

A Markdown Interpreter for T_EX

Vít Novotný
witiko@mail.muni.cz

Version 2.21.0-0-gee15b88
2023-02-28

Contents

1	Introduction	1	3	Implementation	113
1.1	Requirements	2	3.1	Lua Implementation . . .	113
1.2	Feedback	6	3.2	Plain T _E X Implementation	243
1.3	Acknowledgements	6	3.3	L ^A T _E X Implementation . .	262
2	Interfaces	7	3.4	ConT _E Xt Implementation	289
2.1	Lua Interface	7			
2.2	Plain T _E X Interface	44			
2.3	L ^A T _E X Interface	93			
2.4	ConT _E Xt Interface	109			
				References	296
				Index	297

List of Figures

1	A block diagram of the Markdown package	7
2	A sequence diagram of typesetting a document using the T _E X interface . .	40
3	A sequence diagram of typesetting a document using the Lua CLI	41
4	Various formats of mathematical formulae	100
5	The banner of the Markdown package	101
6	A pushdown automaton that recognizes T _E X comments	178

1 Introduction

The Markdown package¹ converts markdown² markup to T_EX commands. The functionality is provided both as a Lua module and as plain T_EX, L^AT_EX, and ConT_EXt macro packages that can be used to directly typeset T_EX documents containing markdown markup. Unlike other convertors, 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

¹See <https://ctan.org/pkg/markdown>.

²See <https://daringfireball.net/projects/markdown/basics>.

implementation of the package. The technical documentation contains only a limited number of tutorials and code examples. You can find more of these in the user manual.³

```
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 Novotný",
5   copyright = {"2009-2016 John MacFarlane, Hans Hagen",
6               "2016-2023 Vít 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_{TEX} 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_{TEX} \geq 0.72.0 (T_EXLive \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_{TEX} (T_EXLive \geq 2008).

```
13 local unicode
14 (function()
15   local ran_ok
16   ran_ok, unicode = pcall(require, "unicode")
```

If the Selene Unicode library is unavailable and we are using Lua \geq 5.3, we will use the built-in support for Unicode.

```
17   if not ran_ok then
```

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

```

18     unicode = {utf8 = {char=utf8.char}}
19   end
20 end)()

```

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_{TEX} (T_{EX}Live \geq 2008).

```

21 local md5 = require("md5")

```

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 T_{EX} Live \geq 2020.

```

22 local uni_case
23 (function()
24   local ran_ok
25   -- TODO: Stop loading kpse module to a global kpse variable
26   -- after https://github.com/latex3/lua-uni-algos/issues/3 has been fixed.
27   -- Remove kpse global also from file .luacheckrc.
28   ran_ok, kpse = pcall(require, "kpse")
29   if ran_ok then
30     kpse.set_program_name("luatex")
31     ran_ok, uni_case = pcall(require, "lua-uni-case")
32   end

```

If the lua-uni-algos library is unavailable but the Selene Unicode library is available, we will use its Unicode lower-casing support instead of the more proper case-folding.

```

33   if not ran_ok then
34     if unicode.utf8.lower then
35       uni_case = {casefold = unicode.utf8.lower}
36     else

```

If the Selene Unicode library is also unavailable, we will defer to using ASCII lower-casing.

```

37       uni_case = {casefold = string.lower}
38     end
39   end
40 end)()

```

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

1.1.2 Plain T_EX Requirements

The plain T_EX part of the package requires that the plain T_EX 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^AT_EX3 kernel in T_EX 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.

```
41 <@@=markdown>
42 \ifx\ExplSyntaxOn\undefined
43   \input expl3-generic\relax
44 \fi
```

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

The plain T_EX 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 T_EX 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 LuaT_EX (T_EXLive \geq 2008).

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

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

Unless you convert markdown documents to T_EX manually using the Lua command-line interface (see Section 2.1.6), the plain T_EX part of the package will require that either the LuaT_EX `\directlua` primitive or the shell access file stream 18 is available in your T_EX engine. If only the shell access file stream is available in your T_EX engine (as is the case with pdfT_EX and X_YT_EX) or if you enforce the use of shell using the `\markdownMode` macro, then unless your T_EX 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^AT_EX Requirements

The L^AT_EX part of the package requires that the L^AT_EX 2_ε format is loaded,

```
45 \NeedsTeXFormat{LaTeX2e}%
```

a \TeX engine that extends $\varepsilon\text{-}\TeX$, 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 2.2.4 and 3.3.4) or \LaTeX themes (see Section 2.3.2.2) and will not be loaded if the `plain` package option has been enabled (see Section 2.3.2.1):

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` \LaTeX theme (see Section 2.3.2.2).

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` \TeX command that is used in the default renderer prototype for citations. The package is included in \TeX Live \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` \LaTeX theme, see Section 2.3.2.2.

grffile A package that extends the name processing of package `graphics` to support a larger range of file names in $2006 \leq \text{TeX Live} \leq 2019$. Since $\text{TeX Live} \geq 2020$, the functionality of the package has been integrated in the $\LaTeX 2_\varepsilon$ kernel. It is used in the `witiko/dot` and `witiko/graphicx/http` \LaTeX themes, see Section 2.3.2.2.

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.

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.

```
46 \RequirePackage{expl3}
```

1.1.4 ConT_EXt Prerequisites

The ConT_EXt part of the package requires that either the Mark II or the Mark IV format is loaded, all the plain T_EX prerequisites (see Section 1.1.2), and the following ConT_EXt modules:

m-database A module that provides the default token renderer prototype for iA,Writer content blocks with the CSV filename extension (see Section 2.2.4).

1.2 Feedback

Please use the Markdown project page on GitHub⁴ 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_EX- \LaTeX Stack Exchange.⁵ 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_EX implementation of the package draws inspiration from several sources including the source code of $\LaTeX 2_{\epsilon}$, the `minted` package by Geoffrey M. Poore, which likewise tackles the issue of interfacing with an external interpreter from T_EX, the `filecontents` package by Scott Pakin and others.

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

⁵See <https://tex.stackexchange.com>.

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 $\text{T}_{\text{E}}\text{X}$ nor Lua provide interfaces as a language construct, the separation to interfaces and implementations is a *gentlemen's agreement*. It serves as a means of structuring this documentation and as a promise to the user that if they only access the package through the interface, the future minor versions of the package should remain backwards compatible.

Figure 1 shows the high-level structure of the Markdown package: The translation from markdown to $\text{T}_{\text{E}}\text{X}$ *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 $\text{C}^{\text{o}}\text{nT}_{\text{E}}\text{Xt}$ layers provide syntactic sugar on top of plain $\text{T}_{\text{E}}\text{X}$ macros. The user can interface with any and all layers.

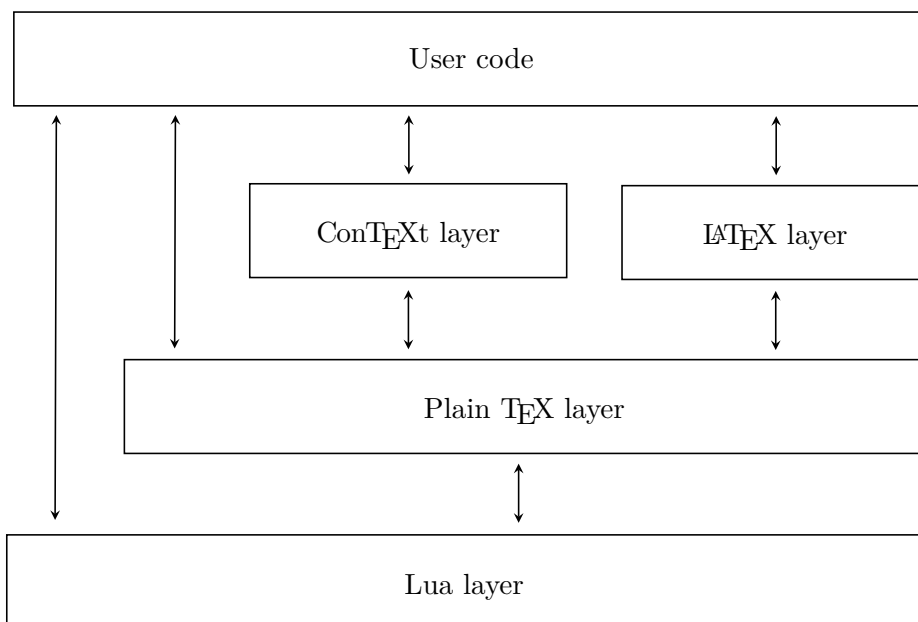


Figure 1: A block diagram of the Markdown package

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.

```
47 local M = {metadata = metadata}
```

2.1.1 Conversion from Markdown to Plain T_EX

The Lua interface exposes the `new(options)` function. This function returns a conversion function from markdown to plain T_EX 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 T_EX output using the default options and prints the T_EX 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.

```
48 local walkable_syntax = {
49   Block = {
50     "Blockquote",
51     "Verbatim",
52     "ThematicBreak",
53     "BulletList",
54     "OrderedList",
55     "Heading",
56     "DisplayHtml",
57     "Paragraph",
58     "Plain",
59   },
60   Inline = {
61     "Str",
62     "Space",
63     "Endline",
64     "U1OrStarLine",
65     "Strong",
```



```

66     "Emph",
67     "Link",
68     "Image",
69     "Code",
70     "AutoLinkUrl",
71     "AutoLinkEmail",
72     "AutoLinkRelativeReference",
73     "InlineHtml",
74     "HtmlEntity",
75     "EscapedChar",
76     "Smart",
77     "Symbol",
78   },
79 }

```

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 -> Emph` and `Inline -> Link` rules, we would call `reader->insert_pattern` with `"Inline after Emph"` (or `"Inline before Link"`) 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.

```
80 local defaultOptions = {}
```

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

```
81 \ExplSyntaxOn
82 \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.

```
83 \prop_new:N \g_@@_lua_option_types_prop
84 \prop_new:N \g_@@_default_lua_options_prop
```

```

85 \seq_new:N \g_@@_option_layers_seq
86 \tl_const:Nn \c_@@_option_layer_lua_tl { lua }
87 \seq_gput_right:NV \g_@@_option_layers_seq \c_@@_option_layer_lua_tl
88 \cs_new:Nn
89   \@@_add_lua_option:nnn
90   {
91     \@@_add_option:Vnnn
92     \c_@@_option_layer_lua_tl
93     { #1 }
94     { #2 }
95     { #3 }
96   }
97 \cs_new:Nn
98   \@@_add_option:nnnn
99   {
100    \seq_gput_right:cn
101    { g_@@_ #1 _options_seq }
102    { #2 }
103    \prop_gput:cnn
104    { g_@@_ #1 _option_types_prop }
105    { #2 }
106    { #3 }
107    \prop_gput:cnn
108    { g_@@_default_ #1 _options_prop }
109    { #2 }
110    { #4 }
111    \@@_typecheck_option:n
112    { #2 }
113  }
114 \cs_generate_variant:Nn
115   \@@_add_option:nnnn
116   { Vnnn }
117 \tl_const:Nn \c_@@_option_value_true_tl { true }
118 \tl_const:Nn \c_@@_option_value_false_tl { false }
119 \cs_new:Nn \@@_typecheck_option:n
120   {
121     \@@_get_option_type:nN
122     { #1 }
123     \l_tmpa_tl
124     \str_case_e:Vn
125     \l_tmpa_tl
126     {
127       { \c_@@_option_type_boolean_tl }
128       {
129         \@@_get_option_value:nN
130         { #1 }
131         \l_tmpa_tl

```

```

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

```

```

179 \cs_new:Nn
180 \@@_get_option_type:nN
181 {
182   \bool_set_false:N
183     \l_tmpa_bool
184   \seq_map_inline:Nn
185     \g_@@_option_layers_seq
186     {
187       \prop_get:cnNT
188         { g_@@_ ##1 _option_types_prop }
189         { #1 }
190       \l_tmpa_tl
191       {
192         \bool_set_true:N
193           \l_tmpa_bool
194         \seq_map_break:
195       }
196     }
197   \bool_if:nF
198     \l_tmpa_bool
199   {
200     \msg_error:nnn
201       { @@ }
202       { undefined-option }
203       { #1 }
204   }
205   \seq_if_in:NVF
206     \g_@@_option_types_seq
207     \l_tmpa_tl
208   {
209     \msg_error:nnnV
210       { @@ }
211       { unknown-option-type }
212       { #1 }
213     \l_tmpa_tl
214   }
215   \tl_set_eq:NN
216     #2
217     \l_tmpa_tl
218 }
219 \msg_new:nnn
220 { @@ }
221 { unknown-option-type }
222 {
223   Option~#1~has~unknown~type~#2.
224 }
225 \msg_new:nnn

