10918     * (parsers.colon * Cc("c")
10919         + parsers.dash * Cc("l"))
10920
10921     local table_hline = pipe_table_row(false
10922         , table_hline_column

```



```

10923         , table_hline_separator
10924         , table_hline_column)
10925
10926     local table_caption_beginning = (parsers.check_minimal_blank_indent_and_any_tra
10927         * parsers.optionalspace * parsers.newline)^0
10928         * parsers.check_minimal_indent_and_trail
10929         * (P("Table")^-1 * parsers.colon)
10930         * parsers.optionalspace
10931
10932     local function strip_trailing_spaces(s)
10933         return s:gsub("%s*$", "")
10934     end
10935
10936     local table_row = pipe_table_row(true
10937         , (C((parsers.linechar - parsers.pipe)^1)
10938         / strip_trailing_spaces
10939         / self.parser_functions.parse_inlines)
10940         , parsers.pipe
10941         , (C((parsers.linechar - parsers.pipe)^0)
10942         / strip_trailing_spaces
10943         / self.parser_functions.parse_inlines))
10944
10945     local table_caption
10946     if table_captions then
10947         table_caption = #table_caption_beginning
10948             * table_caption_beginning
10949         if table_attributes then
10950             table_caption = table_caption
10951                 * (C(((( parsers.linechar
10952                     - (parsers.attributes
10953                     * parsers.optionalspace
10954                     * parsers.newline
10955                     * -( parsers.optionalspace
10956                     * parsers.linechar)))
10957                     + ( parsers.newline
10958                     * #( parsers.optionalspace
10959                     * parsers.linechar)
10960                     * C(parsers.optionalspace) / writer.space))
10961                     * (parsers.linechar
10962                     - parsers.lbrace)^0)^1)
10963                     / self.parser_functions.parse_inlines)
10964                 * (parsers.newline
10965                     + ( Ct(parsers.attributes)
10966                     * parsers.optionalspace
10967                     * parsers.newline))
10968         else
10969             table_caption = table_caption

```

```

10970             * C(( parsers.linechar
10971                 + ( parsers.newline
10972                   * #( parsers.optionalspace
10973                     * parsers.linechar)
10974                   * C(parsers.optionalspace) / writer.space))^1)
10975             / self.parser_functions.parse_inlines
10976             * parsers.newline
10977         end
10978     else
10979         table_caption = parsers.fail
10980     end
10981
10982     local PipeTable = Ct(table_row * parsers.newline * (parsers.check_minimal_indent
10983                 * table_hline * parsers.newline
10984                 * ((parsers.check_minimal_indent / {}) * table_row * parsers.
10985                 / make_pipe_table_rectangular
10986                 * table_caption^-1
10987                 / writer.table
10988
10989         self.insert_pattern("Block after Blockquote",
10990             PipeTable, "PipeTable")
10991     end
10992 }
10993 end

```

### 3.1.7.15 Raw Attributes

The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```

10994 M.extensions.raw_inline = function()
10995     return {
10996         name = "built-in raw_inline syntax extension",
10997         extend_writer = function(self)
10998             local options = self.options
10999

```

Define `writer->rawInline` as a function that will transform an input inline raw span `s` with the raw attribute `attr` to the output format.

```

11000         function self.rawInline(s, attr)
11001             if not self.is_writing then return "" end
11002             if self.flatten_inlines then return s end
11003             local name = util.cache_verbatim(options.cacheDir, s)
11004             return {"\\markdownRendererInputRawInline{" ,
11005                 name,"}{" , self.string(attr),""}
11006         end
11007     end, extend_reader = function(self)
11008         local writer = self.writer
11009

```

```

11010     local RawInline = parsers.inticks
11011                 * parsers.raw_attribute
11012                 / writer.rawInline
11013
11014     self.insert_pattern("Inline before Code",
11015                       RawInline, "RawInline")
11016   end
11017 }
11018 end

```

### 3.1.7.16 Strike-Through

The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```

11019 M.extensions.strike_through = function()
11020   return {
11021     name = "built-in strike_through syntax extension",
11022     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.

```

11023     function self.strike_through(s)
11024       if self.flatten_inlines then return s end
11025       return {"\\markdownRendererStrikeThrough{" ,s,"}"}
11026     end
11027   end, extend_reader = function(self)
11028     local parsers = self.parsers
11029     local writer = self.writer
11030
11031     local StrikeThrough = (
11032       parsers.between(parsers.Inline, parsers.doubletildes,
11033                     parsers.doubletildes)
11034     ) / writer.strike_through
11035
11036     self.insert_pattern("Inline after LinkAndEmph",
11037                       StrikeThrough, "StrikeThrough")
11038
11039     self.add_special_character("~")
11040   end
11041 }
11042 end

```

### 3.1.7.17 Subscripts

The `extensions.subscripts` function implements the Pandoc subscript syntax extension.

```

11043 M.extensions.subscripts = function()
11044   return {

```

```

11045     name = "built-in subscripts syntax extension",
11046     extend_writer = function(self)

```

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

```

11047         function self.subscript(s)
11048             if self.flatten_inlines then return s end
11049             return {"\\markdownRendererSubscript{" ,s,"}"}
11050         end
11051     end, extend_reader = function(self)
11052         local parsers = self.parsers
11053         local writer = self.writer
11054
11055         local Subscript = (
11056             parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
11057         ) / writer.subscript
11058
11059         self.insert_pattern("Inline after LinkAndEmph",
11060                             Subscript, "Subscript")
11061
11062         self.add_special_character("~")
11063     end
11064 }
11065 end

```

### 3.1.7.18 Superscripts

The `extensions.superscripts` function implements the Pandoc superscript syntax extension.

```

11066 M.extensions.superscripts = function()
11067     return {
11068         name = "built-in superscripts syntax extension",
11069         extend_writer = function(self)

```

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

```

11070         function self.superscript(s)
11071             if self.flatten_inlines then return s end
11072             return {"\\markdownRendererSuperscript{" ,s,"}"}
11073         end
11074     end, extend_reader = function(self)
11075         local parsers = self.parsers
11076         local writer = self.writer
11077
11078         local Superscript = (
11079             parsers.between(parsers.Str, parsers.circumflex, parsers.circumflex)
11080         ) / writer.superscript
11081

```

```

11082     self.insert_pattern("Inline after LinkAndEmph",
11083                          Superscript, "Superscript")
11084
11085     self.add_special_character("^")
11086   end
11087 }
11088 end

```

### 3.1.7.19 T<sub>E</sub>X Math

The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```

11089 M.extensions.tex_math = function(tex_math_dollars,
11090                                   tex_math_single_backslash,
11091                                   tex_math_double_backslash)
11092   return {
11093     name = "built-in tex_math syntax extension",
11094     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.

```

11095     function self.display_math(s)
11096       if self.flatten_inlines then return s end
11097       return {"\\markdownRendererDisplayMath{" ,self.math(s),"}"}
11098     end

```

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

```

11099     function self.inline_math(s)
11100       if self.flatten_inlines then return s end
11101       return {"\\markdownRendererInlineMath{" ,self.math(s),"}"}
11102     end
11103   end, extend_reader = function(self)
11104     local parsers = self.parsers
11105     local writer = self.writer
11106
11107     local function between(p, starter, ender)
11108       return (starter * Cs(p * (p - ender)^0) * ender)
11109     end
11110
11111     local function strip_preceding_whitespaces(str)
11112       return str:gsub("^%s*(.-)$", "%1")
11113     end
11114
11115     local allowed_before_closing = B( parsers.backslash * parsers.any
11116                                       + parsers.any * (parsers.any - parsers.backslash)
11117
11118     local allowed_before_closing_no_space = B( parsers.backslash * parsers.any

```

```

11119                                     + parsers.any * (parsers.nonspacechar
11120

```

The following patterns implement the Pandoc dollar math syntax extension.

```

11121     local dollar_math_content = (parsers.newline * (parsers.check_optional_indent /
11122     + parsers.backslash^-1
11123     * parsers.linechar)
11124     - parsers.blankline^2
11125     - parsers.dollar
11126
11127     local inline_math_opening_dollars = parsers.dollar
11128     * #(parsers.nonspacechar)
11129
11130     local inline_math_closing_dollars = allowed_before_closing_no_space
11131     * parsers.dollar
11132     * -(parsers.digit)
11133
11134     local inline_math_dollars = between(Cs( dollar_math_content),
11135     inline_math_opening_dollars,
11136     inline_math_closing_dollars)
11137
11138     local display_math_opening_dollars = parsers.dollar
11139     * parsers.dollar
11140
11141     local display_math_closing_dollars = parsers.dollar
11142     * parsers.dollar
11143
11144     local display_math_dollars = between(Cs( dollar_math_content),
11145     display_math_opening_dollars,
11146     display_math_closing_dollars)

```

The following patterns implement the Pandoc single and double backslash math syntax extensions.

```

11147     local backslash_math_content = (parsers.newline * (parsers.check_optional_inde
11148     + parsers.linechar)
11149     - parsers.blankline^2

```

The following patterns implement the Pandoc double backslash math syntax extension.

```

11150     local inline_math_opening_double = parsers.backslash
11151     * parsers.backslash
11152     * parsers.lparent
11153
11154     local inline_math_closing_double = allowed_before_closing
11155     * parsers.spacechar^0
11156     * parsers.backslash
11157     * parsers.backslash
11158     * parsers.rparent

```

```

11159
11160     local inline_math_double = between(Cs( backslash_math_content),
11161                                       inline_math_opening_double,
11162                                       inline_math_closing_double)
11163                                       / strip_preceding_whitespaces
11164
11165     local display_math_opening_double = parsers.backslash
11166                                       * parsers.backslash
11167                                       * parsers.lbracket
11168
11169     local display_math_closing_double = allowed_before_closing
11170                                       * parsers.spacechar^0
11171                                       * parsers.backslash
11172                                       * parsers.backslash
11173                                       * parsers.rbracket
11174
11175     local display_math_double = between(Cs( backslash_math_content),
11176                                         display_math_opening_double,
11177                                         display_math_closing_double)
11178                                         / strip_preceding_whitespaces

```

The following patterns implement the Pandoc single backslash math syntax extension.

```

11179     local inline_math_opening_single = parsers.backslash
11180                                       * parsers.lparent
11181
11182     local inline_math_closing_single = allowed_before_closing
11183                                       * parsers.spacechar^0
11184                                       * parsers.backslash
11185                                       * parsers.rparent
11186
11187     local inline_math_single = between(Cs( backslash_math_content),
11188                                       inline_math_opening_single,
11189                                       inline_math_closing_single)
11190                                       / strip_preceding_whitespaces
11191
11192     local display_math_opening_single = parsers.backslash
11193                                       * parsers.lbracket
11194
11195     local display_math_closing_single = allowed_before_closing
11196                                       * parsers.spacechar^0
11197                                       * parsers.backslash
11198                                       * parsers.rbracket
11199
11200     local display_math_single = between(Cs( backslash_math_content),
11201                                       display_math_opening_single,
11202                                       display_math_closing_single)
11203                                       / strip_preceding_whitespaces
11204

```

```

11205     local display_math = parsers.fail
11206
11207     local inline_math = parsers.fail
11208
11209     if tex_math_dollars then
11210         display_math = display_math + display_math_dollars
11211         inline_math = inline_math + inline_math_dollars
11212     end
11213
11214     if tex_math_double_backslash then
11215         display_math = display_math + display_math_double
11216         inline_math = inline_math + inline_math_double
11217     end
11218
11219     if tex_math_single_backslash then
11220         display_math = display_math + display_math_single
11221         inline_math = inline_math + inline_math_single
11222     end
11223
11224     local TexMath = display_math / writer.display_math
11225                   + inline_math / writer.inline_math
11226
11227     self.insert_pattern("Inline after LinkAndEmph",
11228                       TexMath, "TexMath")
11229
11230     if tex_math_dollars then
11231         self.add_special_character("$")
11232     end
11233
11234     if tex_math_single_backslash or tex_math_double_backslash then
11235         self.add_special_character("\\")
11236         self.add_special_character("[")
11237         self.add_special_character("]")
11238         self.add_special_character("(")
11239         self.add_special_character("(")
11240     end
11241 end
11242 }
11243 end

```

### 3.1.7.20 YAML Metadata

The `extensions.jekyll_data` function implements the Pandoc YAML metadata block syntax extension. When the `expect_jekyll_data` parameter is `true`, then a markdown document may begin directly with YAML metadata and may contain nothing but YAML metadata.

```

11244 M.extensions.jekyll_data = function(expect_jekyll_data)

```



```

11245 return {
11246   name = "built-in jekyll_data syntax extension",
11247   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`.

```

11248   function self.jekyllData(d, t, p)
11249     if not self.is_writing then return "" end
11250
11251     local buf = {}
11252
11253     local keys = {}
11254     for k, _ in pairs(d) do
11255       table.insert(keys, k)
11256     end

```

For reproducibility, sort the keys. For mixed string-and-numeric keys, sort numeric keys before string keys.

```

11257     table.sort(keys, function(first, second)
11258       if type(first) ~= type(second) then
11259         return type(first) < type(second)
11260       else
11261         return first < second
11262       end
11263     end)
11264
11265     if not p then
11266       table.insert(buf, "\\markdownRendererJekyllDataBegin")
11267     end
11268
11269     local is_sequence = false
11270     if #d > 0 and #d == #keys then
11271       for i=1, #d do
11272         if d[i] == nil then
11273           goto not_a_sequence
11274         end
11275       end
11276       is_sequence = true
11277     end
11278     ::not_a_sequence::
11279
11280     if is_sequence then
11281       table.insert(buf, "\\markdownRendererJekyllDataSequenceBegin{")
11282       table.insert(buf, self.identifier(p or "null"))

```

```

11283         table.insert(buf, "}{")
11284         table.insert(buf, #keys)
11285         table.insert(buf, "}")
11286     else
11287         table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
11288         table.insert(buf, self.identified(p or "null"))
11289         table.insert(buf, "}{")
11290         table.insert(buf, #keys)
11291         table.insert(buf, "}")
11292     end
11293
11294     for _, k in ipairs(keys) do
11295         local v = d[k]
11296         local typ = type(v)
11297         k = tostring(k or "null")
11298         if typ == "table" and next(v) ~= nil then
11299             table.insert(
11300                 buf,
11301                 self.jekyllData(v, t, k)
11302             )
11303         else
11304             k = self.identified(k)
11305             v = tostring(v)
11306             if typ == "boolean" then
11307                 table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
11308                 table.insert(buf, k)
11309                 table.insert(buf, "}{")
11310                 table.insert(buf, v)
11311                 table.insert(buf, "}")
11312             elseif typ == "number" then
11313                 table.insert(buf, "\\markdownRendererJekyllDataNumber{")
11314                 table.insert(buf, k)
11315                 table.insert(buf, "}{")
11316                 table.insert(buf, v)
11317                 table.insert(buf, "}")
11318             elseif typ == "string" then
11319                 table.insert(buf, "\\markdownRendererJekyllDataString{")
11320                 table.insert(buf, k)
11321                 table.insert(buf, "}{")
11322                 table.insert(buf, t(v))
11323                 table.insert(buf, "}")
11324             elseif typ == "table" then
11325                 table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
11326                 table.insert(buf, k)
11327                 table.insert(buf, "}")
11328             else
11329                 error(format("Unexpected type %s for value of " ..

```

```

11330             "YAML key %s", typ, k))
11331         end
11332     end
11333 end
11334
11335     if is_sequence then
11336         table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
11337     else
11338         table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
11339     end
11340
11341     if not p then
11342         table.insert(buf, "\\markdownRendererJekyllDataEnd")
11343     end
11344
11345     return buf
11346 end
11347 end, extend_reader = function(self)
11348     local parsers = self.parsers
11349     local writer = self.writer
11350
11351     local JekyllData
11352         = Cmt( C((parsers.line - P("----") - P("..."))^0)
11353             , function(s, i, text) -- luacheck: ignore s i
11354                 local data
11355                 local ran_ok, _ = pcall(function()
11356                     -- TODO: Replace with `require("tinyyaml")` in TeX Live
11357                     local tinyyaml = require("markdown-tinyyaml")
11358                     data = tinyyaml.parse(text, {timestamps=false})
11359                 end)
11360                 if ran_ok and data ~= nil then
11361                     return true, writer.jekyllData(data, function(s)
11362                         return self.parser_functions.parse_blocks_nested(s)
11363                     end, nil)
11364                 else
11365                     return false
11366                 end
11367             end
11368         )
11369
11370     local UnexpectedJekyllData
11371         = P("----")
11372         * parsers.blankline / 0
11373         * #(-parsers.blankline) -- if followed by blank, it's thematic b
11374         * JekyllData
11375         * (P("----") + P("..."))
11376

```

```

11377     local ExpectedJekyllData
11378         = ( P("----")
11379             * parsers.blankline / 0
11380             * #(-parsers.blankline) -- if followed by blank, it's thematic
11381             )^-1
11382             * JekyllData
11383             * (P("----") + P("..."))^-1
11384
11385     self.insert_pattern("Block before Blockquote",
11386                       UnexpectedJekyllData, "UnexpectedJekyllData")
11387     if expect_jekyll_data then
11388         self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
11389     end
11390 end
11391 }
11392 end

```

### 3.1.8 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.

```

11393 function M.new(options)

```

Make the `options` table inherit from the `defaultOptions` table.

```

11394     options = options or {}
11395     setmetatable(options, { __index = function (_, key)
11396         return defaultOptions[key] end })

```

If the singleton cache contains a conversion function for the same `options`, reuse it.

```

11397     if options.singletonCache and singletonCache.convert then
11398         for k, v in pairs(defaultOptions) do
11399             if type(v) == "table" then
11400                 for i = 1, math.max(#singletonCache.options[k], #options[k]) do
11401                     if singletonCache.options[k][i] ~= options[k][i] then
11402                         goto miss
11403                     end
11404                 end
11405             elseif singletonCache.options[k] ~= options[k] then
11406                 goto miss
11407             end
11408         end
11409         return singletonCache.convert
11410     end
11411     ::miss::

```

Apply built-in syntax extensions based on `options`.