```

```

226 { @@ }
227 { undefined-option }
228 {
229   Option~#1~is~undefined.
230 }
231 \cs_new:Nn
232 \@@_get_default_option_value:nN
233 {
234   \bool_set_false:N
235     \l_tmpa_bool
236   \seq_map_inline:Nn
237     \g_@@_option_layers_seq
238     {
239       \prop_get:cnNT
240         { g_@@_default_ ##1 _options_prop }
241         { #1 }
242         #2
243         {
244           \bool_set_true:N
245             \l_tmpa_bool
246           \seq_map_break:
247         }
248       }
249   \bool_if:nF
250     \l_tmpa_bool
251     {
252       \msg_error:nnn
253         { @@ }
254         { undefined-option }
255         { #1 }
256     }
257 }
258 \cs_new:Nn
259 \@@_get_option_value:nN
260 {
261   \@@_option_tl_to_csname:nN
262     { #1 }
263   \l_tmpa_tl
264   \cs_if_free:cTF
265     { \l_tmpa_tl }
266     {
267       \@@_get_default_option_value:nN
268         { #1 }
269       #2
270     }
271   {
272     \@@_get_option_type:nN

```

```

273     { #1 }
274     \l_tmpa_tl
275     \str_if_eq:NNTF
276     \c_@@_option_type_counter_tl
277     \l_tmpa_tl
278     {
279     \@@_option_tl_to_csname:nN
280     { #1 }
281     \l_tmpa_tl
282     \tl_set:Nx
283     #2
284     { \the \cs:w \l_tmpa_tl \cs_end: }
285     }
286     {
287     \@@_option_tl_to_csname:nN
288     { #1 }
289     \l_tmpa_tl
290     \tl_set:Nv
291     #2
292     { \l_tmpa_tl }
293     }
294     }
295     }
296     \cs_new:Nn \@@_option_tl_to_csname:nN
297     {
298     \tl_set:Nn
299     \l_tmpa_tl
300     { \str_uppercase:n { #1 } }
301     \tl_set:Nx
302     #2
303     {
304     markdownOption
305     \tl_head:f { \l_tmpa_tl }
306     \tl_tail:n { #1 }
307     }
308     }
309     \seq_new:N \g_@@_cases_seq
310     \cs_new:Nn \@@_with_various_cases:nn
311     {
312     \seq_clear:N
313     \l_tmpa_seq
314     \seq_map_inline:Nn
315     \g_@@_cases_seq
316     {
317     \tl_set:Nn
318     \l_tmpa_tl
319     { #1 }

```

```

320     \use:c { ##1 }
321     \l_tmpa_tl
322     \seq_put_right:NV
323     \l_tmpa_seq
324     \l_tmpa_tl
325   }
326   \seq_map_inline:Nn
327     \l_tmpa_seq
328     { #2 }
329 }
330 \cs_new:Nn \@@_camel_case:N
331 {
332   \regex_replace_all:nnN
333     { _ ([a-z]) }
334     { \c { str_uppercase:n } \cB\{ \1 \cE\} }
335     #1
336   \tl_set:Nx
337     #1
338     { #1 }
339 }
340 \seq_gput_right:Nn \g_@@_cases_seq { @@_camel_case:N }
341 \cs_new:Nn \@@_snake_case:N
342 {
343   \regex_replace_all:nnN
344     { ([a-z])([A-Z]) }
345     { \1 _ \c { str_lowercase:n } \cB\{ \2 \cE\} }
346     #1
347   \tl_set:Nx
348     #1
349     { #1 }
350 }
351 \seq_gput_right:Nn \g_@@_cases_seq { @@_snake_case:N }

```

2.1.4 File and Directory Names

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

A path to the directory containing auxiliary cache files. If the last segment of the path does not exist, it will be created by the Lua command-line and plain T_EX 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.

```

352 \@@_add_lua_option:nnn
353   { cacheDir }
354   { path }
355   { \markdownOptionOutputDir / _markdown_ \jobname }
356 defaultOptions.cacheDir = "."

```

`contentBlocksLanguageMap`= $\langle filename \rangle$
 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 2.2.3.7 for more information.

```

357 \@@_add_lua_option:nnn
358   { contentBlocksLanguageMap }
359   { path }
360   { markdown-languages.json }
361 defaultOptions.contentBlocksLanguageMap = "markdown-languages.json"

```

`debugExtensionsFileName`= $\langle filename \rangle$ default: `debug-extensions.json`

The filename of the JSON file that will be produced when the `debugExtensions` option is enabled. This file will contain the extensible subset of the PEG grammar of markdown (see the `walkable_syntax` hash table) after built-in syntax extensions (see Section 3.1.6) and user-defined syntax extensions (see Section 2.1.2) have been applied.

```

362 \@@_add_lua_option:nnn
363   { debugExtensionsFileName }
364   { path }
365   { \markdownOptionOutputDir / \jobname .debug-extensions.json }
366 defaultOptions.debugExtensionsFileName = "debug-extensions.json"

```

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

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

The frozen cache makes it possible to later typeset a plain $\text{T}_{\text{E}}\text{X}$ 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.


```

367 \@@_add_lua_option:nnn
368   { frozenCacheFileName }
369   { path }
370   { \markdownOptionCacheDir / frozenCache.tex }
371 defaultOptions.frozenCacheFileName = "frozenCache.tex"

```

2.1.5 Parser Options

`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.

```

372 \@@_add_lua_option:nnn
373   { blankBeforeBlockquote }
374   { boolean }
375   { false }
376 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.

```

377 \@@_add_lua_option:nnn
378   { blankBeforeCodeFence }
379   { boolean }
380   { false }
381 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.

```

382 \@@_add_lua_option:nnn
383   { blankBeforeDivFence }
384   { boolean }
385   { false }
386 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.

```
387 \@@_add_lua_option:nnn
388 { blankBeforeHeading }
389 { boolean }
390 { false }

391 defaultOptions.blankBeforeHeading = false
```

`bracketedSpans=true, false` default: false

- `true` Enable the Pandoc bracketed spans extension:

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

- `false` Disable the Pandoc bracketed spans extension:

```
392 \@@_add_lua_option:nnn
393 { bracketedSpans }
394 { boolean }
395 { false }

396 defaultOptions.bracketedSpans = false
```

`breakableBlockquotes=true, false` default: false

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

```
397 \@@_add_lua_option:nnn
398 { breakableBlockquotes }
399 { boolean }
400 { false }

401 defaultOptions.breakableBlockquotes = false
```

`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.

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

406 defaultOptions.citationNbsps = true
```

`citations=true, false`

default: `false`

`true` Enable the Pandoc citation syntax extension:

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.

```
407 \@@_add_lua_option:nnn
408 { citations }
409 { boolean }
410 { false }

411 defaultOptions.citations = false
```

`codeSpans=true, false`

default: true

true Enable the code span syntax:

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

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

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

```
412 \@@_add_lua_option:nnn  
413 { codeSpans }  
414 { boolean }  
415 { true }  
  
416 defaultOptions.codeSpans = true
```

`contentBlocks=true, false`

default: false

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

```
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.

```
417 \@@_add_lua_option:nnn  
418 { contentBlocks }  
419 { boolean }  
420 { false }  
  
421 defaultOptions.contentBlocks = false
```

`debugExtensions=true, false`

default: `false`

- true** Produce a JSON file that will contain the extensible subset of the PEG grammar of markdown (see the `walkable_syntax` hash table) after built-in syntax extensions (see Section 3.1.6) and user-defined syntax extensions (see Section 2.1.2) have been applied. This helps you to see how the different extensions interact. The name of the produced JSON file is controlled by the `debugExtensionsFileName` option.
- false** Do not produce a JSON file with the PEG grammar of markdown.

```
422 \@@_add_lua_option:nnn
423   { debugExtensions }
424   { boolean }
425   { false }

426 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.

```
427 \@@_add_lua_option:nnn
428   { definitionLists }
429   { boolean }
430   { false }

431 defaultOptions.definitionLists = false
```

`eagerCache=true, false`

default: true

true Converted markdown documents will be cached in `cacheDir`. This can be useful for post-processing the converted documents and for recovering historical versions of the documents from the cache. 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. Recursive nesting of markdown document fragments is undefined behavior when `eagerCache` is disabled.

```
432 \@@_add_lua_option:nnn
433   { eagerCache }
434   { boolean }
435   { true }
436 defaultOptions.eagerCache = true
```

`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}

```

```

437 \@@_add_lua_option:nnn
438   { expectJekyllData }
439   { boolean }
440   { false }
441 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 = 2,
  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 Emph", 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:

```

442 metadata.user_extension_api_version = 2
443 metadata.grammar_version = 2

```

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 `\luamref{reader}` object, such as the `\luamref{reader->insert_pattern}` and


```
\luamref{reader->add_special_character} methods,  
see Section <#luauserextensions>.
```

```
444 \cs_generate_variant:Nn  
445   \@@_add_lua_option:nnn  
446   { nnV }  
447 \@@_add_lua_option:nnV  
448   { extensions }  
449   { clist }  
450   \c_empty_clist  
451 defaultOptions.extensions = {}
```

`fancyLists=true, false`

default: false

`true` Enable the Pandoc fancy list extension:

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

`false` Disable the Pandoc fancy list extension.

```
452 \@@_add_lua_option:nnn  
453   { fancyLists }  
454   { boolean }  
455   { false }  
456 defaultOptions.fancyLists = false
```

`fencedCode=true, false`

default: false

`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.

```
457 \@@_add_lua_option:nnn
458 { fencedCode }
459 { boolean }
460 { false }

461 defaultOptions.fencedCode = false
```

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

**true** Enable the Pandoc fenced code attribute extension:

```
~~~~ {#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 extension.

```
462 \@@_add_lua_option:nnn
463 { fencedCodeAttributes }
464 { boolean }
465 { false }

466 defaultOptions.fencedCodeAttributes = false
```

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

**true** Enable the Pandoc fenced divs extension:

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

And another.
:::
```

**false** Disable the Pandoc fenced divs extension:

```

467 \@@_add_lua_option:nnn
468 { fencedDivs }
469 { boolean }
470 { false }

471 defaultOptions.fencedDivs = false

```

`finalizeCache=true, false`

default: false

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

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

```

472 \@@_add_lua_option:nnn
473 { finalizeCache }
474 { boolean }
475 { false }

476 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  $\TeX$  macro `\markdownFrozenCache<number>` that will typeset markdown document number `<number>`.

```

477 \@@_add_lua_option:nnn
478 { frozenCacheCounter }
479 { counter }
480 { 0 }

481 defaultOptions.frozenCacheCounter = 0

```

`hardLineBreaks=true, false` default: `false`

`true` Interpret all newlines within a paragraph as hard line breaks instead of spaces.

`false` Interpret all newlines within a paragraph as spaces.

```
482 \@@_add_lua_option:nnn
483 { hardLineBreaks }
484 { boolean }
485 { false }
```

The `hardLineBreaks` option has been deprecated and will be removed in Markdown 3.0.0. From then on, all line breaks within a paragraph will be interpreted as soft line breaks.

```
486 defaultOptions.hardLineBreaks = 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.

```
487 \@@_add_lua_option:nnn
488 { hashEnumerators }
489 { boolean }
490 { false }

491 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.

```

492 \@@_add_lua_option:nnn
493 { headerAttributes }
494 { boolean }
495 { false }

496 defaultOptions.headerAttributes = false

```

`html=true, false` default: false

- 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.

```

497 \@@_add_lua_option:nnn
498 { html }
499 { boolean }
500 { false }

501 defaultOptions.html = false

```

`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.

```

502 \@@_add_lua_option:nnn
503 { hybrid }
504 { boolean }
505 { false }

506 defaultOptions.hybrid = false

```

`inlineNotes=true, false`

default: false

`true` Enable the Pandoc inline note syntax extension:

```
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.

The `inlineFootnotes` option has been deprecated and will be removed in Markdown 3.0.0.

```
507 \@@_add_lua_option:nnn
508 { inlineFootnotes }
509 { boolean }
510 { false }
511 \@@_add_lua_option:nnn
512 { inlineNotes }
513 { boolean }
514 { false }

515 defaultOptions.inlineFootnotes = false
516 defaultOptions.inlineNotes = false
```

`jeekyllData=true, false`

default: false

`true` Enable the Pandoc `yaml_metadata_block` syntax extension for entering metadata in YAML:

```

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

 It consists of two paragraphs.

```

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

```

517 \@@_add_lua_option:nnn
518 { jekyllData }
519 { boolean }
520 { false }

521 defaultOptions.jekyllData = false

```

`lineBlocks=true, false`

default: false

**true** Enable the Pandoc line block syntax extension.

```

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

```

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

```

522 \@@_add_lua_option:nnn
523 { lineBlocks }
524 { boolean }
525 { false }

526 defaultOptions.lineBlocks = false

```

`notes=true, false`

default: false

**true** Enable the Pandoc note syntax extension:

```

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.

The footnotes option has been deprecated and will be removed in Markdown 3.0.0.

```
527 \@@_add_lua_option:nnn
528 { footnotes }
529 { boolean }
530 { false }
531 \@@_add_lua_option:nnn
532 { notes }
533 { boolean }
534 { false }

535 defaultOptions.footnotes = false
536 defaultOptions.notes = false
```

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

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

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

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

```
537 \@@_add_lua_option:nnn
538 { pipeTables }
539 { boolean }
540 { false }

541 defaultOptions.pipeTables = false
```

**preserveTabs=true, false** default: false

**true**      Preserve tabs in code block and fenced code blocks.

**false**      Convert any tabs in the input to spaces.

```
542 \@@_add_lua_option:nnn
543 { preserveTabs }
544 { boolean }
545 { false }

546 defaultOptions.preserveTabs = false
```



`rawAttribute=true, false`

default: false

`true` Enable the Pandoc raw attribute syntax extension:

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

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

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

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

`false` Disable the Pandoc raw attribute syntax extension.

```
547 \@@_add_lua_option:nnn
548   { rawAttribute }
549   { boolean }
550   { false }
551 defaultOptions.rawAttribute = true
```

`relativeReferences=true, false`

default: false

`true` Enable relative references⁶ 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!
```

⁶See <https://datatracker.ietf.org/doc/html/rfc3986#section-4.2>.

`false` Disable relative references in autolinks.

```
552 \@@_add_lua_option:nnn
553   { relativeReferences }
554   { boolean }
555   { false }

556 defaultOptions.relativeReferences = false
```

`shiftHeadings=<shift amount>` default: 0

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

```
557 \@@_add_lua_option:nnn
558   { shiftHeadings }
559   { number }
560   { 0 }

561 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.

```
562 \@@_add_lua_option:nnn
563   { slice }
564   { slice }
565   { ^~$ }

566 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.

```
567 \@@_add_lua_option:nnn
568 { smartEllipses }
569 { boolean }
570 { false }

571 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.

```
572 \@@_add_lua_option:nnn
573 { startNumber }
574 { boolean }
575 { true }

576 defaultOptions.startNumber = true
```

`strikeThrough=true, false` default: false

`true` Enable the Pandoc strike-through syntax extension:

This ~~is deleted text.~~

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

```
577 \@@_add_lua_option:nnn
578 { strikeThrough }
579 { boolean }
580 { false }

581 defaultOptions.strikeThrough = false
```

`stripIndent=true, false`

default: false

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

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

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

```
582 \@@_add_lua_option:nnn
583   { stripIndent }
584   { boolean }
585   { false }
586 defaultOptions.stripIndent = false
```

`subscripts=true, false`

default: false

true Enable the Pandoc subscript syntax extension:

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

false Disable the Pandoc subscript syntax extension.

```
587 \@@_add_lua_option:nnn
588   { subscripts }
589   { boolean }
590   { false }
591 defaultOptions.subscripts = false
```

`superscripts=true, false`

default: false

true Enable the Pandoc superscript syntax extension:

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

false Disable the Pandoc superscript syntax extension.

```

592 \@@_add_lua_option:nnn
593   { superscripts }
594   { boolean }
595   { false }

596 defaultOptions.superscripts = false

```

`tableCaptions=true, false`

default: false

true Enable the Pandoc `table_captions` syntax extension for pipe tables (see the `pipeTables` option).

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

: Demonstration of pipe table syntax.

false Disable the Pandoc `table_captions` syntax extension.

```

597 \@@_add_lua_option:nnn
598   { tableCaptions }
599   { boolean }
600   { false }

601 defaultOptions.tableCaptions = false

```

`taskLists=true, false`

default: false

true Enable the Pandoc `task_lists` syntax extension.

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

false Disable the Pandoc `task_lists` syntax extension.

```

602 \@@_add_lua_option:nnn
603   { taskLists }
604   { boolean }
605   { false }

606 defaultOptions.taskLists = false

```

`texComments=true, false`

default: false

true Strip T_EX-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_EX-style comments.

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

`texMathDollars=true, false`

default: false

true Enable the Pandoc `tex_math_dollars` syntax extension.

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

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

false Disable the Pandoc `tex_math_dollars` syntax extension.

```
612 \@@_add_lua_option:nnn
613   { texMathDollars }
614   { boolean }
615   { false }
616 defaultOptions.texMathDollars = 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:

```
- 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.

```
617 \@@_add_lua_option:nnn
618 { tightLists }
619 { boolean }
620 { true }
621 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.

```
622 \@@_add_lua_option:nnn
623 { underscores }
624 { boolean }
625 { true }
626 \ExplSyntaxOff
627 defaultOptions.underscores = true
```

2.1.6 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.

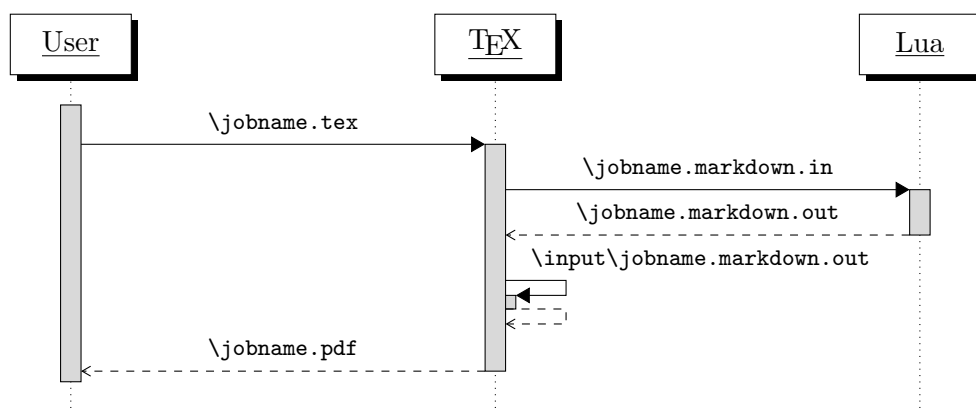


Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the $\text{T}_{\text{E}}\text{X}$ interface

```
628
629 local HELP_STRING = [[
630 Usage: texlua ]] .. arg[0] .. [[ [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
631 where OPTIONS are documented in the Lua interface section of the
632 technical Markdown package documentation.
633
634 When OUTPUT_FILE is unspecified, the result of the conversion will be
635 written to the standard output. When INPUT_FILE is also unspecified, the
636 result of the conversion will be read from the standard input.
637
638 Report bugs to: witiko@mail.muni.cz
639 Markdown package home page: <https://github.com/witiko/markdown>]]
640
```

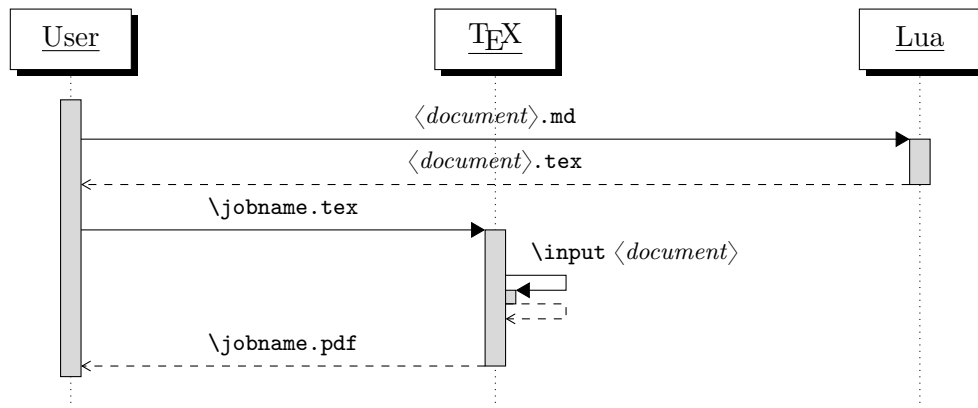



Figure 3: A sequence diagram of the Markdown package typesetting a markdown document using the Lua command-line interface

```

641 local VERSION_STRING = [[
642 markdown-cli.lua (Markdown) ]] .. metadata.version .. [[
643
644 Copyright (C) ]] .. table.concat(metadata.copyright,
645                                     "\nCopyright (C) ") .. [[
646
647 License: ]] .. metadata.license
648
649 local function warn(s)
650   io.stderr:write("Warning: " .. s .. "\n") end
651
652 local function error(s)
653   io.stderr:write("Error: " .. s .. "\n")
654   os.exit(1)
655 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.

```

656 local function camel_case(option_name)
657   local cased_option_name = option_name:gsub("_(%l)", function(match)
658     return match:sub(2, 2):upper()
659   end)
660   return cased_option_name
661 end
662
663 local function snake_case(option_name)
664   local cased_option_name = option_name:gsub("%l%u", function(match)
665     return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()
  
```

```

666     end)
667     return cased_option_name
668 end
669
670 local cases = {camel_case, snake_case}
671 local various_case_options = {}
672 for option_name, _ in pairs(defaultOptions) do
673     for _, case in ipairs(cases) do
674         various_case_options[case(option_name)] = option_name
675     end
676 end
677
678 local process_options = true
679 local options = {}
680 local input_filename
681 local output_filename
682 for i = 1, #arg do
683     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.

```

684         if arg[i] == "--" then
685             process_options = false
686             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.

```

687         elseif arg[i]:match("=") then
688             local key, value = arg[i]:match("(.)=(.*)")
689             if defaultOptions[key] == nil and
690                 various_case_options[key] ~= nil then
691                 key = various_case_options[key]
692             end

```

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

```

693         local default_type = type(defaultOptions[key])
694         if default_type == "boolean" then
695             options[key] = (value == "true")
696         elseif default_type == "number" then
697             options[key] = tonumber(value)
698         elseif default_type == "table" then
699             options[key] = {}
700             for item in value:gmatch("[^ ,]+") do
701                 table.insert(options[key], item)

```

```

702     end
703   else
704     if default_type ~= "string" then
705       if default_type == "nil" then
706         warn('Option "' .. key .. '" not recognized.')
707       else
708         warn('Option "' .. key .. '" type not recognized, please file ' ..
709             'a report to the package maintainer.')
710       end
711       warn('Parsing the ' .. 'value "' .. value ..'" of option "' ..
712           key .. '" as a string.')
713     end
714     options[key] = value
715   end
716   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.

```

717     elseif arg[i] == "--help" or arg[i] == "-h" then
718       print(HELP_STRING)
719       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.

```

720     elseif arg[i] == "--version" or arg[i] == "-v" then
721       print(VERSION_STRING)
722       os.exit()
723     end
724   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 \TeX document.

```

725   if input_filename == nil then
726     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 \TeX document that will result from the conversion.

```

727   elseif output_filename == nil then
728     output_filename = arg[i]
729   else
730     error('Unexpected argument: "' .. arg[i] .. "'.')
731   end
732   ::continue::
733 end

```

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

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

to convert the Markdown document `hello.md` to a T_EX document `hello.tex`. After the Markdown package for our T_EX format has been loaded, the converted document can be typeset as follows:

```
\input hello
```

2.2 Plain T_EX Interface

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

```
734 \def\markdownLastModified{((LASTMODIFIED))}%  
735 \def\markdownVersion{((VERSION))}%
```

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

```
\input markdown
```

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

2.2.1 Typesetting Markdown

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

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

```
736 \let\markdownBegin\relax  
737 \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_EX [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_EX 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_EX code showcases the usage of the `\markdownBegin` and `\markdownEnd` macros:

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

The `\markdownInput` macro accepts a single parameter with the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain T_EX.

```
738 \let\markdownInput\relax
```

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

The following example plain T_EX 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 T_EX document and executes the T_EX 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.

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

```
740 \ExplSyntaxOn
741 \seq_new:N \g_@@_plain_tex_options_seq
```

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

```
742 \prop_new:N \g_@@_plain_tex_option_types_prop
743 \prop_new:N \g_@@_default_plain_tex_options_prop
744 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
745 \seq_gput_right:NV \g_@@_option_layers_seq \c_@@_option_layer_plain_tex_tl
746 \cs_new:Nn
747   \@@_add_plain_tex_option:nnn
748   {
749     \@@_add_option:Vnnn
750     \c_@@_option_layer_plain_tex_tl
751     { #1 }
752     { #2 }
753     { #3 }
754   }
```

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 TeX document and their auxiliary files cached in the `cacheDir` directory.

The `\markdownOptionFrozenCache` option uses the mapping previously created by the `finalizeCache` option, and uses it to typeset the plain TeX document without invoking Lua. 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. It defaults to `false`.

```
755 \@@_add_plain_tex_option:nnn
756   { frozenCache }
757   { boolean }
```

```
758 { 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.
3. Typeset the plain T_EX document to populate and finalize the cache.
4. Enable the `frozenCache` option.
5. Publish the source code of the plain T_EX document and the `cacheDir` directory.

2.2.2.2 File and Directory Names The `\markdownOptionHelperScriptFileName` macro sets the filename of the helper Lua script file that is created during the conversion from markdown to plain T_EX in T_EX engines without the `\directlua` primitive. It defaults to `\jobname.markdown.lua`, where `\jobname` is the base name of the document being typeset.