```

11412     local extensions = {}

```

```

11413
11414 if options.bracketedSpans then
11415     local bracketed_spans_extension = M.extensions.bracketed_spans()
11416     table.insert(extensions, bracketed_spans_extension)
11417 end
11418
11419 if options.contentBlocks then
11420     local content_blocks_extension = M.extensions.content_blocks(
11421         options.contentBlocksLanguageMap)
11422     table.insert(extensions, content_blocks_extension)
11423 end
11424
11425 if options.definitionLists then
11426     local definition_lists_extension = M.extensions.definition_lists(
11427         options.tightLists)
11428     table.insert(extensions, definition_lists_extension)
11429 end
11430
11431 if options.fencedCode then
11432     local fenced_code_extension = M.extensions.fenced_code(
11433         options.blankBeforeCodeFence,
11434         options.fencedCodeAttributes,
11435         options.rawAttribute)
11436     table.insert(extensions, fenced_code_extension)
11437 end
11438
11439 if options.fencedDivs then
11440     local fenced_div_extension = M.extensions.fenced_divs(
11441         options.blankBeforeDivFence)
11442     table.insert(extensions, fenced_div_extension)
11443 end
11444
11445 if options.headerAttributes then
11446     local header_attributes_extension = M.extensions.header_attributes()
11447     table.insert(extensions, header_attributes_extension)
11448 end
11449
11450 if options.inlineCodeAttributes then
11451     local inline_code_attributes_extension =
11452         M.extensions.inline_code_attributes()
11453     table.insert(extensions, inline_code_attributes_extension)
11454 end
11455
11456 if options.jekyllData then
11457     local jekyll_data_extension = M.extensions.jekyll_data(
11458         options.expectJekyllData)
11459     table.insert(extensions, jekyll_data_extension)

```

```

11460 end
11461
11462 if options.linkAttributes then
11463     local link_attributes_extension =
11464         M.extensions.link_attributes()
11465     table.insert(extensions, link_attributes_extension)
11466 end
11467
11468 if options.lineBlocks then
11469     local line_block_extension = M.extensions.line_blocks()
11470     table.insert(extensions, line_block_extension)
11471 end
11472
11473 if options.mark then
11474     local mark_extension = M.extensions.mark()
11475     table.insert(extensions, mark_extension)
11476 end
11477
11478 if options.pipeTables then
11479     local pipe_tables_extension = M.extensions.pipe_tables(
11480         options.tableCaptions, options.tableAttributes)
11481     table.insert(extensions, pipe_tables_extension)
11482 end
11483
11484 if options.rawAttribute then
11485     local raw_inline_extension = M.extensions.raw_inline()
11486     table.insert(extensions, raw_inline_extension)
11487 end
11488
11489 if options.strikeThrough then
11490     local strike_through_extension = M.extensions.strike_through()
11491     table.insert(extensions, strike_through_extension)
11492 end
11493
11494 if options.subscripts then
11495     local subscript_extension = M.extensions.subscripts()
11496     table.insert(extensions, subscript_extension)
11497 end
11498
11499 if options.superscripts then
11500     local superscript_extension = M.extensions.superscripts()
11501     table.insert(extensions, superscript_extension)
11502 end
11503
11504 if options.texMathDollars or
11505     options.texMathSingleBackslash or
11506     options.texMathDoubleBackslash then

```

```

11507     local tex_math_extension = M.extensions.tex_math(
11508         options.texMathDollars,
11509         options.texMathSingleBackslash,
11510         options.texMathDoubleBackslash)
11511     table.insert(extensions, tex_math_extension)
11512 end
11513
11514 if options.notes or options.inlineNotes then
11515     local notes_extension = M.extensions.notes(
11516         options.notes, options.inlineNotes)
11517     table.insert(extensions, notes_extension)
11518 end
11519
11520 if options.citations then
11521     local citations_extension = M.extensions.citations(options.citationNbsps)
11522     table.insert(extensions, citations_extension)
11523 end
11524
11525 if options.fancyLists then
11526     local fancy_lists_extension = M.extensions.fancy_lists()
11527     table.insert(extensions, fancy_lists_extension)
11528 end

```

Apply user-defined syntax extensions based on `options.extensions`.

```

11529 for _, user_extension_filename in ipairs(options.extensions) do
11530     local user_extension = (function(filename)

```

First, load and compile the contents of the user-defined syntax extension.

```

11531     local pathname = kpse.lookup(filename)
11532     local input_file = assert(io.open(pathname, "r"),
11533         [[Could not open user-defined syntax extension "]]
11534         .. pathname .. [[[" for reading]])
11535     local input = assert(input_file:read("*a"))
11536     assert(input_file:close())
11537     local user_extension, err = load([[
11538         local sandbox = {}
11539         setmetatable(sandbox, {__index = _G})
11540         _ENV = sandbox
11541     ]]) .. input)()
11542     assert(user_extension,
11543         [[Failed to compile user-defined syntax extension "]]
11544         .. pathname .. [[[" : ]] .. (err or [[]]))

```

Then, validate the user-defined syntax extension.

```

11545     assert(user_extension.api_version ~= nil,
11546         [[User-defined syntax extension "]] .. pathname
11547         .. [[[" does not specify mandatory field "api_version"]])
11548     assert(type(user_extension.api_version) == "number",

```

```

11549     [[User-defined syntax extension "]] .. pathname
11550     .. [[[" specifies field "api_version" of type "]]
11551     .. type(user_extension.api_version)
11552     .. [[[" but "number" was expected]]]
11553     assert(user_extension.api_version > 0
11554            and user_extension.api_version <= metadata.user_extension_api_version,
11555            [[User-defined syntax extension "]] .. pathname
11556            .. [[[" uses syntax extension API version "]]
11557            .. user_extension.api_version .. [[ but markdown.lua ]]
11558            .. metadata.version .. [[ uses API version ]]
11559            .. metadata.user_extension_api_version
11560            .. [[[, which is incompatible]])
11561
11562     assert(user_extension.grammar_version ~= nil,
11563            [[User-defined syntax extension "]] .. pathname
11564            .. [[[" does not specify mandatory field "grammar_version"]])
11565     assert(type(user_extension.grammar_version) == "number",
11566            [[User-defined syntax extension "]] .. pathname
11567            .. [[[" specifies field "grammar_version" of type "]]
11568            .. type(user_extension.grammar_version)
11569            .. [[[" but "number" was expected]])
11570     assert(user_extension.grammar_version == metadata.grammar_version,
11571            [[User-defined syntax extension "]] .. pathname
11572            .. [[[" uses grammar version "]] .. user_extension.grammar_version
11573            .. [[ but markdown.lua ]] .. metadata.version
11574            .. [[ uses grammar version ]] .. metadata.grammar_version
11575            .. [[[, which is incompatible]])
11576
11577     assert(user_extension.finalize_grammar ~= nil,
11578            [[User-defined syntax extension "]] .. pathname
11579            .. [[[" does not specify mandatory "finalize_grammar" field]])
11580     assert(type(user_extension.finalize_grammar) == "function",
11581            [[User-defined syntax extension "]] .. pathname
11582            .. [[[" specifies field "finalize_grammar" of type "]]
11583            .. type(user_extension.finalize_grammar)
11584            .. [[[" but "function" was expected]])

```

Finally, cast the user-defined syntax extension to the internal format of user extensions used by the Markdown package (see Section 3.1.7.)

```

11585     local extension = {
11586         name = [[user-defined "]] .. pathname .. [[[" syntax extension]],
11587         extend_reader = user_extension.finalize_grammar,
11588         extend_writer = function() end,
11589     }
11590     return extension
11591 end)(user_extension_filename)
11592 table.insert(extensions, user_extension)

```



```

11593 end
    Produce a conversion function from markdown to plain TeX.
11594 local writer = M.writer.new(options)
11595 local reader = M.reader.new(writer, options)
11596 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.
11597 collectgarbage("collect")
    Update the singleton cache.
11598 if options.singletonCache then
11599     local singletonCacheOptions = {}
11600     for k, v in pairs(options) do
11601         singletonCacheOptions[k] = v
11602     end
11603     setmetatable(singletonCacheOptions,
11604         { __index = function (_, key)
11605             return defaultOptions[key] end })
11606     singletonCache.options = singletonCacheOptions
11607     singletonCache.convert = convert
11608 end
    Return the conversion function from markdown to plain TeX.
11609 return convert
11610 end
11611
11612 return M

```

### 3.1.9 Command-Line Implementation

The command-line implementation provides the actual conversion routine for the command-line interface described in Section 2.1.7.

```

11613
11614 local input
11615 if input_filename then
11616     local input_file = assert(io.open(input_filename, "r"),
11617         [[Could not open file ]] .. input_filename .. [[ for reading]])
11618     input = assert(input_file:read("*a"))
11619     assert(input_file:close())
11620 else
11621     input = assert(io.read("*a"))
11622 end
11623

```

First, ensure that the `options.cacheDir` directory exists.

```

11624 local lfs = require("lfs")

```

```

11625 if options.cacheDir and not lfs.isdir(options.cacheDir) then
11626   assert(lfs.mkdir(options["cacheDir"]))
11627 end

```

If Kpathsea has not been loaded before or if Lua<sub>TEX</sub> has not yet been initialized, configure Kpathsea on top of loading it.

```

11628 local kpse
11629 (function()
11630   local should_initialize = package.loaded.kpse == nil
11631                       or tex.initialize ~= nil
11632   kpse = require("kpse")
11633   if should_initialize then
11634     kpse.set_program_name("luatex")
11635   end
11636 end)()
11637 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.

```

11638 if metadata.version ~= md.metadata.version then
11639   warn("markdown-cli.lua " .. metadata.version .. " used with " ..
11640       "markdown.lua " .. md.metadata.version .. ".")
11641 end
11642 local convert = md.new(options)
11643 local output = convert(input)
11644
11645 if output_filename then
11646   local output_file = assert(io.open(output_filename, "w"),
11647   [[Could not open file ]] .. output_filename .. [[ for writing]])
11648   assert(output_file:write(output))
11649   assert(output_file:close())
11650 else
11651   assert(io.write(output))
11652 end

```

Remove the `options.cacheDir` directory if it is empty.

```

11653 if options.cacheDir then
11654   lfs.rmdir(options["cacheDir"])
11655 end

```

### 3.2 Plain $\text{\TeX}$ Implementation

The plain  $\text{\TeX}$  implementation provides macros for the interfacing between  $\text{\TeX}$  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  $\text{\TeX}$  exposed by the plain  $\text{\TeX}$  interface (see Section 2.2).

### 3.2.1 Logging Facilities

```
11656 \ExplSyntaxOn
11657 \cs_if_free:NT
11658   \markdownInfo
11659   {
11660     \cs_new:Npn
11661       \markdownInfo #1
11662       {
11663         \msg_info:nne
11664           { markdown }
11665           { generic-message }
11666           { #1 }
11667       }
11668   }
11669 \cs_if_free:NT
11670   \markdownWarning
11671   {
11672     \cs_new:Npn
11673       \markdownWarning #1
11674       {
11675         \msg_warning:nne
11676           { markdown }
11677           { generic-message }
11678           { #1 }
11679       }
11680   }
11681 \cs_if_free:NT
11682   \markdownError
11683   {
11684     \cs_new:Npn
11685       \markdownError #1 #2
11686       {
11687         \msg_error:nnee
11688           { markdown }
11689           { generic-message-with-help-text }
11690           { #1 }
11691           { #2 }
11692       }
11693   }
11694 \msg_new:nnn
11695   { markdown }
11696   { generic-message }
11697   { #1 }
11698 \msg_new:nnnn
11699   { markdown }
11700   { generic-message-with-help-text }
```

```

11701 { #1 }
11702 { #2 }
11703 \cs_generate_variant:Nn
11704 \msg_info:nnn
11705 { nne }
11706 \cs_generate_variant:Nn
11707 \msg_warning:nnn
11708 { nne }
11709 \cs_generate_variant:Nn
11710 \msg_error:nnnn
11711 { nnee }
11712 \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.

```

11713 \ExplSyntaxOn
11714 \prop_new:N \g_@@_plain_tex_loaded_themes_linenos_prop
11715 \cs_new:Nn
11716 \@@_plain_tex_load_theme:nn
11717 {
11718   \prop_get:NnNTF
11719     \g_@@_plain_tex_loaded_themes_linenos_prop
11720     { #1 }
11721     \l_tmpa_tl
11722     {
11723       \msg_warning:nnnV
11724         { markdown }
11725         { repeatedly-loaded-plain-tex-theme }
11726         { #1 }
11727       \l_tmpa_tl
11728     }
11729     {
11730       \msg_info:nnn
11731         { markdown }
11732         { loading-plain-tex-theme }
11733         { #1 }
11734       \prop_gput:Nnx
11735         \g_@@_plain_tex_loaded_themes_linenos_prop
11736         { #1 }
11737         { \tex_the:D \tex_inputlineno:D }
11738       \file_input:n
11739         { markdown theme #2 }
11740     }
11741 }

```

```

11742 \msg_new:nnn
11743   { markdown }
11744   { loading-plain-tex-theme }
11745   { Loading~plain~TeX~Markdown~theme~#1 }
11746 \msg_new:nnn
11747   { markdown }
11748   { repeatedly-loaded-plain-tex-theme }
11749   {
11750     Plain~TeX~Markdown~theme~#1~was~previously~
11751     loaded~on~line~#2,~not~loading~it~again
11752   }
11753 \cs_generate_variant:Nn
11754   \prop_gput:Nnn
11755   { Nnx }
11756 \cs_gset_eq:NN
11757   \@@_load_theme:nn
11758   \@@_plain_tex_load_theme:nn
11759 \cs_generate_variant:Nn
11760   \@@_load_theme:nn
11761   { nV }

```

Developers can use the `\markdownLoadPlainTeXTheme` macro to load a corresponding plain `TeX` theme from within themes for higher-level `TeX` formats such as `LATEX` and `ConTEXt`.

```

11762 \cs_new:Npn
11763   \markdownLoadPlainTeXTheme
11764   {

```

First, we extract the name of the current theme from the `\g_@@_current_theme_tl` macro.

```

11765     \tl_set:NV
11766       \l_tmpa_tl
11767       \g_@@_current_theme_tl
11768     \tl_reverse:N
11769       \l_tmpa_tl
11770     \tl_set:Ne
11771       \l_tmpb_tl
11772     {
11773       \tl_tail:V
11774         \l_tmpa_tl
11775     }
11776     \tl_reverse:N
11777       \l_tmpb_tl

```

Next, we munge the theme name.

```

11778     \str_set:NV
11779       \l_tmpa_str
11780       \l_tmpb_tl

```

```

11781 \str_replace_all:Nnn
11782 \l_tmpa_str
11783 { / }
11784 { _ }

```

Finally, we load the plain TeX theme.

```

11785 \@_plain_tex_load_theme:VV
11786 \l_tmpb_tl
11787 \l_tmpa_str
11788 }
11789 \cs_generate_variant:Nn
11790 \tl_set:Nn
11791 { Ne }
11792 \cs_generate_variant:Nn
11793 \@_plain_tex_load_theme:nn
11794 { VV }
11795 \ExplSyntaxOff

```

The [witiko/tilde](#) theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```

11796 \markdownSetup {
11797   rendererPrototypes = {
11798     tilde = {~},
11799   },
11800 }

```

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.

```

11801 \def\markdownRendererInterblockSeparatorPrototype{\par}%
11802 \def\markdownRendererParagraphSeparatorPrototype{%
11803   \markdownRendererInterblockSeparator}%
11804 \def\markdownRendererHardLineBreakPrototype{\hfil\break}%
11805 \def\markdownRendererSoftLineBreakPrototype{ }%
11806 \let\markdownRendererEllipsisPrototype\dots
11807 \def\markdownRendererNbspPrototype{~}%
11808 \def\markdownRendererLeftBracePrototype{\char`\\}%
11809 \def\markdownRendererRightBracePrototype{\char`\\}%
11810 \def\markdownRendererDollarSignPrototype{\char`$}%
11811 \def\markdownRendererPercentSignPrototype{\char`\}%
11812 \def\markdownRendererAmpersandPrototype{\&%
11813 \def\markdownRendererUnderscorePrototype{\char`_%
11814 \def\markdownRendererHashPrototype{\char`#}%
11815 \def\markdownRendererCircumflexPrototype{\char`^}%
11816 \def\markdownRendererBackslashPrototype{\char`\}%