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

```
759 \@@_add_plain_tex_option:nnn
760 { helperScriptFileName }
761 { path }
762 { \jobname.markdown.lua }
```

The `helperScriptFileName` macro has been deprecated and will be removed in Markdown 3.0.0. To control the filename of the helper Lua script file, use the `\g_luabridge_helper_script_filename_str` macro from the `lt3luabridge` package.

```
763 \str_new:N
764 \g_luabridge_helper_script_filename_str
765 \tl_gset:Nn
766 \g_luabridge_helper_script_filename_str
767 { \markdownOptionHelperScriptFileName }
```

The `\markdownOptionInputTempFileName` macro sets the filename of the temporary input file that is created during the buffering of markdown text from a T_EX source. It defaults to `\jobname.markdown.in`. The same limitations as in the case of the `helperScriptFileName` macro apply here.

```
768 \@@_add_plain_tex_option:nnn
769 { inputTempFileName }
770 { path }
771 { \jobname.markdown.in }
```

The `\markdownOptionOutputTempFileName` macro sets the filename of the temporary output file that is created during the conversion from markdown to plain T_EX in `\markdownMode` other than 2. It defaults to `\jobname.markdown.out`. The same limitations apply here as in the case of the `helperScriptFileName` macro.

```
772 \@@_add_plain_tex_option:nnn
773 { outputTempFileName }
```

```

774 { path }
775 { \jobname.markdown.out }

```

The `outputTempFileName` macro has been deprecated and will be removed in Markdown 3.0.0.

```

776 \str_new:N
777 \g_luabridge_standard_output_filename_str
778 \tl_gset:Nn
779 \g_luabridge_standard_output_filename_str
780 { \markdownOptionOutputTempFileName }

```

The `\markdownOptionErrorTempFileName` macro sets the filename of the temporary output file that is created when a Lua error is encountered during the conversion from markdown to plain T_EX in `\markdownMode` other than 2. It defaults to `\jobname.markdown.err`. The same limitations apply here as in the case of the `helperScriptFileName` macro.

```

781 \@@_add_plain_tex_option:nnn
782 { errorTempFileName }
783 { path }
784 { \jobname.markdown.err }

```

The `errorTempFileName` macro has been deprecated and will be removed in Markdown 3.0.0. To control the filename of the temporary file for Lua errors, use the `\g_luabridge_error_output_filename_str` macro from the `lt3luabridge` package.

```

785 \str_new:N
786 \g_luabridge_error_output_filename_str
787 \tl_gset:Nn
788 \g_luabridge_error_output_filename_str
789 { \markdownOptionErrorTempFileName }

```

The `\markdownOptionOutputDir` macro sets the path to the directory that will contain the auxiliary cache files produced by the Lua implementation and also the auxiliary files produced by the plain T_EX implementation. The option defaults to `..`.

The path must be set to the same value as the `-output-directory` option of your T_EX 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 `helperScriptFileName` macro.

```

790 \@@_add_plain_tex_option:nnn
791 { outputDir }
792 { path }
793 { . }

```

Here, we automatically define plain T_EX macros for the above plain T_EX options.

Furthermore, we also define macros 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_EX implementation, only passed along to Lua.

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 `helperScriptFileName` macro.

```

794 \cs_new:Nn \@@_plain_tex_define_option_commands:
795   {
796     \seq_map_inline:Nn
797       \g_@@_option_layers_seq
798       {
799         \seq_map_inline:cn
800           { g_@@_ ##1 _options_seq }
801           {
802             \@@_plain_tex_define_option_command:n
803               { ####1 }
804           }
805       }
806   }
807 \cs_new:Nn \@@_plain_tex_define_option_command:n
808   {
809     \@@_get_default_option_value:nN
810       { #1 }
811     \l_tmpa_tl
812     \@@_set_option_value:nV
813       { #1 }
814     \l_tmpa_tl
815   }
816 \cs_new:Nn
817   \@@_set_option_value:nn
818   {
819     \@@_define_option:n
820       { #1 }
821     \@@_get_option_type:nN
822       { #1 }
823     \l_tmpa_tl
824     \str_if_eq:NNTF
825       \c_@@_option_type_counter_tl
826       \l_tmpa_tl
827     {
828       \@@_option_tl_to_csname:nN
829         { #1 }
830       \l_tmpa_tl
831       \int_gset:cn
832         { \l_tmpa_tl }
833         { #2 }
834     }
835     {
836       \@@_option_tl_to_csname:nN
837         { #1 }

```

```

838         \l_tmpa_tl
839     \cs_set:cpn
840     { \l_tmpa_tl }
841     { #2 }
842 }
843 }
844 \cs_generate_variant:Nn
845 \@@_set_option_value:nn
846 { nV }
847 \cs_new:Nn
848 \@@_define_option:n
849 {
850     \@@_option_tl_to_csname:nN
851     { #1 }
852     \l_tmpa_tl
853     \cs_if_free:cT
854     { \l_tmpa_tl }
855     {
856         \@@_get_option_type:nN
857         { #1 }
858         \l_tmpb_tl
859         \str_if_eq:NNT
860         \c_@@_option_type_counter_tl
861         \l_tmpb_tl
862         {
863             \@@_option_tl_to_csname:nN
864             { #1 }
865             \l_tmpa_tl
866             \int_new:c
867             { \l_tmpa_tl }
868         }
869     }
870 }
871 \@@_plain_tex_define_option_commands:

```

2.2.2.3 Miscellaneous Options The `\markdownOptionStripPercentSigns` macro controls whether a percent sign (%) at the beginning of a line will be discarded when buffering Markdown input (see Section 3.2.4) or not. Notably, this enables the use of markdown when writing T_EX package documentation using the Doc L^AT_EX package [7] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```

872 \seq_gput_right:Nn
873 \g_@@_plain_tex_options_seq
874 { stripPercentSigns }
875 \prop_gput:Nnn
876 \g_@@_plain_tex_option_types_prop

```

```

877 { stripPercentSigns }
878 { boolean }
879 \prop_gput:Nnx
880 \g_@@_default_plain_tex_options_prop
881 { stripPercentSigns }
882 { false }
883 \ExplSyntaxOff

```

2.2.3 Token Renderers

The following T_EX 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 2.2.4).

To enable the enumeration of token renderers, we will maintain the `\g_@@_renderers_seq` sequence.

```

884 \ExplSyntaxOn
885 \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.

```

886 \prop_new:N \g_@@_renderer_arities_prop
887 \ExplSyntaxOff

```

2.2.3.1 Attribute Renderers The following macros are only produced, when the `headerAttributes` option is enabled.

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

`\markdownRendererAttributeClassName` represents the $\langle class name \rangle$ of a markdown element (`class="⟨class name⟩ ..."` in HTML and `.⟨class name⟩` in Markdown's `headerAttributes` syntax extension). 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.

```

888 \def\markdownRendererAttributeIdentifier{%
889 \markdownRendererAttributeIdentifierPrototype}%
890 \ExplSyntaxOn
891 \seq_gput_right:Nn
892 \g_@@_renderers_seq
893 { attributeIdentifier }
894 \prop_gput:Nnn

```

```

895 \g_@@_renderer_arities_prop
896 { attributeIdentifier }
897 { 1 }
898 \ExplSyntaxOff
899 \def\markdownRendererAttributeName{%
900 \markdownRendererAttributeNamePrototype}%
901 \ExplSyntaxOn
902 \seq_gput_right:Nn
903 \g_@@_renderers_seq
904 { attributeName }
905 \prop_gput:Nnn
906 \g_@@_renderer_arities_prop
907 { attributeName }
908 { 1 }
909 \ExplSyntaxOff
910 \def\markdownRendererAttributeKeyValue{%
911 \markdownRendererAttributeKeyValuePrototype}%
912 \ExplSyntaxOn
913 \seq_gput_right:Nn
914 \g_@@_renderers_seq
915 { attributeKeyValue }
916 \prop_gput:Nnn
917 \g_@@_renderer_arities_prop
918 { attributeKeyValue }
919 { 2 }
920 \ExplSyntaxOff

```

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

```

921 \def\markdownRendererBlockQuoteBegin{%
922 \markdownRendererBlockQuoteBeginPrototype}%
923 \ExplSyntaxOn
924 \seq_gput_right:Nn
925 \g_@@_renderers_seq
926 { blockQuoteBegin }
927 \prop_gput:Nnn
928 \g_@@_renderer_arities_prop
929 { blockQuoteBegin }
930 { 0 }
931 \ExplSyntaxOff

```

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

```

932 \def\markdownRendererBlockQuoteEnd{%
933 \markdownRendererBlockQuoteEndPrototype}%
934 \ExplSyntaxOn

```

```

935 \seq_gput_right:Nn
936   \g_@@_renderers_seq
937   { blockQuoteEnd }
938 \prop_gput:Nnn
939   \g_@@_renderer_arities_prop
940   { blockQuoteEnd }
941   { 0 }
942 \ExplSyntaxOff

```

2.2.3.3 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 an inline bracketed span in which the attributes of the span apply. The macros receive no arguments.

```

943 \def\markdownRendererBracketedSpanAttributeContextBegin{%
944   \markdownRendererBracketedSpanAttributeContextBeginPrototype}%
945 \ExplSyntaxOn
946 \seq_gput_right:Nn
947   \g_@@_renderers_seq
948   { bracketedSpanAttributeContextBegin }
949 \prop_gput:Nnn
950   \g_@@_renderer_arities_prop
951   { bracketedSpanAttributeContextBegin }
952   { 0 }
953 \ExplSyntaxOff
954 \def\markdownRendererBracketedSpanAttributeContextEnd{%
955   \markdownRendererBracketedSpanAttributeContextEndPrototype}%
956 \ExplSyntaxOn
957 \seq_gput_right:Nn
958   \g_@@_renderers_seq
959   { bracketedSpanAttributeContextEnd }
960 \prop_gput:Nnn
961   \g_@@_renderer_arities_prop
962   { bracketedSpanAttributeContextEnd }
963   { 0 }
964 \ExplSyntaxOff

```

2.2.3.4 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.

```

965 \def\markdownRendererUlBegin{%
966   \markdownRendererUlBeginPrototype}%
967 \ExplSyntaxOn
968 \seq_gput_right:Nn
969   \g_@@_renderers_seq

```

```

970 { ulBegin }
971 \prop_gput:Nnn
972 \g_@@_renderer_arities_prop
973 { ulBegin }
974 { 0 }
975 \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.

```

976 \def\markdownRendererUlBeginTight{%
977 \markdownRendererUlBeginTightPrototype}%
978 \ExplSyntaxOn
979 \seq_gput_right:Nn
980 \g_@@_renderers_seq
981 { ulBeginTight }
982 \prop_gput:Nnn
983 \g_@@_renderer_arities_prop
984 { ulBeginTight }
985 { 0 }
986 \ExplSyntaxOff

```

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

```

987 \def\markdownRendererUlItem{%
988 \markdownRendererUlItemPrototype}%
989 \ExplSyntaxOn
990 \seq_gput_right:Nn
991 \g_@@_renderers_seq
992 { ulItem }
993 \prop_gput:Nnn
994 \g_@@_renderer_arities_prop
995 { ulItem }
996 { 0 }
997 \ExplSyntaxOff

```

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

```

998 \def\markdownRendererUlItemEnd{%
999 \markdownRendererUlItemEndPrototype}%
1000 \ExplSyntaxOn
1001 \seq_gput_right:Nn
1002 \g_@@_renderers_seq
1003 { ulItemEnd }
1004 \prop_gput:Nnn

```

```

1005 \g_@@_renderer_arities_prop
1006 { ulItemEnd }
1007 { 0 }
1008 \ExplSyntaxOff

```

The `\markdownRendererUPEnd` 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.

```

1009 \def\markdownRendererUPEnd{%
1010 \markdownRendererUPEndPrototype}%
1011 \ExplSyntaxOn
1012 \seq_gput_right:Nn
1013 \g_@@_renderers_seq
1014 { ulEnd }
1015 \prop_gput:Nnn
1016 \g_@@_renderer_arities_prop
1017 { ulEnd }
1018 { 0 }
1019 \ExplSyntaxOff

```

The `\markdownRendererUPEndTight` 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.

```

1020 \def\markdownRendererUPEndTight{%
1021 \markdownRendererUPEndTightPrototype}%
1022 \ExplSyntaxOn
1023 \seq_gput_right:Nn
1024 \g_@@_renderers_seq
1025 { ulEndTight }
1026 \prop_gput:Nnn
1027 \g_@@_renderer_arities_prop
1028 { ulEndTight }
1029 { 0 }
1030 \ExplSyntaxOff

```

2.2.3.5 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.

```

1031 \def\markdownRendererInputVerbatim{%
1032 \markdownRendererInputVerbatimPrototype}%
1033 \ExplSyntaxOn
1034 \seq_gput_right:Nn
1035 \g_@@_renderers_seq
1036 { inputVerbatim }

```

```

1037 \prop_gput:Nnn
1038   \g_@@_renderer_arities_prop
1039   { inputVerbatim }
1040   { 1 }
1041 \ExplSyntaxOff

```

The `\markdownRendererInputFencedCode` macro represents a fenced code block. This macro will only be produced, when the `fencedCode` option is enabled. The macro receives two arguments that correspond to the filename of a file containing the code block contents and to the code fence infostring.