```

```

11817 \def\markdownRendererTildePrototype{\char`~}%
11818 \def\markdownRendererPipePrototype{|}%
11819 \def\markdownRendererCodeSpanPrototype#1{\tt#1}%
11820 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
11821 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
11822   \markdownInput{#3}}%
11823 \def\markdownRendererContentBlockOnlineImagePrototype{%
11824   \markdownRendererImage}%
11825 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
11826   \markdownRendererInputFencedCode{#3}{#2}{#2}}%
11827 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
11828 \def\markdownRendererUlBeginPrototype{}%
11829 \def\markdownRendererUlBeginTightPrototype{}%
11830 \def\markdownRendererUlItemPrototype{}%
11831 \def\markdownRendererUlItemEndPrototype{}%
11832 \def\markdownRendererUlEndPrototype{}%
11833 \def\markdownRendererUlEndTightPrototype{}%
11834 \def\markdownRendererOlBeginPrototype{}%
11835 \def\markdownRendererOlBeginTightPrototype{}%
11836 \def\markdownRendererFancyOlBeginPrototype#1#2{\markdownRendererOlBegin}%
11837 \def\markdownRendererFancyOlBeginTightPrototype#1#2{\markdownRendererOlBeginTight}%
11838 \def\markdownRendererOlItemPrototype{}%
11839 \def\markdownRendererOlItemWithNumberPrototype#1{}%
11840 \def\markdownRendererOlItemEndPrototype{}%
11841 \def\markdownRendererFancyOlItemPrototype{\markdownRendererOlItem}%
11842 \def\markdownRendererFancyOlItemWithNumberPrototype{\markdownRendererOlItemWithNumber}%
11843 \def\markdownRendererFancyOlItemEndPrototype{}%
11844 \def\markdownRendererOlEndPrototype{}%
11845 \def\markdownRendererOlEndTightPrototype{}%
11846 \def\markdownRendererFancyOlEndPrototype{\markdownRendererOlEnd}%
11847 \def\markdownRendererFancyOlEndTightPrototype{\markdownRendererOlEndTight}%
11848 \def\markdownRendererDlBeginPrototype{}%
11849 \def\markdownRendererDlBeginTightPrototype{}%
11850 \def\markdownRendererDlItemPrototype#1{#1}%
11851 \def\markdownRendererDlItemEndPrototype{}%
11852 \def\markdownRendererDlDefinitionBeginPrototype{}%
11853 \def\markdownRendererDlDefinitionEndPrototype{\par}%
11854 \def\markdownRendererDlEndPrototype{}%
11855 \def\markdownRendererDlEndTightPrototype{}%
11856 \def\markdownRendererEmphasisPrototype#1{\it#1}%
11857 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
11858 \def\markdownRendererBlockQuoteBeginPrototype{\begingroup\it}%
11859 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
11860 \def\markdownRendererLineBlockBeginPrototype{\begingroup\parindent=0pt}%
11861 \def\markdownRendererLineBlockEndPrototype{\endgroup}%
11862 \def\markdownRendererInputVerbatimPrototype#1{%
11863   \par{\tt\input#1\relax{}}\par}%

```

```

11864 \def\markdownRendererInputFencedCodePrototype#1#2#3{%
11865   \markdownRendererInputVerbatim{#1}}%
11866 \def\markdownRendererHeadingOnePrototype#1{#1}%
11867 \def\markdownRendererHeadingTwoPrototype#1{#1}%
11868 \def\markdownRendererHeadingThreePrototype#1{#1}%
11869 \def\markdownRendererHeadingFourPrototype#1{#1}%
11870 \def\markdownRendererHeadingFivePrototype#1{#1}%
11871 \def\markdownRendererHeadingSixPrototype#1{#1}%
11872 \def\markdownRendererThematicBreakPrototype{}%
11873 \def\markdownRendererNotePrototype#1{#1}%
11874 \def\markdownRendererCitePrototype#1{}%
11875 \def\markdownRendererTextCitePrototype#1{}%
11876 \def\markdownRendererTickedBoxPrototype{[X]}%
11877 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
11878 \def\markdownRendererUntickedBoxPrototype{[ ]}%
11879 \def\markdownRendererStrikeThroughPrototype#1{#1}%
11880 \def\markdownRendererSuperscriptPrototype#1{#1}%
11881 \def\markdownRendererSubscriptPrototype#1{#1}%
11882 \def\markdownRendererDisplayMathPrototype#1{$$#1$$}%
11883 \def\markdownRendererInlineMathPrototype#1{$#1$}%
11884 \ExplSyntaxOn
11885 \cs_gset:Npn
11886   \markdownRendererHeaderAttributeContextBeginPrototype
11887   {
11888     \group_begin:
11889     \color_group_begin:
11890   }
11891 \cs_gset:Npn
11892   \markdownRendererHeaderAttributeContextEndPrototype
11893   {
11894     \color_group_end:
11895     \group_end:
11896   }
11897 \cs_gset_eq:NN
11898   \markdownRendererBracketedSpanAttributeContextBeginPrototype
11899   \markdownRendererHeaderAttributeContextBeginPrototype
11900 \cs_gset_eq:NN
11901   \markdownRendererBracketedSpanAttributeContextEndPrototype
11902   \markdownRendererHeaderAttributeContextEndPrototype
11903 \cs_gset_eq:NN
11904   \markdownRendererFencedDivAttributeContextBeginPrototype
11905   \markdownRendererHeaderAttributeContextBeginPrototype
11906 \cs_gset_eq:NN
11907   \markdownRendererFencedDivAttributeContextEndPrototype
11908   \markdownRendererHeaderAttributeContextEndPrototype
11909 \cs_gset_eq:NN
11910   \markdownRendererFencedCodeAttributeContextBeginPrototype

```



```

11911 \markdownRendererHeaderAttributeContextBeginPrototype
11912 \cs_gset_eq:NN
11913 \markdownRendererFencedCodeAttributeContextEndPrototype
11914 \markdownRendererHeaderAttributeContextEndPrototype
11915 \cs_gset:Npn
11916 \markdownRendererReplacementCharacterPrototype
11917 { \codepoint_str_generate:n { fffd } }
11918 \ExplSyntaxOff
11919 \def\markdownRendererSectionBeginPrototype{}%
11920 \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.

```

11921 \ExplSyntaxOn
11922 \cs_new:Nn
11923 \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
11924 {
11925   \str_case:nn
11926     { #2 }
11927     {
11928       { md } { \markdownInput{#1} }
11929       { tex } { \markdownEscape{#1} \unskip }
11930     }
11931 }
11932 \cs_new:Nn
11933 \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
11934 {
11935   \str_case:nn
11936     { #2 }
11937     {
11938       { md } { \markdownInput{#1} }
11939       { tex } { \markdownEscape{#1} }
11940     }
11941 }
11942 \cs_gset:Npn
11943 \markdownRendererInputRawInlinePrototype#1#2
11944 {
11945   \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
11946     { #1 }
11947     { #2 }
11948 }
11949 \cs_gset:Npn
11950 \markdownRendererInputRawBlockPrototype#1#2
11951 {

```

```

11952     \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
11953         { #1 }
11954         { #2 }
11955     }
11956 \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  $p$ .

```

11957 \ExplSyntaxOn
11958 \seq_new:N \g_@@_jekyll_data_datatypes_seq
11959 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
11960 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
11961 \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.

```

11962 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
11963 \cs_new:Nn \markdown_jekyll_data_push_address_segment:n
11964 {
11965     \seq_if_empty:NF
11966         \g_@@_jekyll_data_datatypes_seq
11967         {
11968             \seq_get_right:NN
11969             \g_@@_jekyll_data_datatypes_seq
11970             \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.

```

11971     \str_if_eq:NNTF

```

```

11972     \l_tmpa_tl
11973     \c_@@_jekyll_data_sequence_tl
11974     {
11975         \seq_put_right:Nn
11976         \g_@@_jekyll_data_wildcard_absolute_address_seq
11977         { * }
11978     }
11979     {
11980         \seq_put_right:Nn
11981         \g_@@_jekyll_data_wildcard_absolute_address_seq
11982         { #1 }
11983     }
11984 }
11985 }

```

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.

```

11986 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
11987 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
11988 \cs_new:Nn \markdown_jekyll_data_concatenate_address:NN
11989 {
11990     \seq_pop_left:NN #1 \l_tmpa_tl

```

```

11991     \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
11992     \seq_put_left:NV #1 \l_tmpa_tl
11993   }
11994 \cs_new:Nn \markdown_jekyll_data_update_address_tls:
11995   {
11996     \markdown_jekyll_data_concatenate_address:NN
11997     \g_@@_jekyll_data_wildcard_absolute_address_seq
11998     \g_@@_jekyll_data_wildcard_absolute_address_tl
11999     \seq_get_right:NN
12000     \g_@@_jekyll_data_wildcard_absolute_address_seq
12001     \g_@@_jekyll_data_wildcard_relative_address_tl
12002   }

```

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.

```

12003 \cs_new:Nn \markdown_jekyll_data_push:nN
12004   {
12005     \markdown_jekyll_data_push_address_segment:n
12006     { #1 }
12007     \seq_put_right:NV
12008     \g_@@_jekyll_data_datatypes_seq
12009     #2
12010     \markdown_jekyll_data_update_address_tls:
12011   }
12012 \cs_new:Nn \markdown_jekyll_data_pop:
12013   {
12014     \seq_pop_right:NN
12015     \g_@@_jekyll_data_wildcard_absolute_address_seq
12016     \l_tmpa_tl
12017     \seq_pop_right:NN
12018     \g_@@_jekyll_data_datatypes_seq
12019     \l_tmpa_tl
12020     \markdown_jekyll_data_update_address_tls:
12021   }

```

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.

```

12022 \cs_new:Nn \markdown_jekyll_data_set_keyval:nn
12023   {
12024     \keys_set_known:nn
12025     { markdown/jekyllData }
12026     { { #1 } = { #2 } }
12027   }
12028 \cs_generate_variant:Nn
12029   \markdown_jekyll_data_set_keyval:nn
12030   { Vn }
12031 \cs_new:Nn \markdown_jekyll_data_set_keyvals:nn

```

```

12032 {
12033     \markdown_jekyll_data_push:nN
12034     { #1 }
12035     \c_@@_jekyll_data_scalar_tl
12036     \markdown_jekyll_data_set_keyval:Vn
12037     \g_@@_jekyll_data_wildcard_absolute_address_tl
12038     { #2 }
12039     \markdown_jekyll_data_set_keyval:Vn
12040     \g_@@_jekyll_data_wildcard_relative_address_tl
12041     { #2 }
12042     \markdown_jekyll_data_pop:
12043 }

```

Finally, we will register our macros as token renderer prototypes to be able to react to the traversal of a YAML document.

```

12044 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
12045     \markdown_jekyll_data_push:nN
12046     { #1 }
12047     \c_@@_jekyll_data_sequence_tl
12048 }
12049 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
12050     \markdown_jekyll_data_push:nN
12051     { #1 }
12052     \c_@@_jekyll_data_mapping_tl
12053 }
12054 \def\markdownRendererJekyllDataSequenceEndPrototype{
12055     \markdown_jekyll_data_pop:
12056 }
12057 \def\markdownRendererJekyllDataMappingEndPrototype{
12058     \markdown_jekyll_data_pop:
12059 }
12060 \def\markdownRendererJekyllDataBooleanPrototype#1#2{
12061     \markdown_jekyll_data_set_keyvals:nn
12062     { #1 }
12063     { #2 }
12064 }
12065 \def\markdownRendererJekyllDataEmptyPrototype#1{}
12066 \def\markdownRendererJekyllDataNumberPrototype#1#2{
12067     \markdown_jekyll_data_set_keyvals:nn
12068     { #1 }
12069     { #2 }
12070 }
12071 \def\markdownRendererJekyllDataStringPrototype#1#2{
12072     \markdown_jekyll_data_set_keyvals:nn
12073     { #1 }
12074     { #2 }
12075 }

```

```
12076 \ExplSyntaxOff
```

If plain  $\TeX$  is the top layer, we load the `witiko/markdown/defaults` plain  $\TeX$  theme with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```
12077 \ExplSyntaxOn
12078 \str_if_eq:VVT
12079   \c_@@_top_layer_tl
12080   \c_@@_option_layer_plain_tex_tl
12081   {
12082     \ExplSyntaxOff
12083     \@@_if_option:nF
12084       { noDefaults }
12085     {
12086       \@@_setup:n
12087         {theme = witiko/markdown/defaults}
12088     }
12089   \ExplSyntaxOn
12090 }
12091 \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).

```
12092 \ExplSyntaxOn
12093 \tl_new:N \g_@@_formatted_lua_options_tl
12094 \cs_new:Nn \@@_format_lua_options:
12095   {
12096     \tl_gclear:N
12097       \g_@@_formatted_lua_options_tl
12098     \seq_map_function:NN
12099       \g_@@_lua_options_seq
12100       \@@_format_lua_option:n
12101   }
12102 \cs_new:Nn \@@_format_lua_option:n
12103   {
12104     \@@_typecheck_option:n
12105       { #1 }
12106     \@@_get_option_type:nN
12107       { #1 }
12108     \l_tmpa_tl
12109     \bool_case_true:nF
12110       {
12111         {
12112           \str_if_eq_p:VV
```

```

12113         \l_tmpa_tl
12114         \c_@@_option_type_boolean_tl ||
12115         \str_if_eq_p:VV
12116         \l_tmpa_tl
12117         \c_@@_option_type_number_tl ||
12118         \str_if_eq_p:VV
12119         \l_tmpa_tl
12120         \c_@@_option_type_counter_tl
12121     }
12122     {
12123         \@@_get_option_value:nN
12124         { #1 }
12125         \l_tmpa_tl
12126         \tl_gput_right:Nx
12127         \g_@@_formatted_lua_options_tl
12128         { #1~::~ \l_tmpa_tl ,~ }
12129     }
12130     {
12131         \str_if_eq_p:VV
12132         \l_tmpa_tl
12133         \c_@@_option_type_clist_tl
12134     }
12135     {
12136         \@@_get_option_value:nN
12137         { #1 }
12138         \l_tmpa_tl
12139         \tl_gput_right:Nx
12140         \g_@@_formatted_lua_options_tl
12141         { #1~::~\c_left_brace_str }
12142         \clist_map_inline:Vn
12143         \l_tmpa_tl
12144         {
12145             \tl_gput_right:Nx
12146             \g_@@_formatted_lua_options_tl
12147             { "##1" ,~ }
12148         }
12149         \tl_gput_right:Nx
12150         \g_@@_formatted_lua_options_tl
12151         { \c_right_brace_str ,~ }
12152     }
12153 }
12154 {
12155     \@@_get_option_value:nN
12156     { #1 }
12157     \l_tmpa_tl
12158     \tl_gput_right:Nx
12159     \g_@@_formatted_lua_options_tl

```

```

12160         { #1~::~ " \l_tmpa_tl " ,~ }
12161     }
12162 }
12163 \cs_generate_variant:Nn
12164   \clist_map_inline:nn
12165   { Vn }
12166 \let\markdownPrepareLuaOptions=\@@_format_lua_options:
12167 \def\markdownLuaOptions{{ \g_@@_formatted_lua_options_tl }}
12168 \ExplSyntaxOff

```

The `\markdownPrepare` macro contains the Lua code that is executed prior to any conversion from markdown to plain TeX. It exposes the `convert` function for the use by any further Lua code.

```

12169 \def\markdownPrepare{%
First, ensure that the cacheDir directory exists.
12170   local lfs = require("lfs")
12171   local cacheDir = "\markdownOptionCacheDir"
12172   if not lfs.isdir(cacheDir) then
12173     assert(lfs.mkdir(cacheDir))
12174   end

```

Next, load the `markdown` module and create a converter function using the plain TeX options, which were serialized to a Lua table via the `\markdownLuaOptions` macro.

```

12175   local md = require("markdown")
12176   local convert = md.new(\markdownLuaOptions)
12177 }%

```

The `\markdownCleanup` macro contains the Lua code that is executed after any conversion from markdown to plain TeX.

```

12178 \def\markdownCleanup{%
Remove the options.cacheDir directory if it is empty.
12179   lfs.rmdir(cacheDir)
12180 }%

```

### 3.2.5 Buffering Block-Level Markdown Input

The macros `\markdownInputFileStream` and `\markdownOutputFileStream` contain the number of the input and output file streams that will be used for the IO operations of the package.

```

12181 \csname newread\endcsname\markdownInputFileStream
12182 \csname newwrite\endcsname\markdownOutputFileStream

```

The `\markdownReadAndConvertTab` macro contains the tab character literal.

```

12183 \begingroup
12184   \catcode\^^I=12%
12185   \gdef\markdownReadAndConvertTab{^^I}%
12186 \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.

```
12187 \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.

```
12188 \catcode\^^M=13%
12189 \catcode\^^I=13%
12190 \catcode|=0%
12191 \catcode\=12%
12192 |catcode%=14%
12193 |catcode%=12&
12194 |gdef|markdownReadAndConvert#1#2{&
12195 |begingroup&
```

If we are not reading markdown documents from the frozen cache, open the `inputTempFileName` file for writing.

```
12196 |markdownIfOption{frozenCache}{}{&
12197 |immediate|openout|markdownOutputFileStream&
12198 |markdownOptionInputTempFileName|relax&
12199 |markdownInfo{&
12200 Buffering block-level markdown input into the temporary &
12201 input file "|markdownOptionInputTempFileName" and scanning &
12202 for the closing token sequence "#1"}&
12203 }&
```

Locally change the category of the special plain T<sub>E</sub>X characters to *other* in order to prevent unwanted interpretation of the input. Change also the category of the space character, so that we can retrieve it unaltered.

```
12204 |def|do##1{|catcode`##1=12}|dospecials&
12205 |catcode`|=12&
12206 |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.

```
12207 |def|markdownReadAndConvertStripPercentSign##1{&
12208 |markdownIfOption{stripPercentSigns}{&
12209 |if##1%&
12210 |expandafter|expandafter|expandafter&
12211 |markdownReadAndConvertProcessLine&
12212 |else&
12213 |expandafter|expandafter|expandafter&
```

```

12214         |markdownReadAndConvertProcessLine@
12215         |expandafter|expandafter|expandafter##1@
12216     |fi@
12217 }{@
12218     |expandafter@
12219         |markdownReadAndConvertProcessLine@
12220         |expandafter##1@
12221 }@
12222 }@

```

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.

```

12223 |def|markdownReadAndConvertProcessLine##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.

```

12224     |ifx|relax##3|relax@
12225         |markdownIfOption{frozenCache}{}{@
12226         |immediate|write|markdownOutputFileStream{##1}@
12227     }@
12228     |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.

```

12229     |def^^M{@
12230         |markdownInfo{The ending token sequence was found}@
12231         |markdownIfOption{frozenCache}{}{@
12232         |immediate|closeout|markdownOutputFileStream@
12233     }@
12234     |endgroup@
12235     |markdownInput{@
12236         |markdownOptionOutputDir@
12237         /|markdownOptionInputTempFileName@
12238     }@
12239     #2}@
12240 |fi@

```

Repeat with the next line.

```

12241     ^^M}@

```

Make the tab character active at expansion time and make it expand to a literal tab character.

```

12242     |catcode`|^I=13@

```

```
12243 |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.

```
12244 |catcode`\^^M=13@
```

```
12245 |def^^M##1^^M{@
```

```
12246 |def^^M###1^^M{@
```

```
12247 |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
```

```
12248 ^^M}@
```

```
12249 ^^M}@
```

Reset the character categories back to the former state.

```
12250 |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.

```
12251 \ExplSyntaxOn
```

```
12252 \cs_new:Npn
```

```
12253 \markdownLuaExecute
```

```
12254 #1
```

```
12255 {
```

```
12256 \int_compare:nNt
```

```
12257 { \g_luabridge_method_int }
```

```
12258 =
```

```
12259 { \c_luabridge_method_shell_int }
```

```
12260 {
```

```
12261 \sys_if_shell_unrestricted:F
```

```
12262 {
```

```
12263 \sys_if_shell:TF
```

```
12264 {
```

```
12265 \msg_error:nn
```

```
12266 { markdown }
```

```
12267 { restricted-shell-access }
```

```
12268 }
```

```
12269 {
```

```
12270 \msg_error:nn
```

```
12271 { markdown }
```

```
12272 { disabled-shell-access }
```

```
12273 }
```

```
12274 }
```

```
12275 }
```

```
12276 \luabridge_now:e
```

```
12277 { #1 }
```

```
12278 }
```

```
12279 \cs_generate_variant:Nn
```

```
12280 \msg_new:nnnn
```

```
12281 { nnnV }
```

```

12282 \tl_set:Nn
12283   \l_tmpa_tl
12284   {
12285     You-may-need-to-run-TeX-with-the---shell-escape-or-the-
12286     --enable-write18-flag,~or~write-shell_escape=t-in-the-
12287     texmf.cnf-file.
12288   }
12289 \msg_new:nnnV
12290   { markdown }
12291   { restricted-shell-access }
12292   { Shell-escape-is-restricted }
12293   \l_tmpa_tl
12294 \msg_new:nnnV
12295   { markdown }
12296   { disabled-shell-access }
12297   { Shell-escape-is-disabled }
12298   \l_tmpa_tl
12299 \ExplSyntaxOff

```

### 3.2.6 Buffering Inline Markdown Input

This section describes the implementation of the macro `\markinline`.

```

12300 \ExplSyntaxOn
12301 \tl_new:N
12302   \g_@@_after_markinline_tl
12303 \tl_gset:Nn
12304   \g_@@_after_markinline_tl
12305   { \unskip }
12306 \cs_new:Npn
12307   \markinline
12308   {

```

Locally change the category of the special plain  $\TeX$  characters to *other* in order to prevent unwanted interpretation of the input markdown text as  $\TeX$  code.

```

12309   \group_begin:
12310   \cctab_select:N
12311     \c_other_cctab

```

Unless we are reading markdown documents from the frozen cache, open the file `inputTempFileName` for writing.

```

12312   \@@_if_option:nF
12313     { frozenCache }
12314     {
12315       \immediate
12316         \openout
12317           \markdownOutputFileStream
12318           \markdownOptionInputTempFileName

```

```

12319         \relax
12320     \msg_info:nne
12321         { markdown }
12322         { buffering-markinline }
12323         { \markdownOptionInputTempFileName }
12324     }

```

Peek ahead and extract the inline markdown text.

```

12325     \peek_regex_replace_once:nnF
12326     { { (.*) } }
12327     {

```

Unless we are reading markdown documents from the frozen cache, store the text in the file `inputTempFileName` and close it.

```

12328         \c { @@_if_option:nF }
12329         \cB { frozenCache \cE }
12330         \cB {
12331             \c { immediate }
12332             \c { write }
12333             \c { markdownOutputFileStream }
12334             \cB { \1 \cE }
12335             \c { immediate }
12336             \c { closeout }
12337             \c { markdownOutputFileStream }
12338         \cE }

```

Reset the category codes and `\input` the result of the conversion.

```

12339     \c { group_end: }
12340     \c { group_begin: }
12341     \c { @@_setup:n }
12342     \cB { contentLevel = inline \cE }
12343     \c { markdownInput }
12344     \cB {
12345         \c { markdownOptionOutputDir } /
12346         \c { markdownOptionInputTempFileName }
12347     \cE }
12348     \c { group_end: }
12349     \c { tl_use:N }
12350     \c { g_@@_after_markinline_tl }
12351 }
12352 {
12353     \msg_error:nn
12354     { markdown }
12355     { markinline-peek-failure }
12356     \group_end:
12357     \tl_use:N
12358     \g_@@_after_markinline_tl
12359 }

```

```

12360 }
12361 \msg_new:nnn
12362 { markdown }
12363 { buffering-markinline }
12364 { Buffering~inline~markdown~input~into~the~temporary~input~file~"#1". }
12365 \msg_new:nnnn
12366 { markdown }
12367 { markinline-peek-failure }
12368 { Use-of~\iow_char:N \\ markinline~doesn't~match~its~definition }
12369 { The~macro~should~be~followed~by~inline~markdown~text~in~curly~braces }
12370 \ExplSyntaxOff

```

### 3.2.7 Typesetting Markdown

The `\markdownInput` macro uses an implementation of the `\markdownLuaExecute` macro to convert the contents of the file whose filename it has received as its single argument from markdown to plain T<sub>E</sub>X.

```

12371 \ExplSyntaxOn
12372 \cs_new:Npn
12373   \markdownInput
12374   #1
12375   {

```

If the file does not exist in the current directory, we will search for it in the directories specified in `\l_file_search_path_seq`. On L<sup>A</sup>T<sub>E</sub>X, this also includes the directories specified in `\input@path`.

```

12376   \file_get_full_name:nNTF
12377   { #1 }
12378   \l_tmpa_tl
12379   {
12380     \exp_args:NV
12381     \markdownInputRaw
12382     \l_tmpa_tl
12383   }
12384   {
12385     \msg_error:nnnV
12386     { markdown }
12387     { markdown-file-does-not-exist }
12388     { #1 }
12389   }
12390 }
12391 \msg_new:nnn
12392 { markdown }
12393 { markdown-file-does-not-exist }
12394 {
12395   Markdown~file~#1~does~not~exist
12396 }

```

```
12397 \ExplSyntaxOff
12398 \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.

```
12399 \catcode`\|=0%
12400 \catcode`\|=12%
12401 \catcode`\&=6%
12402 |gdef|markdownInputRaw#1{%
```

Change the category code of the percent sign (%) to other, so that a user of the [hybrid](#) Lua option or a malevolent actor can't produce TeX comments in the plain TeX output of the Markdown package.

```
12403 |begingroup
12404 |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.

```
12405 |catcode`|#=12
```

If we are reading from the frozen cache, input it, expand the corresponding `\markdownFrozenCache<number>` macro, and increment `frozenCacheCounter`.

```
12406 |markdownIfOption{frozenCache}{%
12407 |ifnum|markdownOptionFrozenCacheCounter=0|relax
12408 |markdownInfo{Reading frozen cache from
12409 |"|markdownOptionFrozenCacheFileName"%}
12410 |input|markdownOptionFrozenCacheFileName|relax
12411 |fi
12412 |markdownInfo{Including markdown document number
12413 |"|the|markdownOptionFrozenCacheCounter" from frozen cache}%
12414 |csname markdownFrozenCache|the|markdownOptionFrozenCacheCounter|endcsname
12415 |global|advance|markdownOptionFrozenCacheCounter by 1|relax
12416 }{%
12417 |markdownInfo{Including markdown document "&1"}%
```

Attempt to open the markdown document to record it in the `.log` and `.fls` files. This allows external programs such as `LATEX Mk` to track changes to the markdown document.

```
12418 |openin|markdownInputFileStream&1
12419 |closein|markdownInputFileStream
12420 |markdownPrepareLuaOptions
12421 |markdownLuaExecute{%
12422 |markdownPrepare
12423 local file = assert(io.open("&1", "r"),
12424 [[Could not open file "&1" for reading]])
12425 local input = assert(file:read("*a"))
```

```

12426     assert(file:close())
12427     print(convert(input))
12428     |markdownCleanup}%

```

If we are finalizing the frozen cache, increment `frozenCacheCounter`.

```

12429     |markdownIfOption{finalizeCache}{}%
12430     |global|advance|markdownOptionFrozenCacheCounter by 1|relax}{}%
12431     }%
12432     |endgroup
12433     }%
12434 |endgroup

```

The `\markdownEscape` macro resets the category codes of the percent sign and the hash sign back to comment and parameter, respectively, before using the `\input` built-in of  $\TeX$  to execute a  $\TeX$  document in the middle of a markdown document fragment.

```

12435 \gdef\markdownEscape#1{%
12436   \catcode`\%=14\relax
12437   \catcode`\#=6\relax
12438   \input #1\relax
12439   \catcode`\%=12\relax
12440   \catcode`\#=12\relax
12441 }%

```

### 3.3 $\LaTeX$ Implementation

The  $\LaTeX$  implementation makes use of the fact that, apart from some subtle differences,  $\LaTeX$  implements the majority of the plain  $\TeX$  format [12, Section 9]. As a consequence, we can directly reuse the existing plain  $\TeX$  implementation.

```

12442 \def\markdownVersionSpace{ }%
12443 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
12444   \markdownVersion\markdownVersionSpace markdown renderer]%

```

#### 3.3.1 Typesetting Markdown

The `\markinlinePlainTeX` macro is used to store the original plain  $\TeX$  implementation of the `\markinline` macro. The `\markinline` macro is then redefined to accept an optional argument with options recognized by the  $\LaTeX$  interface (see Section 2.3.2).

```

12445 \ExplSyntaxOn
12446 \cs_gset_eq:NN
12447   \markinlinePlainTeX
12448   \markinline
12449 \cs_gset:Npn
12450   \markinline
12451   {

```



```

12452 \peek_regex_replace_once:nn
12453 { ( \[ (.*) \] ) ? }
12454 {
    Apply the options locally.
12455 \c { group_begin: }
12456 \c { @@_setup:n }
12457 \cB { \2 \cE }
12458 \c { tl_put_right:Nn }
12459 \c { g_@@_after_markinline_tl }
12460 \cB { \c { group_end: } \cE }
12461 \c { markinlinePlainTeX }
12462 }
12463 }
12464 \ExplSyntaxOff

```

The `\markdownInputPlainTeX` macro is used to store the original plain `TeX` implementation of the `\markdownInput` macro. The `\markdownInput` macro is then redefined to accept an optional argument with options recognized by the `LATEX` interface (see Section 2.3.2).

```

12465 \let\markdownInputPlainTeX\markdownInput
12466 \renewcommand\markdownInput [2] [] {%
12467 \begingroup
12468 \markdownSetup{#1}%
12469 \markdownInputPlainTeX{#2}%
12470 \endgroup}%

```

The `markdown`, and `markdown*` `LATEX` environments are implemented using the `\markdownReadAndConvert` macro.

```

12471 \ExplSyntaxOn
12472 \renewenvironment
12473 { markdown }
12474 {

```

In our implementation of the `markdown` `LATEX` environment, we want to distinguish between the following two cases:

<code>\begin{markdown} [smartEllipses]</code>	<code>\begin{markdown}</code>
<code>% This is an optional argument ^</code>	<code>[smartEllipses]</code>
<code>% ...</code>	<code>% ^ This is link</code>
<code>\end{markdown}</code>	<code>\end{markdown}</code>

Therefore, we cannot use the built-in `LATEX` support for environments with optional arguments or packages such as `xparse`. Instead, we must read the optional argument manually and prevent reading past the end of a line.

To prevent reading past the end of a line when looking for the optional argument of the `markdown` `LATEX` environment and accidentally tokenizing markdown text, we

change the category code of carriage return (`\r`, ASCII character 13 in decimal) from 5 (end of line).

While any category code other than 5 (end of line) would work, we switch to the category 13 (active), which is also used by the `\markdownReadAndConvert` macro. This is necessary if we read until the end of a line, because then the carriage return character will be produced by T<sub>E</sub>X via the `\endlinechar` plain T<sub>E</sub>X macro and it needs to have the correct category code, so that `\markdownReadAndConvert` processes it correctly.

```
12475     \group_begin:
12476     \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).

```
12477     \char_set_catcode_letter:n { 35 }
```

After we have matched the opening `[` that begins the optional argument, we accept carriage returns as well.

```
12478     \peek_regex_replace_once:nnF
12479     { \ *[\r*([^\r]*)\][^\r]* }
12480     {
```

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.

```
12481         \c { group_end: }
12482         \c { t1_set_rescan:Nnn } \c { l_tmpa_t1 } { } { \1 }
```

Then, we pass the retokenized content to the `\markdownSetup` macro.

```
12483         \c { @@_setup:V } \c { l_tmpa_t1 }
```

Finally, regardless of whether or not we have matched the optional argument, we let the `\markdownReadAndConvert` macro process the rest of the L<sup>A</sup>T<sub>E</sub>X environment.

```
12484         \c { markdownReadAndConvert@markdown } { }
12485     }
12486     {
12487         \group_end:
12488         \markdownReadAndConvert@markdown { }
12489     }
12490 }
12491 { \markdownEnd }
12492 \renewenvironment
12493 { markdown* }
12494 [ 1 ]
12495 {
```

```

12496   \msg_warning:nnn
12497     { markdown }
12498     { latex-markdown-star-deprecated }
12499     { #1 }
12500   \@@_setup:n
12501     { #1 }
12502   \markdownReadAndConvert@markdown *
12503   }
12504   { \markdownEnd }
12505 \msg_new:nnn
12506   { markdown }
12507   { latex-markdown-star-deprecated }
12508   {
12509     The~markdown*~LaTeX~environment~has~been~deprecated~and~will~
12510     be~removed~in~the~next~major~version~of~the~Markdown~package.
12511   }
12512 \ExplSyntaxOff
12513 \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*.

```

12514   \catcode`\|=0\catcode`\<=1\catcode`\>=2%
12515   \catcode`\|=12\catcode`\{=12\catcode`\}=12%
12516   |gdef|markdownReadAndConvert@markdown#1<%
12517     |markdownReadAndConvert<\end{markdown#1}>%
12518     <|end<markdown#1>>>%
12519 |endgroup

```

### 3.3.2 Options

The supplied package options are processed using the `\markdownSetup` macro.

```

12520 \DeclareOption*{%
12521   \expandafter\markdownSetup\expandafter{\CurrentOption}}%
12522 \ProcessOptions\relax

```

### 3.3.3 Themes

This section overrides the plain  $\TeX$  implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in  $\LaTeX$  themes provided with the Markdown package.

```

12523 \ExplSyntaxOn
12524 \cs_gset:Nn
12525   \@@_load_theme:nn
12526   {

```

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.

```
12527 \ifmarkdownLaTeXLoaded
12528 \ifx\@onlypreamble\@notprerr
```

If both conditions are true does, end with an error, since we cannot load L<sup>A</sup>T<sub>E</sub>X themes after the preamble. Otherwise, try loading a plain T<sub>E</sub>X theme instead.

```
12529 \file_if_exist:nTF
12530 { markdown theme #2.sty }
12531 {
12532 \msg_error:nnn
12533 { markdown }
12534 { latex-theme-after-preamble }
12535 { #1 }
12536 }
12537 {
12538 \@@_plain_tex_load_theme:nn
12539 { #1 }
12540 { #2 }
12541 }
12542 \else
```

If the Markdown package has already been loaded but we are still in the preamble, load a L<sup>A</sup>T<sub>E</sub>X theme if it exists or load a plain T<sub>E</sub>X theme otherwise.

```
12543 \file_if_exist:nTF
12544 { markdown theme #2.sty }
12545 {
12546 \msg_info:nnn
12547 { markdown }
12548 { loading-latex-theme }
12549 { #1 }
12550 \RequirePackage
12551 { markdown theme #2 }
12552 }
12553 {
12554 \@@_plain_tex_load_theme:nn
12555 { #1 }
12556 { #2 }
12557 }
12558 \fi
12559 \else
```

If the Markdown package has not yet been loaded, postpone the loading until the Markdown package has finished loading.

```
12560 \msg_info:nnn
12561 { markdown }
```

```

12562     { theme-loading-postponed }
12563     { #1 }
12564     \AtEndOfPackage
12565     {
12566         \@_load_theme:nn
12567         { #1 }
12568         { #2 }
12569     }
12570     \fi
12571 }
12572 \msg_new:nnn
12573 { markdown }
12574 { theme-loading-postponed }
12575 {
12576     Postponing~loading~Markdown~theme~#1~until~
12577     Markdown~package~has~finished~loading
12578 }
12579 \msg_new:nnn
12580 { markdown }
12581 { loading-latex-theme }
12582 { Loading~LaTeX~Markdown~theme~#1 }
12583 \cs_generate_variant:Nn
12584 \msg_new:nnnn
12585 { nnVV }
12586 \tl_set:Nn
12587 \l_tmpa_tl
12588 { Cannot~load~LaTeX~Markdown~theme~#1~after~ }
12589 \tl_put_right:NV
12590 \l_tmpa_tl
12591 \c_backslash_str
12592 \tl_put_right:Nn
12593 \l_tmpa_tl
12594 { begin{document} }
12595 \tl_set:Nn
12596 \l_tmpb_tl
12597 { Load~Markdown~theme~#1~before~ }
12598 \tl_put_right:NV
12599 \l_tmpb_tl
12600 \c_backslash_str
12601 \tl_put_right:Nn
12602 \l_tmpb_tl
12603 { begin{document} }
12604 \msg_new:nnVV
12605 { markdown }
12606 { latex-theme-after-preamble }
12607 \l_tmpa_tl
12608 \l_tmpb_tl

```

12609 `\ExplSyntaxOff`

The `witiko/dot` theme enables the `fencedCode` Lua option:

12610 `\markdownSetup{fencedCode}%`

We load the `ifthen` and `grffile` packages, see also Section 1.1.3:

12611 `\RequirePackage{ifthen,grffile}`

We store the previous definition of the fenced code token renderer prototype:

12612 `\let\markdown@witiko@dot@oldRendererInputFencedCodePrototype`

12613 `\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:

12614 `\renewcommand\markdownRendererInputFencedCodePrototype[3]{%`

12615 `\def\next##1 ##2\relax{%`

12616 `\ifthenelse{equal{##1}{dot}}{%`

12617 `\markdownIfOption{frozenCache}{}{%`

12618 `\immediate\write18{%`

12619 `if ! test -e #1.pdf.source || ! diff #1 #1.pdf.source;`

12620 `then`

12621 `dot -Tpdf -o #1.pdf #1;`

12622 `cp #1 #1.pdf.source;`

12623 `fi}}%`

We include the typeset image using the image token renderer:

12624 `\markdownRendererImage{Graphviz image}{#1.pdf}{#1.pdf}{##2}%`

If the infostring does not start with `dot ...`, we use the previous definition of the fenced code token renderer prototype:

12625 `}{%`

12626 `\markdown@witiko@dot@oldRendererInputFencedCodePrototype{#1}{#2}{#3}%`

12627 `}%`

12628 `}%`

12629 `\next#2 \relax}%`

The `witiko/graphicx/http` theme stores the previous definition of the image token renderer prototype:

12630 `\let\markdown@witiko@graphicx@http@oldRendererImagePrototype`

12631 `\markdownRendererImagePrototype`

We load the `catchfile` and `grffile` packages, see also Section 1.1.3:

12632 `\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.