```

1042 \def\markdownRendererInputFencedCode{%
1043   \markdownRendererInputFencedCodePrototype}%
1044 \ExplSyntaxOn
1045 \seq_gput_right:Nn
1046   \g_@@_renderers_seq
1047   { inputFencedCode }
1048 \prop_gput:Nnn
1049   \g_@@_renderer_arities_prop
1050   { inputFencedCode }
1051   { 2 }
1052 \ExplSyntaxOff

```

2.2.3.6 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.

```

1053 \def\markdownRendererCodeSpan{%
1054   \markdownRendererCodeSpanPrototype}%
1055 \ExplSyntaxOn
1056 \seq_gput_right:Nn
1057   \g_@@_renderers_seq
1058   { codeSpan }
1059 \prop_gput:Nnn
1060   \g_@@_renderer_arities_prop
1061   { codeSpan }
1062   { 1 }
1063 \ExplSyntaxOff

```

2.2.3.7 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.

```

1064 \def\markdownRendererContentBlock{%
1065   \markdownRendererContentBlockPrototype}%

```



```

1066 \ExplSyntaxOn
1067 \seq_gput_right:Nn
1068   \g_@@_renderers_seq
1069   { contentBlock }
1070 \prop_gput:Nnn
1071   \g_@@_renderer_arities_prop
1072   { contentBlock }
1073   { 4 }
1074 \ExplSyntaxOff

```

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

```

1075 \def\markdownRendererContentBlockOnlineImage{%
1076   \markdownRendererContentBlockOnlineImagePrototype}%
1077 \ExplSyntaxOn
1078 \seq_gput_right:Nn
1079   \g_@@_renderers_seq
1080   { contentBlockOnlineImage }
1081 \prop_gput:Nnn
1082   \g_@@_renderer_arities_prop
1083   { contentBlockOnlineImage }
1084   { 4 }
1085 \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`⁷ contains a record (k, v) , then a non-online-image content block with the filename extension $s, s:\text{lower}() = k$ is considered to be in a known programming language v . The macro receives five arguments: the local file name extension s cast to the lower case, the language v , the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the content block.

Note that you will need to place a `markdown-languages.json` file inside your working directory or inside your local T_EX 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.

```

1086 \def\markdownRendererContentBlockCode{%
1087   \markdownRendererContentBlockCodePrototype}%
1088 \ExplSyntaxOn

```

⁷Filenames other than `markdown-languages.json` may be specified using the `contentBlocksLanguageMap` Lua option.

```

1089 \seq_gput_right:Nn
1090   \g_@@_renderers_seq
1091   { contentBlockCode }
1092 \prop_gput:Nnn
1093   \g_@@_renderer_arities_prop
1094   { contentBlockCode }
1095   { 5 }
1096 \ExplSyntaxOff

```

2.2.3.8 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.

```

1097 \def\markdownRendererDlBegin{%
1098   \markdownRendererDlBeginPrototype}%
1099 \ExplSyntaxOn
1100 \seq_gput_right:Nn
1101   \g_@@_renderers_seq
1102   { dlBegin }
1103 \prop_gput:Nnn
1104   \g_@@_renderer_arities_prop
1105   { dlBegin }
1106   { 0 }
1107 \ExplSyntaxOff

```

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

```

1108 \def\markdownRendererDlBeginTight{%
1109   \markdownRendererDlBeginTightPrototype}%
1110 \ExplSyntaxOn
1111 \seq_gput_right:Nn
1112   \g_@@_renderers_seq
1113   { dlBeginTight }
1114 \prop_gput:Nnn
1115   \g_@@_renderer_arities_prop
1116   { dlBeginTight }
1117   { 0 }
1118 \ExplSyntaxOff

```

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

```

1119 \def\markdownRendererDlItem{%

```

```

1120 \markdownRendererDlItemPrototype}%
1121 \ExplSyntaxOn
1122 \seq_gput_right:Nn
1123 \g_@@_renderers_seq
1124 { dlItem }
1125 \prop_gput:Nnn
1126 \g_@@_renderer_arities_prop
1127 { dlItem }
1128 { 1 }
1129 \ExplSyntaxOff

```

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

```

1130 \def\markdownRendererDlItemEnd{%
1131 \markdownRendererDlItemEndPrototype}%
1132 \ExplSyntaxOn
1133 \seq_gput_right:Nn
1134 \g_@@_renderers_seq
1135 { dlItemEnd }
1136 \prop_gput:Nnn
1137 \g_@@_renderer_arities_prop
1138 { dlItemEnd }
1139 { 0 }
1140 \ExplSyntaxOff

```

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

```

1141 \def\markdownRendererDlDefinitionBegin{%
1142 \markdownRendererDlDefinitionBeginPrototype}%
1143 \ExplSyntaxOn
1144 \seq_gput_right:Nn
1145 \g_@@_renderers_seq
1146 { dlDefinitionBegin }
1147 \prop_gput:Nnn
1148 \g_@@_renderer_arities_prop
1149 { dlDefinitionBegin }
1150 { 0 }
1151 \ExplSyntaxOff

```

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

```

1152 \def\markdownRendererDlDefinitionEnd{%
1153 \markdownRendererDlDefinitionEndPrototype}%
1154 \ExplSyntaxOn
1155 \seq_gput_right:Nn
1156 \g_@@_renderers_seq
1157 { dlDefinitionEnd }

```

```

1158 \prop_gput:Nnn
1159   \g_@@_renderer_arities_prop
1160   { dlDefinitionEnd }
1161   { 0 }
1162 \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.

```

1163 \def\markdownRendererDlEnd{%
1164   \markdownRendererDlEndPrototype}%
1165 \ExplSyntaxOn
1166 \seq_gput_right:Nn
1167   \g_@@_renderers_seq
1168   { dlEnd }
1169 \prop_gput:Nnn
1170   \g_@@_renderer_arities_prop
1171   { dlEnd }
1172   { 0 }
1173 \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.

```

1174 \def\markdownRendererDlEndTight{%
1175   \markdownRendererDlEndTightPrototype}%
1176 \ExplSyntaxOn
1177 \seq_gput_right:Nn
1178   \g_@@_renderers_seq
1179   { dlEndTight }
1180 \prop_gput:Nnn
1181   \g_@@_renderer_arities_prop
1182   { dlEndTight }
1183   { 0 }
1184 \ExplSyntaxOff

```

2.2.3.9 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.

```

1185 \def\markdownRendererEllipsis{%
1186   \markdownRendererEllipsisPrototype}%
1187 \ExplSyntaxOn
1188 \seq_gput_right:Nn
1189   \g_@@_renderers_seq

```

```

1190 { ellipsis }
1191 \prop_gput:Nnn
1192 \g_@@_renderer_arities_prop
1193 { ellipsis }
1194 { 0 }
1195 \ExplSyntaxOff

```

2.2.3.10 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.

```

1196 \def\markdownRendererEmphasis{%
1197 \markdownRendererEmphasisPrototype}%
1198 \ExplSyntaxOn
1199 \seq_gput_right:Nn
1200 \g_@@_renderers_seq
1201 { emphasis }
1202 \prop_gput:Nnn
1203 \g_@@_renderer_arities_prop
1204 { emphasis }
1205 { 1 }
1206 \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.

```

1207 \def\markdownRendererStrongEmphasis{%
1208 \markdownRendererStrongEmphasisPrototype}%
1209 \ExplSyntaxOn
1210 \seq_gput_right:Nn
1211 \g_@@_renderers_seq
1212 { strongEmphasis }
1213 \prop_gput:Nnn
1214 \g_@@_renderer_arities_prop
1215 { strongEmphasis }
1216 { 1 }
1217 \ExplSyntaxOff

```

2.2.3.11 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.

```

1218 \def\markdownRendererFencedCodeAttributeContextBegin{%
1219 \markdownRendererFencedCodeAttributeContextBeginPrototype}%

```

```

1220 \ExplSyntaxOn
1221 \seq_gput_right:Nn
1222   \g_@@_renderers_seq
1223   { fencedCodeAttributeContextBegin }
1224 \prop_gput:Nnn
1225   \g_@@_renderer_arities_prop
1226   { fencedCodeAttributeContextBegin }
1227   { 0 }
1228 \ExplSyntaxOff
1229 \def\markdownRendererFencedCodeAttributeContextEnd{%
1230   \markdownRendererFencedCodeAttributeContextEndPrototype}%
1231 \ExplSyntaxOn
1232 \seq_gput_right:Nn
1233   \g_@@_renderers_seq
1234   { fencedCodeAttributeContextEnd }
1235 \prop_gput:Nnn
1236   \g_@@_renderer_arities_prop
1237   { fencedCodeAttributeContextEnd }
1238   { 0 }
1239 \ExplSyntaxOff

```

2.2.3.12 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 div in which the attributes of the div apply. The macros receive no arguments.

```

1240 \def\markdownRendererFencedDivAttributeContextBegin{%
1241   \markdownRendererFencedDivAttributeContextBeginPrototype}%
1242 \ExplSyntaxOn
1243 \seq_gput_right:Nn
1244   \g_@@_renderers_seq
1245   { fencedDivAttributeContextBegin }
1246 \prop_gput:Nnn
1247   \g_@@_renderer_arities_prop
1248   { fencedDivAttributeContextBegin }
1249   { 0 }
1250 \ExplSyntaxOff
1251 \def\markdownRendererFencedDivAttributeContextEnd{%
1252   \markdownRendererFencedDivAttributeContextEndPrototype}%
1253 \ExplSyntaxOn
1254 \seq_gput_right:Nn
1255   \g_@@_renderers_seq
1256   { fencedDivAttributeContextEnd }
1257 \prop_gput:Nnn
1258   \g_@@_renderer_arities_prop
1259   { fencedDivAttributeContextEnd }

```

```

1260 { 0 }
1261 \ExplSyntaxOff

```

2.2.3.13 Header Attribute Context Renderers The following macros are only produced, when the `headerAttributes` option is enabled.

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

These semantics have been deprecated and will be changed in Markdown 3.0.0. From then on, header attribute contexts will only span headings, not the surrounding sections.

```

1262 \def\markdownRendererHeaderAttributeContextBegin{%
1263   \markdownRendererHeaderAttributeContextBeginPrototype}%
1264 \ExplSyntaxOn
1265 \seq_gput_right:Nn
1266   \g_@@_renderers_seq
1267   { headerAttributeContextBegin }
1268 \prop_gput:Nnn
1269   \g_@@_renderer_arities_prop
1270   { headerAttributeContextBegin }
1271   { 0 }
1272 \ExplSyntaxOff
1273 \def\markdownRendererHeaderAttributeContextEnd{%
1274   \markdownRendererHeaderAttributeContextEndPrototype}%
1275 \ExplSyntaxOn
1276 \seq_gput_right:Nn
1277   \g_@@_renderers_seq
1278   { headerAttributeContextEnd }
1279 \prop_gput:Nnn
1280   \g_@@_renderer_arities_prop
1281   { headerAttributeContextEnd }
1282   { 0 }
1283 \ExplSyntaxOff

```

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

```

1284 \def\markdownRendererHeadingOne{%
1285   \markdownRendererHeadingOnePrototype}%
1286 \ExplSyntaxOn
1287 \seq_gput_right:Nn
1288   \g_@@_renderers_seq
1289   { headingOne }
1290 \prop_gput:Nnn

```

```

1291 \g_@@_renderer_arities_prop
1292 { headingOne }
1293 { 1 }
1294 \ExplSyntaxOff

```

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

```

1295 \def\markdownRendererHeadingTwo{%
1296 \markdownRendererHeadingTwoPrototype}%
1297 \ExplSyntaxOn
1298 \seq_gput_right:Nn
1299 \g_@@_renderers_seq
1300 { headingTwo }
1301 \prop_gput:Nnn
1302 \g_@@_renderer_arities_prop
1303 { headingTwo }
1304 { 1 }
1305 \ExplSyntaxOff

```

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

```

1306 \def\markdownRendererHeadingThree{%
1307 \markdownRendererHeadingThreePrototype}%
1308 \ExplSyntaxOn
1309 \seq_gput_right:Nn
1310 \g_@@_renderers_seq
1311 { headingThree }
1312 \prop_gput:Nnn
1313 \g_@@_renderer_arities_prop
1314 { headingThree }
1315 { 1 }
1316 \ExplSyntaxOff

```

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

```

1317 \def\markdownRendererHeadingFour{%
1318 \markdownRendererHeadingFourPrototype}%
1319 \ExplSyntaxOn
1320 \seq_gput_right:Nn
1321 \g_@@_renderers_seq
1322 { headingFour }
1323 \prop_gput:Nnn
1324 \g_@@_renderer_arities_prop
1325 { headingFour }
1326 { 1 }
1327 \ExplSyntaxOff

```


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

```

1328 \def\markdownRendererHeadingFive{%
1329   \markdownRendererHeadingFivePrototype}%
1330 \ExplSyntaxOn
1331 \seq_gput_right:Nn
1332   \g_@@_renderers_seq
1333   { headingFive }
1334 \prop_gput:Nnn
1335   \g_@@_renderer_arities_prop
1336   { headingFive }
1337   { 1 }
1338 \ExplSyntaxOff

```

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

```

1339 \def\markdownRendererHeadingSix{%
1340   \markdownRendererHeadingSixPrototype}%
1341 \ExplSyntaxOn
1342 \seq_gput_right:Nn
1343   \g_@@_renderers_seq
1344   { headingSix }
1345 \prop_gput:Nnn
1346   \g_@@_renderer_arities_prop
1347   { headingSix }
1348   { 1 }
1349 \ExplSyntaxOff

```

2.2.3.15 HTML Comment Renderers 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.

The `\markdownRendererBlockHtmlCommentBegin` and `\markdownRendererBlockHtmlCommentEnd` macros represent the beginning and the end of a block HTML comment. The macros receive no arguments.