```

12633 \newcount\markdown@witiko@graphicx@http@counter
12634 \markdown@witiko@graphicx@http@counter=0
12635 \newcommand\markdown@witiko@graphicx@http@filename{%
12636   \markdownOptionCacheDir/witiko_graphicx_http%
12637   .\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.

```

12638 \newcommand\markdown@witiko@graphicx@http@download[2]{%
12639   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:

```

12640 \begingroup
12641 \catcode`\%=12
12642 \catcode`\^^A=14

```

We redefine the image token renderer prototype, so that it tries to download an online image.

```

12643 \global\def\markdownRendererImagePrototype#1#2#3#4{^^A
12644   \begingroup
12645   \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:

```

12646   \markdownIfOption{frozenCache}{}{^^A
12647     \immediate\write18{^^A
12648       mkdir -p "\markdownOptionCacheDir";
12649       if printf '%s' "#3" | grep -q -E '^https?:';
12650       then

```

The image will be downloaded to the pathname `cacheDir/⟨the MD5 digest of the image URL⟩.⟨the suffix of the image URL⟩`:

```

12651       OUTPUT_PREFIX="\markdownOptionCacheDir";
12652       OUTPUT_BODY="$ (printf '%s' '#3' | md5sum | cut -d' ' -f1)";
12653       OUTPUT_SUFFIX="$ (printf '%s' '#3' | sed 's/.*[.]//)";
12654       OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";

```

The image will be downloaded only if it has not already been downloaded:

```

12655       if ! [ -e "$OUTPUT" ];
12656       then
12657         \markdown@witiko@graphicx@http@download{'#3'}{"$OUTPUT"};
12658         printf '%s' "$OUTPUT" > "\filename";
12659       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:

```

12660     else
12661         printf '%s' '#3' > "\filename";
12662     fi}}^^A

```

We load the pathname of the downloaded image and we typeset the image using the previous definition of the image renderer prototype:

```

12663     \CatchFileDef{\filename}{\filename}{\endlinechar=-1}^^A
12664     \markdown@witiko@graphicx@http@oldRendererImagePrototype^^A
12665     {#1}{#2}{\filename}{#4}^^A
12666     \endgroup
12667     \global\advance\markdown@witiko@graphicx@http@counter by 1\relax}^^A
12668 \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:

```

12669 \markdownLoadPlainTeXTheme

```

Next, the L<sup>A</sup>T<sub>E</sub>X theme overrides some of the plain T<sub>E</sub>X definitions. See Section 3.3.4 for the actual definitions.

### 3.3.4 Token Renderer Prototypes

The following configuration should be considered placeholder. If the option `plain` has been enabled (see Section 2.2.2.3), none of the definitions will take effect.

```

12670 \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.

```

12671 \@ifclassloaded{beamer}{}{}%
12672 \markdownIfOption{tightLists}{\RequirePackage{paralist}}{}%
12673 \markdownIfOption{fancyLists}{\RequirePackage{paralist}}{}%
12674 }

```

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.

```

12675 \ExplSyntaxOn
12676 \@ifpackageloaded{paralist}{
12677     \tl_new:N
12678     \l_@@_latex_fancy_list_item_label_number_style_tl
12679     \tl_new:N
12680     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
12681     \cs_new:Nn
12682     \@@_latex_fancy_list_item_label_number:nn
12683     {
12684         \str_case:nn
12685         { #1 }

```



```

12686     {
12687     { Decimal } { #2 }
12688     { LowerRoman } { \int_to_roman:n { #2 } }
12689     { UpperRoman } { \int_to_Roman:n { #2 } }
12690     { LowerAlpha } { \int_to_alph:n { #2 } }
12691     { UpperAlpha } { \int_to_Alph:n { #2 } }
12692     }
12693   }
12694   \cs_new:Nn
12695     \@@_latex_fancy_list_item_label_delimiter:n
12696     {
12697     \str_case:nn
12698       { #1 }
12699       {
12700         { Default } { . }
12701         { OneParen } { ) }
12702         { Period } { . }
12703       }
12704     }
12705   \cs_new:Nn
12706     \@@_latex_fancy_list_item_label:nnn
12707     {
12708     \@@_latex_fancy_list_item_label_number:nn
12709       { #1 }
12710       { #3 }
12711     \@@_latex_fancy_list_item_label_delimiter:n
12712       { #2 }
12713     }
12714   \cs_new:Nn
12715     \@@_latex_paralist_style:nn
12716     {
12717     \str_case:nn
12718       { #1 }
12719       {
12720         { Decimal } { 1 }
12721         { LowerRoman } { i }
12722         { UpperRoman } { I }
12723         { LowerAlpha } { a }
12724         { UpperAlpha } { A }
12725       }
12726     \@@_latex_fancy_list_item_label_delimiter:n
12727       { #2 }
12728     }
12729   \markdownSetup{rendererPrototypes={

```

Make tight bullet lists a little less compact by adding extra vertical space above and below them.

```

12730     ulBeginTight = {%
12731         \group_begin:
12732         \pltopsep=\topsep
12733         \plpartopsep=\partopsep
12734         \begin{compactitem}
12735     },
12736     ulEndTight = {
12737         \end{compactitem}
12738         \group_end:
12739     },
12740     fancyOlBegin = {
12741         \group_begin:
12742         \tl_set:Nn
12743             \l_@@_latex_fancy_list_item_label_number_style_tl
12744             { #1 }
12745         \tl_set:Nn
12746             \l_@@_latex_fancy_list_item_label_delimiter_style_tl
12747             { #2 }
12748         \@@_if_option:nTF
12749             { startNumber }
12750             {
12751                 \tl_set:Nn
12752                     \l_tmpa_tl
12753                     { \begin{enumerate} }
12754             }
12755             {
12756                 \tl_set:Nn
12757                     \l_tmpa_tl
12758                     { \begin{enumerate}[ ] }
12759                 \tl_put_right:Nx
12760                     \l_tmpa_tl
12761                     { \@@_latex_paralist_style:nn { #1 } { #2 } }
12762                 \tl_put_right:Nn
12763                     \l_tmpa_tl
12764                     { ] }
12765             }
12766         \tl_use:N
12767             \l_tmpa_tl
12768     },
12769     fancyOlEnd = {
12770         \end{enumerate}
12771         \group_end:
12772     },

```

Make tight ordered lists a little less compact by adding extra vertical space above and below them.

```

12773     olBeginTight = {%

```

```

12774     \group_begin:
12775     \plpartopsep=\partopsep
12776     \pltopsep=\topsep
12777     \begin{compactenum}
12778 },
12779 olEndTight = {
12780     \end{compactenum}
12781     \group_end:
12782 },
12783 fancyOlBeginTight = {
12784     \group_begin:
12785     \tl_set:Nn
12786     \l_@@_latex_fancy_list_item_label_number_style_tl
12787     { #1 }
12788     \tl_set:Nn
12789     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
12790     { #2 }
12791     \@@_if_option:nTF
12792     { startNumber }
12793     {
12794         \tl_set:Nn
12795         \l_tmpa_tl
12796         { \begin{compactenum} }
12797     }
12798     {
12799         \tl_set:Nn
12800         \l_tmpa_tl
12801         { \begin{compactenum}[ ] }
12802         \tl_put_right:Nx
12803         \l_tmpa_tl
12804         { \@@_latex_paralist_style:nn { #1 } { #2 } }
12805         \tl_put_right:Nn
12806         \l_tmpa_tl
12807         { ] }
12808     }
12809     \tl_put_left:Nn
12810     \l_tmpa_tl
12811     {
12812         \plpartopsep=\partopsep
12813         \pltopsep=\topsep
12814     }
12815     \tl_use:N
12816     \l_tmpa_tl
12817 },
12818 fancyOlEndTight = {
12819     \end{compactenum}
12820     \group_end:

```

```

12821     },
12822     fancyO1ItemWithNumber = {
12823         \item
12824         [
12825             \@@_latex_fancy_list_item_label:VVn
12826             \l_@@_latex_fancy_list_item_label_number_style_tl
12827             \l_@@_latex_fancy_list_item_label_delimiter_style_tl
12828             { #1 }
12829         ]
12830     },

```

Make tight definition lists a little less compact by adding extra vertical space above and below them.

```

12831     dlBeginTight = {
12832         \group_begin:
12833         \plpartopsep=\partopsep
12834         \pltopsep=\topsep
12835         \begin{compactdesc}
12836     },
12837     dlEndTight = {
12838         \end{compactdesc}
12839         \group_end:
12840     }}}
12841     \cs_generate_variant:Nn
12842     \@@_latex_fancy_list_item_label:nnn
12843     { VVn }
12844 }{
12845     \markdownSetup{rendererPrototypes={
12846         ulBeginTight = {\markdownRendererUlBegin},
12847         ulEndTight = {\markdownRendererUlEnd},
12848         fancyO1Begin = {\markdownRendererO1Begin},
12849         fancyO1End = {\markdownRendererO1End},
12850         olBeginTight = {\markdownRendererO1Begin},
12851         olEndTight = {\markdownRendererO1End},
12852         fancyO1BeginTight = {\markdownRendererO1Begin},
12853         fancyO1EndTight = {\markdownRendererO1End},
12854         dlBeginTight = {\markdownRendererDlBegin},
12855         dlEndTight = {\markdownRendererDlEnd}}}
12856 }
12857 \ExplSyntaxOff
12858 \RequirePackage{amsmath}

```

Unless the unicode-math package has been loaded, load the amssymb package with symbols to be used for tickboxes.

```

12859 \ifpackageloaded{unicode-math}{
12860     \markdownSetup{rendererPrototypes={
12861         untickedBox = {\$ \mdlgwhtsquare $},
12862     }}

```

```

12863 }{
12864   \RequirePackage{amssymb}
12865   \markdownSetup{rendererPrototypes={
12866     untickedBox = {$\square$},
12867   }}
12868 }
12869 \RequirePackage{csvsimple}
12870 \RequirePackage{fancyvrb}
12871 \RequirePackage{graphicx}
12872 \markdownSetup{rendererPrototypes={
12873   hardLineBreak = {\},
12874   leftBrace = {\textbraceleft},
12875   rightBrace = {\textbraceright},
12876   dollarSign = {\textdollar},
12877   underscore = {\textunderscore},
12878   circumflex = {\textasciicircum},
12879   backslash = {\textbackslash},
12880   tilde = {\textasciitilde},
12881   pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by T<sub>E</sub>X during the typesetting. Therefore, even if we don't know whether a span of text is part of math formula or not when we are parsing markdown,<sup>34</sup> we can reliably detect math mode inside the renderer.

Here, we will redefine the code span renderer prototype to typeset upright text in math formulae and typewriter text outside math formulae.

```

12882   codeSpan = {%
12883     \ifmmode
12884       \text{#1}%
12885     \else
12886       \texttt{#1}%
12887     \fi
12888   }}
12889 \ExplSyntaxOn
12890 \markdownSetup{
12891   rendererPrototypes = {
12892     contentBlock = {
12893       \str_case:nnF
12894         { #1 }
12895         {
12896           { csv }
12897           {
12898             \begin{table}

```

---

<sup>34</sup>This property may actually be undecidable. Suppose a span of text is a part of a macro definition. Then, whether the span of text is part of a math formula or not depends on where the macro is later used, which may easily be *both* inside and outside a math formula.

```

12899         \begin{center}
12900             \csvautotabular{#3}
12901         \end{center}
12902         \tl_if_empty:nF
12903             { #4 }
12904             { \caption{#4} }
12905         \end{table}
12906     }
12907     { tex } { \markdownEscape{#3} }
12908 }
12909 { \markdownInput{#3} }
12910 },
12911 },
12912 }
12913 \ExplSyntaxOff
12914 \markdownSetup{rendererPrototypes={
12915     image = {%
12916         \begin{figure}%
12917             \begin{center}%
12918                 \includegraphics[alt={#1}]{#3}%
12919             \end{center}%
12920             \ifx\empty#4\empty\else
12921                 \caption{#4}%
12922             \fi
12923         \end{figure}},
12924     ulBegin = {\begin{itemize}},
12925     ulEnd = {\end{itemize}},
12926     olBegin = {\begin{enumerate}},
12927     olItem = {\item{}},
12928     olItemWithNumber = {\item[#1.]},
12929     olEnd = {\end{enumerate}},
12930     dlBegin = {\begin{description}},
12931     dlItem = {\item[#1]},
12932     dlEnd = {\end{description}},
12933     emphasis = {\emph{#1}},
12934     tickedBox = {${\boxtimes$}},
12935     halfTickedBox = {${\boxdot$}}

```

If HTML identifiers appear after a heading, we make them produce `\label` macros.

```

12936 \ExplSyntaxOn
12937 \seq_new:N
12938 \l_@@_header_identifiers_seq
12939 \markdownSetup
12940 {
12941     rendererPrototypes = {
12942         headerAttributeContextBegin = {
12943             \markdownSetup
12944             {

```

```

12945         rendererPrototypes = {
12946             attributeIdentifier = {
12947                 \seq_put_right:Nn
12948                 \l_@@_header_identifiers_seq
12949                 { ##1 }
12950             },
12951         },
12952     }
12953 },
12954 headerAttributeContextEnd = {
12955     \seq_map_inline:Nn
12956     \l_@@_header_identifiers_seq
12957     { \label { ##1 } }
12958     \seq_clear:N
12959     \l_@@_header_identifiers_seq
12960 },
12961 },
12962 }

```

If the `unnumbered` HTML class (or the `{-}` shorthand) appears after a heading the heading and all its subheadings will be unnumbered.

```

12963 \bool_new:N
12964 \l_@@_header_unnumbered_bool
12965 \markdownSetup
12966 {
12967     rendererPrototypes = {
12968         headerAttributeContextBegin += {
12969             \markdownSetup
12970             {
12971                 rendererPrototypes = {
12972                     attributeClassName = {
12973                         \bool_if:nT
12974                         {
12975                             \str_if_eq_p:nn
12976                             { ##1 }
12977                             { unnumbered } &&
12978                             ! \l_@@_header_unnumbered_bool
12979                         }
12980                     {
12981                         \group_begin:
12982                         \bool_set_true:N
12983                         \l_@@_header_unnumbered_bool
12984                         \c@secnumdepth = 0
12985                         \markdownSetup
12986                         {
12987                             rendererPrototypes = {
12988                                 sectionBegin = {

```

```

12989             \group_begin:
12990             },
12991             sectionEnd = {
12992             \group_end:
12993             },
12994             },
12995             }
12996         },
12997     },
12998 },
12999 }
13000 },
13001 },
13002 }
13003 \ExplSyntaxOff
13004 \markdownSetup{rendererPrototypes={
13005     superscript = {\textsuperscript{#1}},
13006     subscript = {\textsubscript{#1}},
13007     blockQuoteBegin = {\begin{quotation}},
13008     blockQuoteEnd = {\end{quotation}},
13009     inputVerbatim = {\VerbatimInput{#1}},
13010     thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
13011     note = {\footnote{#1}}}}

```

### 3.3.4.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

13012 \RequirePackage{ltxcmds}
13013 \ExplSyntaxOn
13014 \cs_gset:Npn
13015   \markdownRendererInputFencedCodePrototype#1#2#3
13016   {
13017     \tl_if_empty:nTF
13018       { #2 }
13019     { \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.

```

13020     {
13021       \regex_extract_once:nnN
13022         { \w* }
13023         { #2 }
13024         \l_tmpa_seq
13025       \seq_pop_left:NN
13026         \l_tmpa_seq
13027         \l_tmpa_tl

```

When the minted package is loaded, use it for syntax highlighting.



```

13028     \ltx@ifpackageloaded
13029         { minted }
13030     {
13031         \catcode`\#=6\relax
13032         \exp_args:NV
13033             \inputminted
13034             \l_tmpa_tl
13035             { #1 }
13036         \catcode`\#=12\relax
13037     }
13038     {

```

When the listings package is loaded, use it for syntax highlighting.

```

13039     \ltx@ifpackageloaded
13040         { listings }
13041         { \lstinputlisting[language=\l_tmpa_tl]{#1} }

```

When neither the listings package nor the minted package is loaded, act as though no infostrng were given.

```

13042         { \markdownRendererInputFencedCode{#1}{}} }
13043     }
13044 }
13045 }
13046 \ExplSyntaxOff

```

Support the nesting of strong emphasis.

```

13047 \ExplSyntaxOn
13048 \def\markdownLATEXStrongEmphasis#1{%
13049   \str_if_in:NnTF
13050     \f@series
13051     { b }
13052     { \textnormal{#1} }
13053     { \textbf{#1} }
13054 }
13055 \ExplSyntaxOff
13056 \markdownSetup{rendererPrototypes={strongEmphasis={%
13057   \protect\markdownLATEXStrongEmphasis{#1}}}}

```

Support L<sup>A</sup>T<sub>E</sub>X document classes that do not provide chapters.

```

13058 \@ifundefined{chapter}{%
13059   \markdownSetup{rendererPrototypes = {
13060     headingOne = {\section{#1}},
13061     headingTwo = {\subsection{#1}},
13062     headingThree = {\subsubsection{#1}},
13063     headingFour = {\paragraph{#1}},
13064     headingFive = {\subparagraph{#1}}}}
13065 }{%
13066   \markdownSetup{rendererPrototypes = {
13067     headingOne = {\chapter{#1}},

```

```

13068     headingTwo = {\section{#1}},
13069     headingThree = {\subsection{#1}},
13070     headingFour = {\subsubsection{#1}},
13071     headingFive = {\paragraph{#1}},
13072     headingSix = {\subparagraph{#1}}}}
13073 }%

```

### 3.3.4.2 Tickboxes

If the `taskLists` option is enabled, we will hide bullets in unordered list items with tickboxes.

```

13074 \markdownSetup{
13075   rendererPrototypes = {
13076     ulItem = {%
13077       \futurelet\markdownLaTeXCheckbox\markdownLaTeXUItem
13078     },
13079   },
13080 }
13081 \def\markdownLaTeXUItem{%
13082   \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
13083     \item[\markdownLaTeXCheckbox]%
13084     \expandafter\@gobble
13085   \else
13086     \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
13087       \item[\markdownLaTeXCheckbox]%
13088       \expandafter\expandafter\expandafter\@gobble
13089     \else
13090       \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
13091         \item[\markdownLaTeXCheckbox]%
13092         \expandafter\expandafter\expandafter\expandafter
13093         \expandafter\expandafter\expandafter\@gobble
13094       \else
13095         \item{}%
13096       \fi
13097     \fi
13098   \fi
13099 }

```

### 3.3.4.3 HTML elements

If the `html` option is enabled and we are using  $\text{T}_{\text{E}}\text{X}4\text{ht}$ <sup>35</sup>, we will pass HTML elements to the output HTML document unchanged.

```

13100 \@ifundefined{HCode}{}{
13101   \markdownSetup{
13102     rendererPrototypes = {
13103       inlineHtmlTag = {%

```

---

<sup>35</sup>See <https://tug.org/tex4ht/>.

```

13104     \ifvmode
13105         \IgnorePar
13106     \EndP
13107     \fi
13108     \HCode{#1}%
13109 },
13110 inputBlockHtmlElement = {%
13111     \ifvmode
13112         \IgnorePar
13113     \fi
13114     \EndP
13115     \special{t4ht*<#1}%
13116     \par
13117     \ShowPar
13118 },
13119 },
13120 }
13121 }

```

### 3.3.4.4 Citations

Here is a basic implementation for citations that uses the L<sup>A</sup>T<sub>E</sub>X `\cite` macro. There are also implementations that use the natbib `\citep`, and `\citet` macros, and the BibL<sup>A</sup>T<sub>E</sub>X `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```

13122 \newcount\markdownLaTeXCitationsCounter
13123
13124 % Basic implementation
13125 \RequirePackage{gobble}
13126 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
13127     \advance\markdownLaTeXCitationsCounter by 1\relax
13128     \ifx\relax#4\relax
13129         \ifx\relax#5\relax
13130             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
13131                 \cite{#1#2#6}% Without prenotes and postnotes, just accumulate cites
13132                 \expandafter\expandafter\expandafter
13133                 \expandafter\expandafter\expandafter\expandafter
13134                 \@gobblethree
13135             \fi
13136         \else% Before a postnote (#5), dump the accumulator
13137             \ifx\relax#1\relax\else
13138                 \cite{#1}%
13139             \fi
13140             \cite[#5]{#6}%
13141         \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
13142         \else
13143             \expandafter\expandafter\expandafter

```

```

13144     \expandafter\expandafter\expandafter\expandafter
13145     \expandafter\expandafter\expandafter
13146     \expandafter\expandafter\expandafter\expandafter
13147     \markdownLaTeXBasicCitations
13148     \fi
13149     \expandafter\expandafter\expandafter
13150     \expandafter\expandafter\expandafter\expandafter{%
13151     \expandafter\expandafter\expandafter
13152     \expandafter\expandafter\expandafter\expandafter}%
13153     \expandafter\expandafter\expandafter
13154     \expandafter\expandafter\expandafter\expandafter{%
13155     \expandafter\expandafter\expandafter
13156     \expandafter\expandafter\expandafter\expandafter}%
13157     \expandafter\expandafter\expandafter
13158     \@gobblethree
13159     \fi
13160 \else% Before a prenote (#4), dump the accumulator
13161     \ifx\relax#1\relax\else
13162         \cite{#1}%
13163     \fi
13164     \ifnum\markdownLaTeXCitationsCounter>1\relax
13165         \space % Insert a space before the prenote in later citations
13166     \fi
13167     #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
13168     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
13169     \else
13170         \expandafter\expandafter\expandafter
13171         \expandafter\expandafter\expandafter\expandafter
13172         \markdownLaTeXBasicCitations
13173     \fi
13174     \expandafter\expandafter\expandafter{%
13175     \expandafter\expandafter\expandafter}%
13176     \expandafter\expandafter\expandafter{%
13177     \expandafter\expandafter\expandafter}%
13178     \expandafter
13179     \@gobblethree
13180     \fi\markdownLaTeXBasicCitations{#1#2#6},}
13181 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
13182
13183 % Natbib implementation
13184 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
13185     \advance\markdownLaTeXCitationsCounter by 1\relax
13186     \ifx\relax#3\relax
13187         \ifx\relax#4\relax
13188             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
13189                 \citep{#1,#5}% Without prenotes and postnotes, just accumulate cites
13190             \expandafter\expandafter\expandafter

```

```

13191         \expandafter\expandafter\expandafter\expandafter
13192         \@gobbletwo
13193     \fi
13194 \else% Before a postnote (#4), dump the accumulator
13195     \ifx\relax#1\relax\else
13196         \citep{#1}%
13197     \fi
13198     \citep[] [#4]{#5}%
13199     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
13200     \else
13201         \expandafter\expandafter\expandafter
13202         \expandafter\expandafter\expandafter\expandafter
13203         \expandafter\expandafter\expandafter
13204         \expandafter\expandafter\expandafter\expandafter
13205         \markdownLaTeXNatbibCitations
13206     \fi
13207     \expandafter\expandafter\expandafter
13208     \expandafter\expandafter\expandafter\expandafter{%
13209     \expandafter\expandafter\expandafter
13210     \expandafter\expandafter\expandafter\expandafter}%
13211     \expandafter\expandafter\expandafter
13212     \@gobbletwo
13213 \fi
13214 \else% Before a prenote (#3), dump the accumulator
13215     \ifx\relax#1\relax\relax\else
13216         \citep{#1}%
13217     \fi
13218     \citep[#3] [#4]{#5}%
13219     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
13220     \else
13221         \expandafter\expandafter\expandafter
13222         \expandafter\expandafter\expandafter\expandafter
13223         \markdownLaTeXNatbibCitations
13224     \fi
13225     \expandafter\expandafter\expandafter{%
13226     \expandafter\expandafter\expandafter}%
13227     \expandafter
13228     \@gobbletwo
13229 \fi\markdownLaTeXNatbibCitations{#1,#5}}
13230 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
13231     \advance\markdownLaTeXCitationsCounter by 1\relax
13232     \ifx\relax#3\relax
13233         \ifx\relax#4\relax
13234             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
13235                 \citet{#1,#5}% Without prenotes and postnotes, just accumulate cites
13236             \expandafter\expandafter\expandafter
13237             \expandafter\expandafter\expandafter\expandafter

```

```

13238     \@gobbletwo
13239     \fi
13240 \else% After a prenote or a postnote, dump the accumulator
13241     \ifx\relax#1\relax\else
13242         \citet{#1}%
13243     \fi
13244     , \citet[#3][#4]{#5}%
13245     \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
13246     ,
13247     \else
13248         \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
13249         ,
13250     \fi
13251     \fi
13252     \expandafter\expandafter\expandafter
13253     \expandafter\expandafter\expandafter\expandafter
13254     \markdownLaTeXNatbibTextCitations
13255     \expandafter\expandafter\expandafter
13256     \expandafter\expandafter\expandafter\expandafter{%
13257     \expandafter\expandafter\expandafter
13258     \expandafter\expandafter\expandafter\expandafter}%
13259     \expandafter\expandafter\expandafter
13260     \@gobbletwo
13261     \fi
13262 \else% After a prenote or a postnote, dump the accumulator
13263     \ifx\relax#1\relax\relax\else
13264         \citet{#1}%
13265     \fi
13266     , \citet[#3][#4]{#5}%
13267     \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
13268     ,
13269     \else
13270         \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
13271         ,
13272     \fi
13273     \fi
13274     \expandafter\expandafter\expandafter
13275     \markdownLaTeXNatbibTextCitations
13276     \expandafter\expandafter\expandafter{%
13277     \expandafter\expandafter\expandafter}%
13278     \expandafter
13279     \@gobbletwo
13280 \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
13281
13282 % BibLaTeX implementation
13283 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
13284     \advance\markdownLaTeXCitationsCounter by 1\relax

```

```

13285 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
13286 \autocites#1[#3][#4]{#5}%
13287 \expandafter\@gobbletwo
13288 \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}
13289 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
13290 \advance\markdownLaTeXCitationsCounter by 1\relax
13291 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
13292 \textcites#1[#3][#4]{#5}%
13293 \expandafter\@gobbletwo
13294 \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}
13295
13296 \markdownSetup{rendererPrototypes = {
13297 cite = {%
13298 \markdownLaTeXCitationsCounter=1%
13299 \def\markdownLaTeXCitationsTotal{#1}%
13300 \@ifundefined{autocites}{%
13301 \ifundefined{citep}{%
13302 \expandafter\expandafter\expandafter
13303 \markdownLaTeXBasicCitations
13304 \expandafter\expandafter\expandafter{%
13305 \expandafter\expandafter\expandafter}%
13306 \expandafter\expandafter\expandafter{%
13307 \expandafter\expandafter\expandafter}%
13308 }{%
13309 \expandafter\expandafter\expandafter
13310 \markdownLaTeXNatbibCitations
13311 \expandafter\expandafter\expandafter{%
13312 \expandafter\expandafter\expandafter}%
13313 }%
13314 }{%
13315 \expandafter\expandafter\expandafter
13316 \markdownLaTeXBibLaTeXCitations
13317 \expandafter{\expandafter}%
13318 }},
13319 textCite = {%
13320 \markdownLaTeXCitationsCounter=1%
13321 \def\markdownLaTeXCitationsTotal{#1}%
13322 \@ifundefined{autocites}{%
13323 \ifundefined{citep}{%
13324 \expandafter\expandafter\expandafter
13325 \markdownLaTeXBasicTextCitations
13326 \expandafter\expandafter\expandafter{%
13327 \expandafter\expandafter\expandafter}%
13328 \expandafter\expandafter\expandafter{%
13329 \expandafter\expandafter\expandafter}%
13330 }{%
13331 \expandafter\expandafter\expandafter

```

```

13332     \markdownLaTeXNatbibTextCitations
13333     \expandafter\expandafter\expandafter{%
13334     \expandafter\expandafter\expandafter}%
13335   }%
13336 }{%
13337   \expandafter\expandafter\expandafter
13338   \markdownLaTeXBibLaTeXTextCitations
13339   \expandafter{\expandafter}%
13340   }}}

```

### 3.3.4.5 Links

Here is an implementation for hypertext links and relative references.

```

13341 \RequirePackage{url}
13342 \RequirePackage{expl3}
13343 \ExplSyntaxOn
13344 \def\markdownRendererLinkPrototype#1#2#3#4{
13345   \tl_set:Nn \l_tmpa_tl { #1 }
13346   \tl_set:Nn \l_tmpb_tl { #2 }
13347   \bool_set:Nn
13348     \l_tmpa_bool
13349     {
13350       \tl_if_eq_p:NN
13351         \l_tmpa_tl
13352         \l_tmpb_tl
13353     }
13354   \tl_set:Nn \l_tmpa_tl { #4 }
13355   \bool_set:Nn
13356     \l_tmpb_bool
13357     {
13358       \tl_if_empty_p:N
13359         \l_tmpa_tl
13360     }

```

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.

```

13361   \bool_if:nTF
13362     {
13363       \l_tmpa_bool && \l_tmpb_bool
13364     }
13365     {
13366       \markdownLaTeXRendererAutolink { #2 } { #3 }
13367     }{
13368       \markdownLaTeXRendererDirectOrIndirectLink { #1 } { #2 } { #3 } { #4 }
13369     }
13370 }
13371 \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.

```

13372 \tl_set:Nn
13373   \l_tmpa_tl
13374   { #2 }
13375 \tl_trim_spaces:N
13376   \l_tmpa_tl
13377 \tl_set:Nx
13378   \l_tmpb_tl
13379   {
13380     \tl_range:Nnn
13381       \l_tmpa_tl
13382       { 1 }
13383       { 1 }
13384   }
13385 \str_if_eq:NNTF
13386   \l_tmpb_tl
13387   \c_hash_str
13388   {
13389     \tl_set:Nx
13390     \l_tmpb_tl
13391     {
13392       \tl_range:Nnn
13393         \l_tmpa_tl
13394         { 2 }
13395         { -1 }
13396     }
13397     \exp_args:NV
13398     \ref
13399     \l_tmpb_tl
13400   }{
13401     \url { #2 }
13402   }
13403 }
13404 \ExplSyntaxOff
13405 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
13406   #1\footnote{\ifx\empty#4\empty\else#4: \fi\url{#3}}}
```

### 3.3.4.6 Tables

Here is a basic implementation of tables. If the booktabs package is loaded, then it is used to produce horizontal lines.

```

13407 \newcount\markdownLaTeXRowCount
13408 \newcount\markdownLaTeXRowTotal
13409 \newcount\markdownLaTeXColumnCounter
13410 \newcount\markdownLaTeXColumnTotal
13411 \newtoks\markdownLaTeXTable
```

```

13412 \newtoks\markdownLaTeXTableAlignment
13413 \newtoks\markdownLaTeXTableEnd
13414 \AtBeginDocument{%
13415   \@ifpackageloaded{booktabs}{%
13416     \def\markdownLaTeXTopRule{\toprule}%
13417     \def\markdownLaTeXMidRule{\midrule}%
13418     \def\markdownLaTeXBottomRule{\bottomrule}%
13419   }{%
13420     \def\markdownLaTeXTopRule{\hline}%
13421     \def\markdownLaTeXMidRule{\hline}%
13422     \def\markdownLaTeXBottomRule{\hline}%
13423   }%
13424 }
13425 \markdownSetup{rendererPrototypes={
13426   table = {%
13427     \markdownLaTeXTable={}%
13428     \markdownLaTeXTableAlignment={}%
13429     \markdownLaTeXTableEnd={%
13430       \markdownLaTeXBottomRule
13431       \end{tabular}}%
13432     \ifx\empty#1\empty\else
13433       \addto@hook\markdownLaTeXTable{%
13434         \begin{table}
13435           \centering%
13436         \addto@hook\markdownLaTeXTableEnd{%
13437           \caption{#1}
13438           \end{table}}%
13439       \fi
13440       \addto@hook\markdownLaTeXTable{\begin{tabular}}%
13441       \markdownLaTeXRowCount=0%
13442       \markdownLaTeXRowTotal=#2%
13443       \markdownLaTeXColumnTotal=#3%
13444       \markdownLaTeXRenderTableRow
13445     }
13446   }}
13447 \def\markdownLaTeXRenderTableRow#1{%
13448   \markdownLaTeXColumnCounter=0%
13449   \ifnum\markdownLaTeXRowCount=0\relax
13450     \markdownLaTeXReadAlignments#1%
13451     \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
13452       \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
13453         \the\markdownLaTeXTableAlignment}}%
13454     \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
13455   \else
13456     \markdownLaTeXRenderTableCell#1%
13457   \fi
13458   \ifnum\markdownLaTeXRowCount=1\relax

```

```

13459     \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
13460     \fi
13461     \advance\markdownLaTeXRowCount by 1\relax
13462     \ifnum\markdownLaTeXRowCount>\markdownLaTeXRowTotal\relax
13463         \the\markdownLaTeXTable
13464         \the\markdownLaTeXTableEnd
13465         \expandafter\@gobble
13466     \fi\markdownLaTeXRenderTableRow}
13467 \def\markdownLaTeXReadAlignments#1{%
13468     \advance\markdownLaTeXColumnCounter by 1\relax
13469     \if#1d%
13470         \addto@hook\markdownLaTeXTableAlignment{1}%
13471     \else
13472         \addto@hook\markdownLaTeXTableAlignment{#1}%
13473     \fi
13474     \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
13475         \expandafter\@gobble
13476     \fi\markdownLaTeXReadAlignments}
13477 \def\markdownLaTeXRenderTableCell#1{%
13478     \advance\markdownLaTeXColumnCounter by 1\relax
13479     \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
13480         \addto@hook\markdownLaTeXTable{#1&}%
13481     \else
13482         \addto@hook\markdownLaTeXTable{#1\\}%
13483         \expandafter\@gobble
13484     \fi\markdownLaTeXRenderTableCell}