```

1350 \def\markdownRendererInlineHtmlComment{%
1351   \markdownRendererInlineHtmlCommentPrototype}%
1352 \ExplSyntaxOn
1353 \seq_gput_right:Nn
1354   \g_@@_renderers_seq
1355   { inlineHtmlComment }
1356 \prop_gput:Nnn
1357   \g_@@_renderer_arities_prop
1358   { inlineHtmlComment }
1359   { 1 }

```

```

1360 \ExplSyntaxOff
1361 \def\markdownRendererBlockHtmlCommentBegin{%
1362   \markdownRendererBlockHtmlCommentBeginPrototype}%
1363 \ExplSyntaxOn
1364 \seq_gput_right:Nn
1365   \g_@@_renderers_seq
1366   { blockHtmlCommentBegin }
1367 \prop_gput:Nnn
1368   \g_@@_renderer_arities_prop
1369   { blockHtmlCommentBegin }
1370   { 0 }
1371 \ExplSyntaxOff
1372 \def\markdownRendererBlockHtmlCommentEnd{%
1373   \markdownRendererBlockHtmlCommentEndPrototype}%
1374 \ExplSyntaxOn
1375 \seq_gput_right:Nn
1376   \g_@@_renderers_seq
1377   { blockHtmlCommentEnd }
1378 \prop_gput:Nnn
1379   \g_@@_renderer_arities_prop
1380   { blockHtmlCommentEnd }
1381   { 0 }
1382 \ExplSyntaxOff

```

2.2.3.16 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.

```

1383 \def\markdownRendererInlineHtmlTag{%
1384   \markdownRendererInlineHtmlTagPrototype}%
1385 \ExplSyntaxOn
1386 \seq_gput_right:Nn
1387   \g_@@_renderers_seq
1388   { inlineHtmlTag }
1389 \prop_gput:Nnn
1390   \g_@@_renderer_arities_prop
1391   { inlineHtmlTag }
1392   { 1 }
1393 \ExplSyntaxOff
1394 \def\markdownRendererInputBlockHtmlElement{%
1395   \markdownRendererInputBlockHtmlElementPrototype}%

```

```

1396 \ExplSyntaxOn
1397 \seq_gput_right:Nn
1398   \g_@@_renderers_seq
1399   { inputBlockHtmlElement }
1400 \prop_gput:Nnn
1401   \g_@@_renderer_arities_prop
1402   { inputBlockHtmlElement }
1403   { 1 }
1404 \ExplSyntaxOff

```

2.2.3.17 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.

```

1405 \def\markdownRendererImage{%
1406   \markdownRendererImagePrototype}%
1407 \ExplSyntaxOn
1408 \seq_gput_right:Nn
1409   \g_@@_renderers_seq
1410   { image }
1411 \prop_gput:Nnn
1412   \g_@@_renderer_arities_prop
1413   { image }
1414   { 4 }
1415 \ExplSyntaxOff

```

2.2.3.18 Interblock Separator Renderer The `\markdownRendererInterblockSeparator` macro represents a separator between two markdown block elements. The macro receives no arguments.

```

1416 \def\markdownRendererInterblockSeparator{%
1417   \markdownRendererInterblockSeparatorPrototype}%
1418 \ExplSyntaxOn
1419 \seq_gput_right:Nn
1420   \g_@@_renderers_seq
1421   { interblockSeparator }
1422 \prop_gput:Nnn
1423   \g_@@_renderer_arities_prop
1424   { interblockSeparator }
1425   { 0 }
1426 \ExplSyntaxOff

```

2.2.3.19 Line Block Renderer 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.

```

1427 \def\markdownRendererLineBlockBegin{%
1428   \markdownRendererLineBlockBeginPrototype}%
1429 \ExplSyntaxOn
1430 \seq_gput_right:Nn
1431   \g_@@_renderers_seq
1432   { lineBlockBegin }
1433 \prop_gput:Nnn
1434   \g_@@_renderer_arities_prop
1435   { lineBlockBegin }
1436   { 0 }
1437 \ExplSyntaxOff
1438 \def\markdownRendererLineBlockEnd{%
1439   \markdownRendererLineBlockEndPrototype}%
1440 \ExplSyntaxOn
1441 \seq_gput_right:Nn
1442   \g_@@_renderers_seq
1443   { lineBlockEnd }
1444 \prop_gput:Nnn
1445   \g_@@_renderer_arities_prop
1446   { lineBlockEnd }
1447   { 0 }
1448 \ExplSyntaxOff

```

2.2.3.20 Line Break Renderer The `\markdownRendererHardLineBreak` macro represents a hard line break. The macro receives no arguments.

The `\markdownRendererLineBreak` and `\markdownRendererLineBreakPrototype` macros have been deprecated and will be removed in Markdown 3.0.0.

```

1449 \ExplSyntaxOn
1450 \cs_new:Npn
1451   \markdownRendererHardLineBreak
1452   {
1453     \cs_if_exist:NTF
1454       \markdownRendererLineBreak
1455       {
1456         \markdownWarning
1457         {
1458           Line-break-renderer-has-been-deprecated,~
1459           to-be-removed-in-Markdown-3.0.0
1460         }
1461         \markdownRendererLineBreak
1462       }
1463     {

```

```

1464     \cs_if_exist:NTF
1465     \markdownRendererLineBreakPrototype
1466     {
1467       \markdownWarning
1468       {
1469         Line~break~renderer~prototype~has~been~deprecated,~
1470         to~be~removed~in~Markdown~3.0.0
1471       }
1472     \markdownRendererLineBreakPrototype
1473     }
1474     {
1475     \markdownRendererHardLineBreakPrototype
1476     }
1477   }
1478 }
1479 \seq_gput_right:Nn
1480 \g_@@_renderers_seq
1481 { lineBreak }
1482 \prop_gput:Nnn
1483 \g_@@_renderer_arities_prop
1484 { lineBreak }
1485 { 0 }
1486 \seq_gput_right:Nn
1487 \g_@@_renderers_seq
1488 { hardLineBreak }
1489 \prop_gput:Nnn
1490 \g_@@_renderer_arities_prop
1491 { hardLineBreak }
1492 { 0 }
1493 \ExplSyntaxOff

```

2.2.3.21 Link Renderer The `\markdownRendererLink` macro represents a hyper-link. 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.

```

1494 \def\markdownRendererLink{%
1495   \markdownRendererLinkPrototype}%
1496 \ExplSyntaxOn
1497 \seq_gput_right:Nn
1498 \g_@@_renderers_seq
1499 { link }
1500 \prop_gput:Nnn
1501 \g_@@_renderer_arities_prop
1502 { link }
1503 { 4 }
1504 \ExplSyntaxOff

```

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

A \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.

```

1505 \def\markdownRendererDocumentBegin{%
1506   \markdownRendererDocumentBeginPrototype}%
1507 \ExplSyntaxOn
1508 \seq_gput_right:Nn
1509   \g_@@_renderers_seq
1510   { documentBegin }
1511 \prop_gput:Nnn
1512   \g_@@_renderer_arities_prop
1513   { documentBegin }
1514   { 0 }
1515 \ExplSyntaxOff
1516 \def\markdownRendererDocumentEnd{%
1517   \markdownRendererDocumentEndPrototype}%
1518 \ExplSyntaxOn
1519 \seq_gput_right:Nn
1520   \g_@@_renderers_seq
1521   { documentEnd }
1522 \prop_gput:Nnn
1523   \g_@@_renderer_arities_prop
1524   { documentEnd }
1525   { 0 }
1526 \ExplSyntaxOff

```

2.2.3.23 Non-Breaking Space Renderer The `\markdownRendererNbsp` macro represents a non-breaking space.

```

1527 \def\markdownRendererNbsp{%
1528   \markdownRendererNbspPrototype}%
1529 \ExplSyntaxOn
1530 \seq_gput_right:Nn
1531   \g_@@_renderers_seq
1532   { nbsp }
1533 \prop_gput:Nnn
1534   \g_@@_renderer_arities_prop
1535   { nbsp }
1536   { 0 }
1537 \ExplSyntaxOff

```

2.2.3.24 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.

The `\markdownRendererFootnote` and `\markdownRendererFootnotePrototype` macros have been deprecated and will be removed in Markdown 3.0.0.

```

1538 \ExplSyntaxOn
1539 \cs_new:Npn
1540   \markdownRendererNote
1541   {
1542     \cs_if_exist:NTF
1543       \markdownRendererFootnote
1544       {
1545         \markdownWarning
1546         {
1547           Footnote~renderer~has~been~deprecated,~
1548           to~be~removed~in~Markdown~3.0.0
1549         }
1550       \markdownRendererFootnote
1551     }
1552     {
1553       \cs_if_exist:NTF
1554         \markdownRendererFootnotePrototype
1555         {
1556           \markdownWarning
1557           {
1558             Footnote~renderer~prototype~has~been~deprecated,~
1559             to~be~removed~in~Markdown~3.0.0
1560           }
1561         \markdownRendererFootnotePrototype
1562       }
1563     }
1564     \markdownRendererNotePrototype
1565   }
1566 }
1567 }
1568 \seq_gput_right:Nn
1569   \g_@@_renderers_seq
1570   { footnote }
1571 \prop_gput:Nnn
1572   \g_@@_renderer_arities_prop
1573   { footnote }
1574   { 1 }
1575 \seq_gput_right:Nn
1576   \g_@@_renderers_seq
1577   { note }
1578 \prop_gput:Nnn

```

```

1579 \g_@@_renderer_arities_prop
1580 { note }
1581 { 1 }
1582 \ExplSyntaxOff

```

2.2.3.25 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.

```

1583 \def\markdownRendererOlBegin{%
1584 \markdownRendererOlBeginPrototype}%
1585 \ExplSyntaxOn
1586 \seq_gput_right:Nn
1587 \g_@@_renderers_seq
1588 { olBegin }
1589 \prop_gput:Nnn
1590 \g_@@_renderer_arities_prop
1591 { olBegin }
1592 { 0 }
1593 \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.

```

1594 \def\markdownRendererOlBeginTight{%
1595 \markdownRendererOlBeginTightPrototype}%
1596 \ExplSyntaxOn
1597 \seq_gput_right:Nn
1598 \g_@@_renderers_seq
1599 { olBeginTight }
1600 \prop_gput:Nnn
1601 \g_@@_renderer_arities_prop
1602 { olBeginTight }
1603 { 0 }
1604 \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`).

```

1605 \def\markdownRendererFancyOlBegin{%
1606 \markdownRendererFancyOlBeginPrototype}%

```



```

1607 \ExplSyntaxOn
1608 \seq_gput_right:Nn
1609   \g_@@_renderers_seq
1610   { fancyOlBegin }
1611 \prop_gput:Nnn
1612   \g_@@_renderer_arities_prop
1613   { fancyOlBegin }
1614   { 2 }
1615 \ExplSyntaxOff

```

The `\markdownRendererFancyOlBeginTight` macro represents the beginning of a fancy ordered list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `fancyLists` and `tightLists` options are enabled. The macro receives two arguments: the style of the list item labels, and the style of delimiters between list item labels and texts. See the `\markdownRendererFancyOlBegin` macro for the valid style values.

```

1616 \def\markdownRendererFancyOlBeginTight{%
1617   \markdownRendererFancyOlBeginTightPrototype}%
1618 \ExplSyntaxOn
1619 \seq_gput_right:Nn
1620   \g_@@_renderers_seq
1621   { fancyOlBeginTight }
1622 \prop_gput:Nnn
1623   \g_@@_renderer_arities_prop
1624   { fancyOlBeginTight }
1625   { 2 }
1626 \ExplSyntaxOff

```

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

```

1627 \def\markdownRendererOlItem{%
1628   \markdownRendererOlItemPrototype}%
1629 \ExplSyntaxOn
1630 \seq_gput_right:Nn
1631   \g_@@_renderers_seq
1632   { olItem }
1633 \prop_gput:Nnn
1634   \g_@@_renderer_arities_prop
1635   { olItem }
1636   { 0 }
1637 \ExplSyntaxOff

```

The `\markdownRendererOlItemEnd` macro represents the end of an item in an ordered list. This macro will only be produced, when the `fancyLists` option is disabled. The macro receives no arguments.