```

### 3.3.4.7 Line Blocks

Here is a basic implementation of line blocks. If the `verse` package is loaded, then it is used to produce the verses.

```

13485
13486 \markdownIfOption{lineBlocks}{%
13487     \RequirePackage{verse}
13488     \markdownSetup{rendererPrototypes={
13489         lineBlockBegin = {%
13490             \begingroup
13491             \def\markdownRendererHardLineBreak{\\}%
13492             \begin{verse}%
13493         },
13494         lineBlockEnd = {%
13495             \end{verse}%
13496         \endgroup
13497     },
13498 }}
13499 }{}
13500

```

### 3.3.4.8 YAML Metadata

The default setup of YAML metadata will invoke the `\title`, `\author`, and `\date` macros when scalar values for keys that correspond to the `title`, `author`, and `date` relative wildcards are encountered, respectively.

```
13501 \ExplSyntaxOn
13502 \keys_define:nn
13503   { markdown/jekyllData }
13504   {
13505     author .code:n = { \author{#1} },
13506     date   .code:n = { \date{#1}   },
13507     title  .code:n = { \title{#1}  },
13508   }
```

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.

```
13509 \markdownSetup{
13510   rendererPrototypes = {
13511     jekyllDataEnd = {
13512       \AddToHook{begindocument/end}{\maketitle}
13513     },
13514   },
13515 }
13516 \ExplSyntaxOff
```

### 3.3.4.9 Strike-Through

If the `strikeThrough` option is enabled, we will load the `soulutf8` package and use it to implement strike-throughs.

```
13517 \markdownIfOption{strikeThrough}{%
13518   \RequirePackage{soulutf8}%
13519   \markdownSetup{
13520     rendererPrototypes = {
13521       strikeThrough = {%
13522         \st{#1}%
13523       },
13524     }
13525   }
13526 }{}
```

### 3.3.4.10 Marked Text

If the `mark` option is enabled, we will load the `soulutf8` package and use it to implement marked text.

```
13527 \markdownIfOption{mark}{%
13528   \RequirePackage{soulutf8}%
```

```

13529 \markdownSetup{
13530     rendererPrototypes = {
13531         mark = {%
13532             \hl{#1}%
13533         },
13534     }
13535 }
13536 }{}

```

### 3.3.4.11 Image Attributes

If the `linkAttributes` option is enabled, we will load the `graphicx` package. Furthermore, in image attribute contexts, we will make attributes in the form  $\langle key \rangle = \langle value \rangle$  set the corresponding keys of the `graphicx` package to the corresponding values.

```

13537 \ExplSyntaxOn
13538 \@@_if_option:nT
13539 { linkAttributes }
13540 {
13541     \RequirePackage{graphicx}
13542     \markdownSetup{
13543         rendererPrototypes = {
13544             imageAttributeContextBegin = {
13545                 \group_begin:
13546                 \markdownSetup{
13547                     rendererPrototypes = {
13548                         attributeKeyValue = {
13549                             \setkeys
13550                             { Gin }
13551                             { { ##1 } = { ##2 } }
13552                         },
13553                     },
13554                 }
13555             },
13556             imageAttributeContextEnd = {
13557                 \group_end:
13558             },
13559         },
13560     }
13561 }
13562 \ExplSyntaxOff

```

### 3.3.4.12 Raw Attributes

In the raw block and inline raw span renderer prototypes, default to the plain TeX renderer prototypes, translating raw attribute `latex` to `tex`.

```

13563 \ExplSyntaxOn

```

```

13564 \cs_gset:Npn
13565   \markdownRendererInputRawInlinePrototype#1#2
13566   {
13567     \str_case:nnF
13568       { #2 }
13569       {
13570         { latex }
13571         {
13572           \@_plain_tex_default_input_raw_inline_renderer_prototype:nn
13573             { #1 }
13574             { tex }
13575         }
13576       }
13577     {
13578       \@_plain_tex_default_input_raw_inline_renderer_prototype:nn
13579         { #1 }
13580         { #2 }
13581     }
13582   }
13583 \cs_gset:Npn
13584   \markdownRendererInputRawBlockPrototype#1#2
13585   {
13586     \str_case:nnF
13587       { #2 }
13588       {
13589         { latex }
13590         {
13591           \@_plain_tex_default_input_raw_block_renderer_prototype:nn
13592             { #1 }
13593             { tex }
13594         }
13595       }
13596     {
13597       \@_plain_tex_default_input_raw_block_renderer_prototype:nn
13598         { #1 }
13599         { #2 }
13600     }
13601   }
13602 \ExplSyntaxOff
13603 \fi % Closes ~\markdownIfOption{plain}{\iffalse}{\iftrue}~

```

### 3.3.5 Miscellanea

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `inputenc` package. We will do this by redefining the

`\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the `filecontents` package.

```
13604 \newcommand\markdownMakeOther{%
13605   \count0=128\relax
13606   \loop
13607     \catcode\count0=11\relax
13608     \advance\count0 by 1\relax
13609   \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.

```
13610 \def\markdownMakeOther{%
13611   \count0=128\relax
13612   \loop
13613     \catcode\count0=11\relax
13614     \advance\count0 by 1\relax
13615   \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.

```
13616   \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).

```
13617 \long\def\inputmarkdown{%
13618   \dosingleempty
13619   \doinputmarkdown}%
13620 \long\def\doinputmarkdown[#1]#2{%
13621   \begingroup
13622     \iffirstargument
13623       \setupmarkdown[#1]%
13624     \fi
13625     \markdownInput{#2}%
13626   \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. [13, sec. 31]. According to Eijkhout [14, sec. 2.2], this is because “these spaces are hard to see in an editor”. At the moment, there is no option to suppress this behavior in (Lua)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.

```

13627 \startluacode
13628   document.markdown_buffering = false
13629   local function preserve_trailing_spaces(line)
13630     if document.markdown_buffering then
13631       line = line:gsub("[ \t][ \t]$", "\t\t")
13632     end
13633     return line
13634   end
13635   resolvers.installinputlinehandler(preserve_trailing_spaces)
13636 \stopluacode
13637 \begingroup
13638   \catcode\|=0%
13639   \catcode\|=12%
13640   |gdef|startmarkdown{%
13641     |ctxlua{document.markdown_buffering = true}%
13642     |markdownReadAndConvert{\stopmarkdown}%
13643                               {|stopmarkdown}}%
13644   |gdef|stopmarkdown{%
13645     |ctxlua{document.markdown_buffering = false}%
13646     |markdownEnd}%
13647 |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.

```

13648 \ExplSyntaxOn
13649 \cs_gset:Nn
13650   \@@_load_theme:nn
13651   {

```

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.

```

13652   \file_if_exist:nTF
13653     { t - markdown theme #2.tex }
13654     {
13655       \msg_info:nnn

```



```

13656         { markdown }
13657         { loading-context-theme }
13658         { #1 }
13659         \usemodule
13660         [ t ]
13661         [ markdown theme #2 ]
13662     }
13663     {
13664         \@_plain_tex_load_theme:nn
13665         { #1 }
13666         { #2 }
13667     }
13668 }
13669 \msg_new:nnn
13670 { markdown }
13671 { loading-context-theme }
13672 { Loading~ConTeXt~Markdown~theme~#1 }
13673 \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:

```
13674 \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.

```

13675 \markdownIfOption{plain}{\iffalse}{\iftrue}
13676 \def\markdownRendererHardLineBreakPrototype{\blank}%
13677 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
13678 \def\markdownRendererRightBracePrototype{\textbraceright}%
13679 \def\markdownRendererDollarSignPrototype{\textdollar}%
13680 \def\markdownRendererPercentSignPrototype{\percent}%
13681 \def\markdownRendererUnderscorePrototype{\textunderscore}%
13682 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
13683 \def\markdownRendererBackslashPrototype{\textbackslash}%
13684 \def\markdownRendererTildePrototype{\textasciitilde}%
13685 \def\markdownRendererPipePrototype{\char`|}%
13686 \def\markdownRendererLinkPrototype#1#2#3#4{%
13687   \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
13688   \fi\texttt<\hyphenatedurl{#3}>}}%
13689 \usemodule[database]
13690 \defineseparatedlist

```

```

13691 [MarkdownConTeXtCSV]
13692 [separator={,},
13693 before=\bTABLE,after=\eTABLE,
13694 first=\bTR,last=\eTR,
13695 left=\bTD,right=\eTD]
13696 \def\markdownConTeXtCSV{csv}
13697 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
13698 \def\markdownConTeXtCSV@arg{#1}%
13699 \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
13700 \placetable [] [tab:#1]{#4}{%
13701 \processeparatedfile [MarkdownConTeXtCSV] [#3]}%
13702 \else
13703 \markdownInput{#3}%
13704 \fi}%
13705 \def\markdownRendererImagePrototype#1#2#3#4{%
13706 \placefigure [] []{#4}{\externalfigure [#3]}}%
13707 \def\markdownRendererUlBeginPrototype{\startitemize}%
13708 \def\markdownRendererUlBeginTightPrototype{\startitemize [packed]}%
13709 \def\markdownRendererUlItemPrototype{\item}%
13710 \def\markdownRendererUlEndPrototype{\stopitemize}%
13711 \def\markdownRendererUlEndTightPrototype{\stopitemize}%
13712 \def\markdownRendererOlBeginPrototype{\startitemize [n]}%
13713 \def\markdownRendererOlBeginTightPrototype{\startitemize [packed,n]}%
13714 \def\markdownRendererOlItemPrototype{\item}%
13715 \def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
13716 \def\markdownRendererOlEndPrototype{\stopitemize}%
13717 \def\markdownRendererOlEndTightPrototype{\stopitemize}%
13718 \definedescription
13719 [MarkdownConTeXtDlItemPrototype]
13720 [location=hanging,
13721 margin=standard,
13722 headstyle=bold]%
13723 \definestartstop
13724 [MarkdownConTeXtDlPrototype]
13725 [before=\blank,
13726 after=\blank]%
13727 \definestartstop
13728 [MarkdownConTeXtDlTightPrototype]
13729 [before=\blank\startpacked,
13730 after=\stoppacked\blank]%
13731 \def\markdownRendererDlBeginPrototype{%
13732 \startMarkdownConTeXtDlPrototype}%
13733 \def\markdownRendererDlBeginTightPrototype{%
13734 \startMarkdownConTeXtDlTightPrototype}%
13735 \def\markdownRendererDlItemPrototype#1{%
13736 \startMarkdownConTeXtDlItemPrototype{#1}}%
13737 \def\markdownRendererDlItemEndPrototype{%

```

```

13738 \stopMarkdownConTeXtDlItemPrototype}%
13739 \def\markdownRendererDlEndPrototype{%
13740 \stopMarkdownConTeXtDlPrototype}%
13741 \def\markdownRendererDlEndTightPrototype{%
13742 \stopMarkdownConTeXtDlTightPrototype}%
13743 \def\markdownRendererEmphasisPrototype#1{\em#1}%
13744 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
13745 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
13746 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
13747 \def\markdownRendererLineBlockBeginPrototype{%
13748 \begingroup
13749 \def\markdownRendererHardLineBreak{
13750 }%
13751 \startlines
13752 }%
13753 \def\markdownRendererLineBlockEndPrototype{%
13754 \stoptlines
13755 \endgroup
13756 }%
13757 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%

```

### 3.4.3.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

13758 \ExplSyntaxOn
13759 \cs_gset:Npn
13760 \markdownRendererInputFencedCodePrototype#1#2#3
13761 {
13762 \tl_if_empty:nTF
13763 { #2 }
13764 { \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 `\definetyping` macro, which allows the user to set up code highlighting mapping as follows:

```

\definetyping [latex]
\setuptyping [latex] [option=TEX]

\starttext
\startmarkdown
~~~ latex
\documentclass{article}
\begin{document}
Hello world!

```

```

\end{document}
~~~
\stopmarkdown
\stoptext

```

```

13765     {
13766         \regex_extract_once:nnN
13767         { \w* }
13768         { #2 }
13769         \l_tmpa_seq
13770     \seq_pop_left:NN
13771         \l_tmpa_seq
13772         \l_tmpa_tl
13773     \typefile[\l_tmpa_tl] []{#1}
13774     }
13775 }
13776 \ExplSyntaxOff
13777 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
13778 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
13779 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
13780 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
13781 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
13782 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
13783 \def\markdownRendererThematicBreakPrototype{%
13784     \blackrule[height=1pt, width=\hsize]}%
13785 \def\markdownRendererNotePrototype#1{\footnote{#1}}%
13786 \def\markdownRendererTickedBoxPrototype{${\boxtimes}$}
13787 \def\markdownRendererHalfTickedBoxPrototype{${\boxdot}$}
13788 \def\markdownRendererUntickedBoxPrototype{${\square}$}
13789 \def\markdownRendererStrikeThroughPrototype#1{\overstrikes{#1}}
13790 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
13791 \def\markdownRendererSubscriptPrototype#1{\low{#1}}
13792 \def\markdownRendererDisplayMathPrototype#1{\startformula#1\stopformula}%

```

### 3.4.3.2 Tables

There is a basic implementation of tables.

```

13793 \newcount\markdownConTeXtRowCounter
13794 \newcount\markdownConTeXtRowTotal
13795 \newcount\markdownConTeXtColumnCounter
13796 \newcount\markdownConTeXtColumnTotal
13797 \newtoks\markdownConTeXtTable
13798 \newtoks\markdownConTeXtTableFloat
13799 \def\markdownRendererTablePrototype#1#2#3{%
13800     \markdownConTeXtTable={}%
13801     \ifx\empty#1\empty
13802     \markdownConTeXtTableFloat={%

```

```

13803     \the\markdownConTeXtTable}%
13804 \else
13805     \markdownConTeXtTableFloat={%
13806     \placetable{#1}{\the\markdownConTeXtTable}}%
13807 \fi
13808 \begingroup
13809 \setupTABLE[r][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
13810 \setupTABLE[c][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
13811 \setupTABLE[r][1][topframe=on, bottomframe=on]
13812 \setupTABLE[r][#1][bottomframe=on]
13813 \markdownConTeXtRowCounter=0%
13814 \markdownConTeXtRowTotal=#2%
13815 \markdownConTeXtColumnTotal=#3%
13816 \markdownConTeXtRenderTableRow}
13817 \def\markdownConTeXtRenderTableRow#1{%
13818 \markdownConTeXtColumnCounter=0%
13819 \ifnum\markdownConTeXtRowCounter=0\relax
13820 \markdownConTeXtReadAlignments#1%
13821 \markdownConTeXtTable={\bTABLE}%
13822 \else
13823 \markdownConTeXtTable=\expandafter{%
13824 \the\markdownConTeXtTable\bTR}%
13825 \markdownConTeXtRenderTableCell#1%
13826 \markdownConTeXtTable=\expandafter{%
13827 \the\markdownConTeXtTable\eTR}%
13828 \fi
13829 \advance\markdownConTeXtRowCounter by 1\relax
13830 \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
13831 \markdownConTeXtTable=\expandafter{%
13832 \the\markdownConTeXtTable\eTABLE}%
13833 \the\markdownConTeXtTableFloat
13834 \endgroup
13835 \expandafter\gobbleoneargument
13836 \fi\markdownConTeXtRenderTableRow}
13837 \def\markdownConTeXtReadAlignments#1{%
13838 \advance\markdownConTeXtColumnCounter by 1\relax
13839 \if#1d%
13840 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
13841 \fi\if#1l%
13842 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
13843 \fi\if#1c%
13844 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
13845 \fi\if#1r%
13846 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
13847 \fi
13848 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
13849 \expandafter\gobbleoneargument

```

```

13850 \fi\markdownConTeXtReadAlignments}
13851 \def\markdownConTeXtRenderTableCell#1{%
13852 \advance\markdownConTeXtColumnCounter by 1\relax
13853 \markdownConTeXtTable=\expandafter{%
13854 \the\markdownConTeXtTable\bTD#1\eTD}%
13855 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
13856 \expandafter\gobbleoneargument
13857 \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`.

```

13858 \ExplSyntaxOn
13859 \cs_gset:Npn
13860 \markdownRendererInputRawInlinePrototype#1#2
13861 {
13862 \str_case:nnF
13863 { #2 }
13864 {
13865 { latex }
13866 {
13867 \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
13868 { #1 }
13869 { context }
13870 }
13871 }
13872 {
13873 \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
13874 { #1 }
13875 { #2 }
13876 }
13877 }
13878 \cs_gset:Npn
13879 \markdownRendererInputRawBlockPrototype#1#2
13880 {
13881 \str_case:nnF
13882 { #2 }
13883 {
13884 { context }
13885 {
13886 \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
13887 { #1 }
13888 { tex }
13889 }
13890 }
13891 {

```

```

13892     \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
13893     { #1 }
13894     { #2 }
13895   }
13896 }
13897 \cs_gset_eq:NN
13898   \markdownRendererInputRawBlockPrototype
13899   \markdownRendererInputRawInlinePrototype
13900 \fi % Closes ` \markdownIfOption{plain}{\iffalse}{\iftrue}`
13901 \ExplSyntaxOff
13902 \stopmodule
13903 \protect

```

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

```

13904 \markdownIfOption{noDefaults}{}{
13905   \setupmarkdown[theme=witiko/markdown/defaults]
13906 }
13907 \stopmodule
13908 \protect

```

## References

- [1] LuaTeX development team. *LuaTeX 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 TeXbook*. 3rd ed. Vol. A. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. ix, 479. ISBN: 0-201-13447-0.
- [7] Frank Mittelbach. *The doc and shortvrb Packages*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/doc.pdf> (visited on 02/19/2018).