```

1638 \def\markdownRenderer01ItemEnd{%
1639   \markdownRenderer01ItemEndPrototype}%
1640 \ExplSyntaxOn
1641 \seq_gput_right:Nn
1642   \g_@@_renderers_seq
1643   { olItemEnd }
1644 \prop_gput:Nnn
1645   \g_@@_renderer_arities_prop
1646   { olItemEnd }
1647   { 0 }
1648 \ExplSyntaxOff

```

The `\markdownRenderer01ItemWithNumber` 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.

```

1649 \def\markdownRenderer01ItemWithNumber{%
1650   \markdownRenderer01ItemWithNumberPrototype}%
1651 \ExplSyntaxOn
1652 \seq_gput_right:Nn
1653   \g_@@_renderers_seq
1654   { olItemWithNumber }
1655 \prop_gput:Nnn
1656   \g_@@_renderer_arities_prop
1657   { olItemWithNumber }
1658   { 1 }
1659 \ExplSyntaxOff

```

The `\markdownRendererFancy01Item` 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.

```

1660 \def\markdownRendererFancy01Item{%
1661   \markdownRendererFancy01ItemPrototype}%
1662 \ExplSyntaxOn
1663 \seq_gput_right:Nn
1664   \g_@@_renderers_seq
1665   { fancy01Item }
1666 \prop_gput:Nnn
1667   \g_@@_renderer_arities_prop
1668   { fancy01Item }
1669   { 0 }
1670 \ExplSyntaxOff

```

The `\markdownRendererFancy01ItemEnd` 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.

```

1671 \def\markdownRendererFancyOliItemEnd{%
1672   \markdownRendererFancyOliItemEndPrototype}%
1673 \ExplSyntaxOn
1674 \seq_gput_right:Nn
1675   \g_@@_renderers_seq
1676   { fancyOliItemEnd }
1677 \prop_gput:Nnn
1678   \g_@@_renderer_arities_prop
1679   { fancyOliItemEnd }
1680   { 0 }
1681 \ExplSyntaxOff

```

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

```

1682 \def\markdownRendererFancyOliItemWithNumber{%
1683   \markdownRendererFancyOliItemWithNumberPrototype}%
1684 \ExplSyntaxOn
1685 \seq_gput_right:Nn
1686   \g_@@_renderers_seq
1687   { fancyOliItemWithNumber }
1688 \prop_gput:Nnn
1689   \g_@@_renderer_arities_prop
1690   { fancyOliItemWithNumber }
1691   { 1 }
1692 \ExplSyntaxOff

```

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

```

1693 \def\markdownRendererOliEnd{%
1694   \markdownRendererOliEndPrototype}%
1695 \ExplSyntaxOn
1696 \seq_gput_right:Nn
1697   \g_@@_renderers_seq
1698   { oliEnd }
1699 \prop_gput:Nnn
1700   \g_@@_renderer_arities_prop
1701   { oliEnd }
1702   { 0 }
1703 \ExplSyntaxOff

```

The `\markdownRendererOliEndTight` 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.

```
1704 \def\markdownRendererO1EndTight{%
1705   \markdownRendererO1EndTightPrototype}%
1706 \ExplSyntaxOn
1707 \seq_gput_right:Nn
1708   \g_@@_renderers_seq
1709   { olEndTight }
1710 \prop_gput:Nnn
1711   \g_@@_renderer_arities_prop
1712   { olEndTight }
1713   { 0 }
1714 \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.

```
1715 \def\markdownRendererFancyO1End{%
1716   \markdownRendererFancyO1EndPrototype}%
1717 \ExplSyntaxOn
1718 \seq_gput_right:Nn
1719   \g_@@_renderers_seq
1720   { fancyO1End }
1721 \prop_gput:Nnn
1722   \g_@@_renderer_arities_prop
1723   { fancyO1End }
1724   { 0 }
1725 \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.

```
1726 \def\markdownRendererFancyO1EndTight{%
1727   \markdownRendererFancyO1EndTightPrototype}%
1728 \ExplSyntaxOn
1729 \seq_gput_right:Nn
1730   \g_@@_renderers_seq
1731   { fancyO1EndTight }
1732 \prop_gput:Nnn
1733   \g_@@_renderer_arities_prop
1734   { fancyO1EndTight }
1735   { 0 }
1736 \ExplSyntaxOff
```

2.2.3.26 Parenthesized Citations Renderer 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.

```

1737 \def\markdownRendererCite{%
1738   \markdownRendererCitePrototype}%
1739 \ExplSyntaxOn
1740 \seq_gput_right:Nn
1741   \g_@@_renderers_seq
1742   { cite }
1743 \prop_gput:Nnn
1744   \g_@@_renderer_arities_prop
1745   { cite }
1746   { 1 }
1747 \ExplSyntaxOff

```

2.2.3.27 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.

```

1748 \def\markdownRendererInputRawInline{%
1749   \markdownRendererInputRawInlinePrototype}%
1750 \ExplSyntaxOn
1751 \seq_gput_right:Nn
1752   \g_@@_renderers_seq
1753   { inputRawInline }
1754 \prop_gput:Nnn
1755   \g_@@_renderer_arities_prop
1756   { inputRawInline }
1757   { 2 }
1758 \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.

```

1759 \def\markdownRendererInputRawBlock{%
1760   \markdownRendererInputRawBlockPrototype}%
1761 \ExplSyntaxOn
1762 \seq_gput_right:Nn

```

```

1763 \g_@@_renderers_seq
1764 { inputRawBlock }
1765 \prop_gput:Nnn
1766 \g_@@_renderer_arities_prop
1767 { inputRawBlock }
1768 { 2 }
1769 \ExplSyntaxOff

```

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

```

1770 \def\markdownRendererSectionBegin{%
1771 \markdownRendererSectionBeginPrototype}%
1772 \ExplSyntaxOn
1773 \seq_gput_right:Nn
1774 \g_@@_renderers_seq
1775 { sectionBegin }
1776 \prop_gput:Nnn
1777 \g_@@_renderer_arities_prop
1778 { sectionBegin }
1779 { 0 }
1780 \ExplSyntaxOff
1781 \def\markdownRendererSectionEnd{%
1782 \markdownRendererSectionEndPrototype}%
1783 \ExplSyntaxOn
1784 \seq_gput_right:Nn
1785 \g_@@_renderers_seq
1786 { sectionEnd }
1787 \prop_gput:Nnn
1788 \g_@@_renderer_arities_prop
1789 { sectionEnd }
1790 { 0 }
1791 \ExplSyntaxOff

```

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

```

1792 \def\markdownRendererReplacementCharacter{%
1793 \markdownRendererReplacementCharacterPrototype}%
1794 \ExplSyntaxOn
1795 \seq_gput_right:Nn
1796 \g_@@_renderers_seq
1797 { replacementCharacter }
1798 \prop_gput:Nnn
1799 \g_@@_renderer_arities_prop
1800 { replacementCharacter }

```

```

1801 { 0 }
1802 \ExplSyntaxOff

```

2.2.3.30 Special Character Renderers The following macros replace any special plain \TeX characters, including the active pipe character (`|`) of `ConTeXt`, in the input text. These macros will only be produced, when the `hybrid` option is `false`.

```

1803 \def\markdownRendererLeftBrace{%
1804   \markdownRendererLeftBracePrototype}%
1805 \ExplSyntaxOn
1806 \seq_gput_right:Nn
1807   \g_@@_renderers_seq
1808   { leftBrace }
1809 \prop_gput:Nnn
1810   \g_@@_renderer_arities_prop
1811   { leftBrace }
1812   { 0 }
1813 \ExplSyntaxOff
1814 \def\markdownRendererRightBrace{%
1815   \markdownRendererRightBracePrototype}%
1816 \ExplSyntaxOn
1817 \seq_gput_right:Nn
1818   \g_@@_renderers_seq
1819   { rightBrace }
1820 \prop_gput:Nnn
1821   \g_@@_renderer_arities_prop
1822   { rightBrace }
1823   { 0 }
1824 \ExplSyntaxOff
1825 \def\markdownRendererDollarSign{%
1826   \markdownRendererDollarSignPrototype}%
1827 \ExplSyntaxOn
1828 \seq_gput_right:Nn
1829   \g_@@_renderers_seq
1830   { dollarSign }
1831 \prop_gput:Nnn
1832   \g_@@_renderer_arities_prop
1833   { dollarSign }
1834   { 0 }
1835 \ExplSyntaxOff
1836 \def\markdownRendererPercentSign{%
1837   \markdownRendererPercentSignPrototype}%
1838 \ExplSyntaxOn
1839 \seq_gput_right:Nn
1840   \g_@@_renderers_seq
1841   { percentSign }
1842 \prop_gput:Nnn

```

```

1843 \g_@@_renderer_arities_prop
1844 { percentSign }
1845 { 0 }
1846 \ExplSyntaxOff
1847 \def\markdownRendererAmpersand{%
1848 \markdownRendererAmpersandPrototype}%
1849 \ExplSyntaxOn
1850 \seq_gput_right:Nn
1851 \g_@@_renderers_seq
1852 { ampersand }
1853 \prop_gput:Nnn
1854 \g_@@_renderer_arities_prop
1855 { ampersand }
1856 { 0 }
1857 \ExplSyntaxOff
1858 \def\markdownRendererUnderscore{%
1859 \markdownRendererUnderscorePrototype}%
1860 \ExplSyntaxOn
1861 \seq_gput_right:Nn
1862 \g_@@_renderers_seq
1863 { underscore }
1864 \prop_gput:Nnn
1865 \g_@@_renderer_arities_prop
1866 { underscore }
1867 { 0 }
1868 \ExplSyntaxOff
1869 \def\markdownRendererHash{%
1870 \markdownRendererHashPrototype}%
1871 \ExplSyntaxOn
1872 \seq_gput_right:Nn
1873 \g_@@_renderers_seq
1874 { hash }
1875 \prop_gput:Nnn
1876 \g_@@_renderer_arities_prop
1877 { hash }
1878 { 0 }
1879 \ExplSyntaxOff
1880 \def\markdownRendererCircumflex{%
1881 \markdownRendererCircumflexPrototype}%
1882 \ExplSyntaxOn
1883 \seq_gput_right:Nn
1884 \g_@@_renderers_seq
1885 { circumflex }
1886 \prop_gput:Nnn
1887 \g_@@_renderer_arities_prop
1888 { circumflex }
1889 { 0 }

```



```

1890 \ExplSyntaxOff
1891 \def\markdownRendererBackslash{%
1892   \markdownRendererBackslashPrototype}%
1893 \ExplSyntaxOn
1894 \seq_gput_right:Nn
1895   \g_@@_renderers_seq
1896   { backslash }
1897 \prop_gput:Nnn
1898   \g_@@_renderer_arities_prop
1899   { backslash }
1900   { 0 }
1901 \ExplSyntaxOff
1902 \def\markdownRendererTilde{%
1903   \markdownRendererTildePrototype}%
1904 \ExplSyntaxOn
1905 \seq_gput_right:Nn
1906   \g_@@_renderers_seq
1907   { tilde }
1908 \prop_gput:Nnn
1909   \g_@@_renderer_arities_prop
1910   { tilde }
1911   { 0 }
1912 \ExplSyntaxOff
1913 \def\markdownRendererPipe{%
1914   \markdownRendererPipePrototype}%
1915 \ExplSyntaxOn
1916 \seq_gput_right:Nn
1917   \g_@@_renderers_seq
1918   { pipe }
1919 \prop_gput:Nnn
1920   \g_@@_renderer_arities_prop
1921   { pipe }
1922   { 0 }
1923 \ExplSyntaxOff

```

2.2.3.31 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.

```

1924 \def\markdownRendererStrikeThrough{%
1925   \markdownRendererStrikeThroughPrototype}%
1926 \ExplSyntaxOn
1927 \seq_gput_right:Nn
1928   \g_@@_renderers_seq
1929   { strikeThrough }
1930 \prop_gput:Nnn

```

```

1931 \g_@@_renderer_arities_prop
1932 { strikeThrough }
1933 { 1 }
1934 \ExplSyntaxOff

```

2.2.3.32 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.

```

1935 \def\markdownRendererSubscript{%
1936 \markdownRendererSubscriptPrototype}%
1937 \ExplSyntaxOn
1938 \seq_gput_right:Nn
1939 \g_@@_renderers_seq
1940 { subscript }
1941 \prop_gput:Nnn
1942 \g_@@_renderer_arities_prop
1943 { subscript }
1944 { 1 }

```

2.2.3.33 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.

```

1945 \def\markdownRendererSuperscript{%
1946 \markdownRendererSuperscriptPrototype}%
1947 \ExplSyntaxOn
1948 \seq_gput_right:Nn
1949 \g_@@_renderers_seq
1950 { superscript }
1951 \prop_gput:Nnn
1952 \g_@@_renderer_arities_prop
1953 { superscript }
1954 { 1 }
1955 \ExplSyntaxOff