- [8] Till Tantau, Joseph Wright, and Vedran Miletić. *The Beamer class*. Feb. 10, 2021. URL: <https://mirrors.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf> (visited on 02/11/2021).
- [9] Vít Starý Novotný et al. *Convert control sequence with a variable number of undelimited parameters into a token list*. URL: <https://tex.stackexchange.com/q/716362/70941> (visited on 04/28/2024).
- [10] Geoffrey M. Poore. *The minted Package. Highlighted source code in L<sup>A</sup>T<sub>E</sub>X*. July 19, 2017. URL: <https://mirrors.ctan.org/macros/latex/contrib/minted/minted.pdf> (visited on 09/01/2020).
- [11] Roberto Ierusalimsky. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.
- [12] Johannes Braams et al. *The L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> Sources*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/source2e.pdf> (visited on 01/08/2018).
- [13] Donald Ervin Knuth. *T<sub>E</sub>X: The Program*. Vol. B. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. xvi, 594. ISBN: 978-0-201-13437-7.
- [14] Victor Eijkhout. *T<sub>E</sub>X by Topic. A T<sub>E</sub>Xnician's Reference*. Wokingham, England: Addison-Wesley, Feb. 1, 1992. 307 pp. ISBN: 978-0-201-56882-0.

## Index

autoIdentifiers	19, 31, 73, 86
blankBeforeBlockquote	19
blankBeforeCodeFence	20
blankBeforeDivFence	20
blankBeforeHeading	20
blankBeforeList	21
bracketedSpans	21, 75
breakableBlockquotes	21
cacheDir	4, 15, 18, 54, 55, 132, 284, 352, 367
citationNbsps	22
citations	22, 77, 78
codeSpans	23
contentBlocks	18, 23
contentBlocksLanguageMap	18
contentLevel	24
debugExtensions	9, 18, 24, 280



debugExtensionsFileName	18, 24
defaultOptions	9, 48, 332
definitionLists	24, 81
eagerCache	15
entities.char_entity	190
entities.dec_entity	190
entities.hex_entity	190
entities.hex_entity_with_x_char	190
expandtabs	244
expectJekyllData	25
extensions	26, 140, 286
extensions.bracketed_spans	286
extensions.citations	287
extensions.content_blocks	291
extensions.definition_lists	294
extensions.fancy_lists	296
extensions.fenced_code	302
extensions.fenced_divs	307
extensions.header_attributes	311
extensions.inline_code_attributes	312
extensions.jekyll_data	328
extensions.line_blocks	313
extensions.link_attributes	314
extensions.mark	314
extensions.notes	316
extensions.pipe_table	318
extensions.raw_inline	322
extensions.strike_through	323
extensions.subscripts	323
extensions.superscripts	324
extensions.tex_math	325
fancyLists	28, 96–100, 368
fencedCode	28, 36, 78, 85, 101, 366
fencedCodeAttributes	29, 73
fencedDiv	86
fencedDivs	29, 38
finalizeCache	16, 19, 30, 30, 54, 55, 131, 286
frozenCache	19, 30, 55, 131, 134, 135, 366, 367
frozenCacheCounter	30, 286, 359, 360
frozenCacheFileName	19, 30, 54, 286

<code>gfmAutoIdentifiers</code>	19, 30, 73, 86
<code>hashEnumerators</code>	31
<code>headerAttributes</code>	31, 38, 73, 86
<code>html</code>	32, 89, 378
<code>hybrid</code>	32, 37, 43, 45, 58, 66, 102, 132, 195, 245, 359
<code>inlineCodeAttributes</code>	32, 73, 79
<code>inlineNotes</code>	33
<code>\input</code>	52
<code>\inputmarkdown</code>	137, 137, 138, 391
<code>inputTempFileName</code>	55, 58, 353, 354, 356, 357
<code>iterlines</code>	243
<code>jeekyllData</code>	3, 25, 26, 33, 109–112
<code>\l_file_search_path_seq</code>	358
<code>languages_json</code>	291, 291
<code>lineBlocks</code>	34, 91
<code>linkAttributes</code>	34, 73, 90, 93, 263, 389
<code>mark</code>	35, 94, 388
<code>\markdown</code>	130
<code>markdown</code>	130, 130, 361
<code>markdown*</code>	130, 130, 131, 361
<code>\markdown_jeekyll_data_concatenate_address:NN</code>	347
<code>\markdown_jeekyll_data_pop:</code>	348
<code>\markdown_jeekyll_data_push:nN</code>	348
<code>\markdown_jeekyll_data_push_address_segment:n</code>	346
<code>\markdown_jeekyll_data_set_keyval:Nn</code>	348
<code>\markdown_jeekyll_data_set_keyvals:nn</code>	348
<code>\markdown_jeekyll_data_update_address_tls:</code>	347
<code>\markdownBegin</code>	50, 50–52, 128, 130, 137
<code>\markdownCleanup</code>	352
<code>\markdownEnd</code>	50, 50–52, 128, 130, 137
<code>\markdownError</code>	128, 128
<code>\markdownEscape</code>	50, 52, 360
<code>\markdownIfOption</code>	53
<code>\markdownIfSnippetExists</code>	67
<code>\markdownInfo</code>	128, 128
<code>\markdownInput</code>	50, 52, 130, 131, 137, 358, 361
<code>\markdownInputFileStream</code>	352
<code>\markdownInputPlainTeX</code>	361

<code>\markdownLoadPlainTeXTheme</code>	132, 139, 341
<code>\markdownLuaExecute</code>	355, 358
<code>\markdownLuaOptions</code>	350, 352
<code>\markdownMakeOther</code>	128, 391
<code>\markdownOptionFinalizeCache</code>	54
<code>\markdownOptionFrozenCache</code>	54
<code>\markdownOptionHybrid</code>	58
<code>\markdownOptionInputTempFileName</code>	55
<code>\markdownOptionNoDefaults</code>	57
<code>\markdownOptionOutputDir</code>	55, 55
<code>\markdownOptionPlain</code>	56
<code>\markdownOptionStripPercentSigns</code>	57
<code>\markdownOutputFileStream</code>	352
<code>\markdownPrepare</code>	352
<code>\markdownPrepareLuaOptions</code>	350
<code>\markdownReadAndConvert</code>	128, 353, 361, 362, 392
<code>\markdownReadAndConvertProcessLine</code>	354, 355
<code>\markdownReadAndConvertStripPercentSigns</code>	353
<code>\markdownReadAndConvertTab</code>	352
<code>\markdownRendererAttributeClassName</code>	73
<code>\markdownRendererAttributeIdentifier</code>	73
<code>\markdownRendererAttributeKeyValue</code>	73
<code>\markdownRendererBlockQuoteBegin</code>	74
<code>\markdownRendererBlockQuoteEnd</code>	74
<code>\markdownRendererBracketedSpanAttributeContextBegin</code>	75
<code>\markdownRendererBracketedSpanAttributeContextEnd</code>	75
<code>\markdownRendererCite</code>	77, 78
<code>\markdownRendererCodeSpan</code>	79
<code>\markdownRendererCodeSpanAttributeContextBegin</code>	79
<code>\markdownRendererCodeSpanAttributeContextEnd</code>	79
<code>\markdownRendererContentBlock</code>	80, 80
<code>\markdownRendererContentBlockCode</code>	81
<code>\markdownRendererContentBlockOnlineImage</code>	80
<code>\markdownRendererDisplayMath</code>	107
<code>\markdownRendererDlBegin</code>	81
<code>\markdownRendererDlBeginTight</code>	82
<code>\markdownRendererDlDefinitionBegin</code>	83
<code>\markdownRendererDlDefinitionEnd</code>	83
<code>\markdownRendererDlEnd</code>	83
<code>\markdownRendererDlEndTight</code>	83
<code>\markdownRendererDlItem</code>	82
<code>\markdownRendererDlItemEnd</code>	82

<code>\markdownRendererDocumentBegin</code>	94
<code>\markdownRendererDocumentEnd</code>	94
<code>\markdownRendererEllipsis</code>	39, 84
<code>\markdownRendererEmphasis</code>	84, 115
<code>\markdownRendererFancyOlBegin</code>	96, 97
<code>\markdownRendererFancyOlBeginTight</code>	97
<code>\markdownRendererFancyOlEnd</code>	100
<code>\markdownRendererFancyOlEndTight</code>	100
<code>\markdownRendererFancyOlItem</code>	98
<code>\markdownRendererFancyOlItemEnd</code>	98
<code>\markdownRendererFancyOlItemWithNumber</code>	99
<code>\markdownRendererFencedCodeAttributeContextBegin</code>	85
<code>\markdownRendererFencedCodeAttributeContextEnd</code>	85
<code>\markdownRendererFencedDivAttributeContextBegin</code>	86
<code>\markdownRendererFencedDivAttributeContextEnd</code>	86
<code>\markdownRendererHalfTickedBox</code>	108
<code>\markdownRendererHardLineBreak</code>	92
<code>\markdownRendererHeaderAttributeContextBegin</code>	86
<code>\markdownRendererHeaderAttributeContextEnd</code>	86
<code>\markdownRendererHeadingFive</code>	88
<code>\markdownRendererHeadingFour</code>	88
<code>\markdownRendererHeadingOne</code>	87
<code>\markdownRendererHeadingSix</code>	88
<code>\markdownRendererHeadingThree</code>	87
<code>\markdownRendererHeadingTwo</code>	87
<code>\markdownRendererImage</code>	90
<code>\markdownRendererImageAttributeContextBegin</code>	90
<code>\markdownRendererImageAttributeContextEnd</code>	90
<code>\markdownRendererInlineHtmlComment</code>	89
<code>\markdownRendererInlineHtmlTag</code>	89
<code>\markdownRendererInlineMath</code>	107
<code>\markdownRendererInputBlockHtmlElement</code>	89
<code>\markdownRendererInputFencedCode</code>	78
<code>\markdownRendererInputRawBlock</code>	101
<code>\markdownRendererInputRawInline</code>	100
<code>\markdownRendererInputVerbatim</code>	78
<code>\markdownRendererInterblockSeparator</code>	91
<code>\markdownRendererJekyllDataBegin</code>	109
<code>\markdownRendererJekyllDataBoolean</code>	111
<code>\markdownRendererJekyllDataEmpty</code>	112
<code>\markdownRendererJekyllDataEnd</code>	109
<code>\markdownRendererJekyllDataMappingBegin</code>	110

<code>\markdownRendererJekyllDataMappingEnd</code>	110
<code>\markdownRendererJekyllDataNumber</code>	111
<code>\markdownRendererJekyllDataSequenceBegin</code>	110
<code>\markdownRendererJekyllDataSequenceEnd</code>	111
<code>\markdownRendererJekyllDataString</code>	112
<code>\markdownRendererLineBlockBegin</code>	92
<code>\markdownRendererLineBlockEnd</code>	92
<code>\markdownRendererLink</code>	93, 115
<code>\markdownRendererLinkAttributeContextBegin</code>	93
<code>\markdownRendererLinkAttributeContextEnd</code>	93
<code>\markdownRendererMark</code>	94
<code>\markdownRendererNbsp</code>	95
<code>\markdownRendererNote</code>	95
<code>\markdownRendererOlBegin</code>	96
<code>\markdownRendererOlBeginTight</code>	96
<code>\markdownRendererOlEnd</code>	99
<code>\markdownRendererOlEndTight</code>	99
<code>\markdownRendererOlItem</code>	39, 97
<code>\markdownRendererOlItemEnd</code>	97
<code>\markdownRendererOlItemWithNumber</code>	39, 98
<code>\markdownRendererParagraphSeparator</code>	91
<code>\markdownRendererReplacementCharacter</code>	102
<code>\markdownRendererSectionBegin</code>	101
<code>\markdownRendererSectionEnd</code>	101
<code>\markdownRendererSoftLineBreak</code>	92
<code>\markdownRendererStrikeThrough</code>	105
<code>\markdownRendererStrongEmphasis</code>	85
<code>\markdownRendererSubscript</code>	105
<code>\markdownRendererSuperscript</code>	106
<code>\markdownRendererTable</code>	107
<code>\markdownRendererTableAttributeContextBegin</code>	106
<code>\markdownRendererTableAttributeContextEnd</code>	106
<code>\markdownRendererTextCite</code>	78
<code>\markdownRendererThematicBreak</code>	108
<code>\markdownRendererTickedBox</code>	108
<code>\markdownRendererUlBegin</code>	75
<code>\markdownRendererUlBeginTight</code>	76
<code>\markdownRendererUlEnd</code>	77
<code>\markdownRendererUlEndTight</code>	77
<code>\markdownRendererUlItem</code>	76
<code>\markdownRendererUlItemEnd</code>	76
<code>\markdownRendererUntickedBox</code>	108

<code>\markdownSetup</code>	53, 53, 58, 131, 138, 362, 363
<code>\markdownSetupSnippet</code>	66, 66
<code>\markdownWarning</code>	128, 128
<code>\markinline</code>	50, 51, 52, 130, 356, 360
<code>\markinlinePlainTeX</code>	360
<code>new</code>	7, 16, 332
<code>notes</code>	35, 95
<code>parsers</code>	209, 243
<code>parsers.punctuation</code>	211
<code>pipeTables</code>	6, 36, 42, 107
<code>preserveTabs</code>	36, 40, 244
<code>rawAttribute</code>	36, 37, 101
<code>reader</code>	8, 27, 140, 209, 242, 286
<code>reader-&gt;add_special_character</code>	8, 9, 27, 279
<code>reader-&gt;auto_link_email</code>	269
<code>reader-&gt;auto_link_url</code>	269
<code>reader-&gt;create_parser</code>	244
<code>reader-&gt;finalize_grammar</code>	275, 337
<code>reader-&gt;initialize_named_group</code>	280
<code>reader-&gt;insert_pattern</code>	8, 9, 27, 276, 281
<code>reader-&gt;lookup_note_reference</code>	256
<code>reader-&gt;lookup_reference</code>	256
<code>reader-&gt;normalize_tag</code>	243
<code>reader-&gt;options</code>	243
<code>reader-&gt;parser_functions</code>	244
<code>reader-&gt;parser_functions.name</code>	244
<code>reader-&gt;parsers</code>	243, 243
<code>reader-&gt;register_link</code>	256
<code>reader-&gt;update_rule</code>	276, 278, 281
<code>reader-&gt;writer</code>	243
<code>reader.new</code>	242, 243, 337
<code>relativeReferences</code>	37
<code>\setupmarkdown</code>	138
<code>shiftHeadings</code>	6, 38
<code>singletonCache</code>	16
<code>slice</code>	6, 38, 192, 203, 204
<code>smartEllipses</code>	39, 84, 132
<code>\startmarkdown</code>	137, 137, 392
<code>startNumber</code>	39, 97–99

<code>\stopmarkdown</code>	137, 137, 392
<code>strikeThrough</code>	39, 105, 388
<code>stripIndent</code>	40, 244
<code>stripPercentSigns</code>	353
<code>subscripts</code>	40, 105
<code>superscripts</code>	41, 106
<code>syntax</code>	277, 281
<code>tableAttributes</code>	41, 106
<code>tableCaptions</code>	6, 41, 42, 106
<code>taskLists</code>	42, 108, 378
<code>texComments</code>	43, 245
<code>texMathDollars</code>	43, 107
<code>texMathDoubleBackslash</code>	44, 107
<code>texMathSingleBackslash</code>	44, 107
<code>tightLists</code>	44, 76, 77, 82, 84, 96, 97, 99, 100, 368
<code>underscores</code>	45
<code>unicodeNormalization</code>	17, 17
<code>unicodeNormalizationForm</code>	17, 17
<code>util.cache</code>	140, 141
<code>util.cache_verbatim</code>	141
<code>util.encode_json_string</code>	141
<code>util.err</code>	140
<code>util.escaper</code>	143
<code>util.expand_tabs_in_line</code>	141
<code>util.flatten</code>	142
<code>util.intersperse</code>	143
<code>util.map</code>	143
<code>util.pathname</code>	144
<code>util.rope_last</code>	142
<code>util.rope_to_string</code>	142
<code>util.table_copy</code>	141
<code>util.walk</code>	141, 142
<code>walkable_syntax</code>	8, 18, 24, 276, 278–281
<code>writer</code>	140, 140, 191, 286
<code>writer-&gt;active_attributes</code>	202, 202–204
<code>writer-&gt;attribute_type_levels</code>	202
<code>writer-&gt;attributes</code>	200
<code>writer-&gt;block_html_element</code>	199
<code>writer-&gt;blockquote</code>	199
<code>writer-&gt;bulletitem</code>	197

writer->bulletlist	197
writer->citations	287
writer->code	195
writer->contentblock	292
writer->defer_call	209, 209
writer->definitionlist	295
writer->display_math	325
writer->div_begin	307
writer->div_end	307
writer->document	200
writer->ellipsis	194
writer->emphasis	199
writer->escape	195
writer->escape_minimal	195
writer->escape_programmatic_text	195
writer->escape_typographic_text	195
writer->escaped_chars	194, 195
writer->escaped_minimal_strings	194, 195
writer->escaped_strings	194
writer->escaped_uri_chars	194, 195
writer->fancyitem	298
writer->fancylist	297
writer->fencedCode	302
writer->flatten_inlines	191, 191
writer->get_state	209
writer->hard_line_break	193
writer->heading	207
writer->identifier	195
writer->image	196
writer->infostring	195
writer->inline_html_comment	198
writer->inline_html_tag	198
writer->inline_math	325
writer->interblocksep	193
writer->is_writing	192, 192
writer->jekyllData	329
writer->lineblock	313
writer->link	196
writer->mark	314
writer->math	195
writer->nbsp	192
writer->note	316



writer->options	191
writer->ordereditem	198
writer->orderedlist	197
writer->pack	193, 285
writer->paragraph	193
writer->paragraphsep	193
writer->plain	192
writer->pop_attributes	202, 203, 204
writer->push_attributes	202, 203, 204
writer->rawBlock	303
writer->rawInline	322
writer->set_state	209
writer->slice_begin	192
writer->slice_end	192
writer->soft_line_break	193
writer->space	192
writer->span	286
writer->strike_through	323
writer->string	195
writer->strong	199
writer->subscript	324
writer->suffix	192
writer->superscript	324
writer->table	319
writer->thematic_break	194
writer->checkbox	199
writer->undosep	193, 284
writer->uri	195
writer->verbatim	199
writer.new	191, 191, 337