```

2.2.3.34 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.

```

1956 \def\markdownRendererTable{%
1957   \markdownRendererTablePrototype}%
1958 \ExplSyntaxOn
1959 \seq_gput_right:Nn
1960   \g_@@_renderers_seq
1961   { table }
1962 \prop_gput:Nnn
1963   \g_@@_renderer_arities_prop
1964   { table }
1965   { 3 }
1966 \ExplSyntaxOff

```

2.2.3.35 Tex Math Renderers The `\markdownRendererInlineMath` and `\markdownRendererDisplayMath` macros represent inline and display \TeX math. Both macros receive a single argument that corresponds to the tex math content. These macros will only be produced, when the `texMathDollars` option is enabled.

```

1967 \def\markdownRendererInlineMath{%
1968   \markdownRendererInlineMathPrototype}%
1969 \ExplSyntaxOn
1970 \seq_gput_right:Nn
1971   \g_@@_renderers_seq
1972   { inlineMath }
1973 \prop_gput:Nnn
1974   \g_@@_renderer_arities_prop
1975   { inlineMath }
1976   { 1 }
1977 \ExplSyntaxOff
1978 \def\markdownRendererDisplayMath{%
1979   \markdownRendererDisplayMathPrototype}%
1980 \ExplSyntaxOn
1981 \seq_gput_right:Nn
1982   \g_@@_renderers_seq
1983   { displayMath }
1984 \prop_gput:Nnn
1985   \g_@@_renderer_arities_prop
1986   { displayMath }
1987   { 1 }
1988 \ExplSyntaxOff

```

2.2.3.36 Text Citations Renderer 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.

```

1989 \def\markdownRendererTextCite{%
1990   \markdownRendererTextCitePrototype}%
1991 \ExplSyntaxOn
1992 \seq_gput_right:Nn
1993   \g_@@_renderers_seq
1994   { textCite }
1995 \prop_gput:Nnn
1996   \g_@@_renderer_arities_prop
1997   { textCite }
1998   { 1 }
1999 \ExplSyntaxOff

```

2.2.3.37 Thematic Break Renderer The `\markdownRendererThematicBreak` macro represents a thematic break. The macro receives no arguments.

The `\markdownRendererHorizontalRule` and `\markdownRendererHorizontalRulePrototype` macros have been deprecated and will be removed in Markdown 3.0.0.

```

2000 \ExplSyntaxOn
2001 \cs_new:Npn
2002   \markdownRendererThematicBreak
2003   {
2004     \cs_if_exist:NTF
2005       \markdownRendererHorizontalRule
2006       {
2007         \markdownWarning
2008         {
2009           Horizontal~rule~renderer~has~been~deprecated,~
2010           to~be~removed~in~Markdown~3.0.0
2011         }
2012         \markdownRendererHorizontalRule
2013       }
2014     {
2015       \cs_if_exist:NTF
2016         \markdownRendererHorizontalRulePrototype
2017         {
2018           \markdownWarning
2019           {
2020             Horizontal~rule~renderer~prototype~has~been~deprecated,~
2021             to~be~removed~in~Markdown~3.0.0
2022           }
2023           \markdownRendererHorizontalRulePrototype
2024         }
2025       {
2026         \markdownRendererThematicBreakPrototype
2027       }

```

```

2028     }
2029   }
2030 \seq_gput_right:Nn
2031   \g_@@_renderers_seq
2032   { horizontalRule }
2033 \prop_gput:Nnn
2034   \g_@@_renderer_arities_prop
2035   { horizontalRule }
2036   { 0 }
2037 \seq_gput_right:Nn
2038   \g_@@_renderers_seq
2039   { thematicBreak }
2040 \prop_gput:Nnn
2041   \g_@@_renderer_arities_prop
2042   { thematicBreak }
2043   { 0 }
2044 \ExplSyntaxOff

```

2.2.3.38 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.

```

2045 \def\markdownRendererTickedBox{%
2046   \markdownRendererTickedBoxPrototype}%
2047 \ExplSyntaxOn
2048 \seq_gput_right:Nn
2049   \g_@@_renderers_seq
2050   { tickedBox }
2051 \prop_gput:Nnn
2052   \g_@@_renderer_arities_prop
2053   { tickedBox }
2054   { 0 }
2055 \ExplSyntaxOff
2056 \def\markdownRendererHalfTickedBox{%
2057   \markdownRendererHalfTickedBoxPrototype}%
2058 \ExplSyntaxOn
2059 \seq_gput_right:Nn
2060   \g_@@_renderers_seq
2061   { halfTickedBox }
2062 \prop_gput:Nnn
2063   \g_@@_renderer_arities_prop
2064   { halfTickedBox }
2065   { 0 }
2066 \ExplSyntaxOff

```

```

2067 \def\markdownRendererUntickedBox{%
2068   \markdownRendererUntickedBoxPrototype}%
2069 \ExplSyntaxOn
2070 \seq_gput_right:Nn
2071   \g_@@_renderers_seq
2072   { untickedBox }
2073 \prop_gput:Nnn
2074   \g_@@_renderer_arities_prop
2075   { untickedBox }
2076   { 0 }
2077 \ExplSyntaxOff

```

2.2.3.39 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.

```

2078 \def\markdownRendererJekyllDataBegin{%
2079   \markdownRendererJekyllDataBeginPrototype}%
2080 \ExplSyntaxOn
2081 \seq_gput_right:Nn
2082   \g_@@_renderers_seq
2083   { jekyllDataBegin }
2084 \prop_gput:Nnn
2085   \g_@@_renderer_arities_prop
2086   { jekyllDataBegin }
2087   { 0 }
2088 \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.

```

2089 \def\markdownRendererJekyllDataEnd{%
2090   \markdownRendererJekyllDataEndPrototype}%
2091 \ExplSyntaxOn
2092 \seq_gput_right:Nn
2093   \g_@@_renderers_seq
2094   { jekyllDataEnd }
2095 \prop_gput:Nnn
2096   \g_@@_renderer_arities_prop
2097   { jekyllDataEnd }
2098   { 0 }
2099 \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.

```
2100 \def\markdownRendererJekyllDataMappingBegin{%
2101   \markdownRendererJekyllDataMappingBeginPrototype}%
2102 \ExplSyntaxOn
2103 \seq_gput_right:Nn
2104   \g_@@_renderers_seq
2105   { jekyllDataMappingBegin }
2106 \prop_gput:Nnn
2107   \g_@@_renderer_arities_prop
2108   { jekyllDataMappingBegin }
2109   { 2 }
2110 \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.

```
2111 \def\markdownRendererJekyllDataMappingEnd{%
2112   \markdownRendererJekyllDataMappingEndPrototype}%
2113 \ExplSyntaxOn
2114 \seq_gput_right:Nn
2115   \g_@@_renderers_seq
2116   { jekyllDataMappingEnd }
2117 \prop_gput:Nnn
2118   \g_@@_renderer_arities_prop
2119   { jekyllDataMappingEnd }
2120   { 0 }
2121 \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.

```
2122 \def\markdownRendererJekyllDataSequenceBegin{%
2123   \markdownRendererJekyllDataSequenceBeginPrototype}%
2124 \ExplSyntaxOn
2125 \seq_gput_right:Nn
2126   \g_@@_renderers_seq
2127   { jekyllDataSequenceBegin }
2128 \prop_gput:Nnn
2129   \g_@@_renderer_arities_prop
2130   { jekyllDataSequenceBegin }
2131   { 2 }
2132 \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.

```
2133 \def\markdownRendererJekyllDataSequenceEnd{%
2134   \markdownRendererJekyllDataSequenceEndPrototype}%
2135 \ExplSyntaxOn
2136 \seq_gput_right:Nn
2137   \g_@@_renderers_seq
2138   { jekyllDataSequenceEnd }
2139 \prop_gput:Nnn
2140   \g_@@_renderer_arities_prop
2141   { jekyllDataSequenceEnd }
2142   { 0 }
2143 \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.

```
2144 \def\markdownRendererJekyllDataBoolean{%
2145   \markdownRendererJekyllDataBooleanPrototype}%
2146 \ExplSyntaxOn
2147 \seq_gput_right:Nn
2148   \g_@@_renderers_seq
2149   { jekyllDataBoolean }
2150 \prop_gput:Nnn
2151   \g_@@_renderer_arities_prop
2152   { jekyllDataBoolean }
2153   { 2 }
2154 \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.

```
2155 \def\markdownRendererJekyllDataNumber{%
2156   \markdownRendererJekyllDataNumberPrototype}%
2157 \ExplSyntaxOn
2158 \seq_gput_right:Nn
2159   \g_@@_renderers_seq
2160   { jekyllDataNumber }
2161 \prop_gput:Nnn
2162   \g_@@_renderer_arities_prop
```



```

2163 { jekyllDataNumber }
2164 { 2 }
2165 \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.

```

2166 \def\markdownRendererJekyllDataString{%
2167   \markdownRendererJekyllDataStringPrototype}%
2168 \ExplSyntaxOn
2169 \seq_gput_right:Nn
2170   \g_@@_renderers_seq
2171   { jekyllDataString }
2172 \prop_gput:Nnn
2173   \g_@@_renderer_arities_prop
2174   { jekyllDataString }
2175   { 2 }
2176 \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 2.2.4.1 for the description of the high-level expl3 interface that you can also use to react to YAML metadata.

```

2177 \def\markdownRendererJekyllDataEmpty{%
2178   \markdownRendererJekyllDataEmptyPrototype}%
2179 \ExplSyntaxOn
2180 \seq_gput_right:Nn
2181   \g_@@_renderers_seq
2182   { jekyllDataEmpty }
2183 \prop_gput:Nnn
2184   \g_@@_renderer_arities_prop
2185   { jekyllDataEmpty }
2186   { 1 }
2187 \ExplSyntaxOff

```

2.2.4 Token Renderer Prototypes

2.2.4.1 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 \LaTeX 3 kernel.

```

2188 \ExplSyntaxOn
2189 \keys_define:nn

```

```

2190 { markdown/jekyllData }
2191 { }
2192 \ExplSyntaxOff

```

The following T_EX macros provide definitions for the token renderers (see Section 2.2.3) that have not been redefined by the user. These macros are intended to be redefined by macro package authors who wish to provide sensible default token renderers. They are also redefined by the L^AT_EX and ConT_EXt implementations (see sections 3.3 and 3.4).

```

2193 \ExplSyntaxOn
2194 \cs_new:Nn \@@_plaintex_define_renderer_prototypes:
2195 {
2196   \seq_map_function:NN
2197     \g_@@_renderers_seq
2198     \@@_plaintex_define_renderer_prototype:n
2199   \let\markdownRendererBlockHtmlCommentBeginPrototype=\iffalse
2200   \let\markdownRendererBlockHtmlCommentBegin=\iffalse
2201   \let\markdownRendererBlockHtmlCommentEndPrototype=\fi
2202   \let\markdownRendererBlockHtmlCommentEnd=\fi

```

The `\markdownRendererFootnote` and `\markdownRendererFootnotePrototype` macros have been deprecated and will be removed in Markdown 3.0.0.

```

2203 \cs_undefine:N \markdownRendererFootnote
2204 \cs_undefine:N \markdownRendererFootnotePrototype

```

The `\markdownRendererHorizontalRule` and `\markdownRendererHorizontalRulePrototype` macros have been deprecated and will be removed in Markdown 3.0.0.

```

2205 \cs_undefine:N \markdownRendererHorizontalRule
2206 \cs_undefine:N \markdownRendererHorizontalRulePrototype
2207 }
2208 \cs_new:Nn \@@_plaintex_define_renderer_prototype:n
2209 {
2210   \@@_renderer_prototype_tl_to_csname:nN
2211   { #1 }
2212   \l_tmpa_tl
2213   \prop_get:NnN
2214     \g_@@_renderer_arities_prop
2215     { #1 }
2216     \l_tmpb_tl
2217   \@@_plaintex_define_renderer_prototype:cV
2218     { \l_tmpa_tl }
2219     \l_tmpb_tl
2220 }
2221 \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
2222 {
2223   \tl_set:Nn
2224     \l_tmpa_tl
2225     { \str_uppercase:n { #1 } }

```

```

2226 \tl_set:Nx
2227 #2
2228 {
2229     markdownRenderer
2230     \tl_head:f { \l_tmpa_tl }
2231     \tl_tail:n { #1 }
2232     Prototype
2233 }
2234 }
2235 \cs_new:Nn \@@_plaintex_define_renderer_prototype:Nn
2236 {
2237     \cs_generate_from_arg_count:NNnn
2238     #1
2239     \cs_set:Npn
2240     { #2 }
2241     { }
2242 }
2243 \cs_generate_variant:Nn
2244 \@@_plaintex_define_renderer_prototype:Nn
2245 { cV }
2246 \@@_plaintex_define_renderer_prototypes:
2247 \ExplSyntaxOff

```

2.2.5 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.

2.2.6 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*.

```
2248 \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.

```
2249 \let\markdownReadAndConvert\relax
2250 \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*.

```
2251 \catcode`\|=0\catcode`\=12%
2252 |gdef|markdownBegin{%
2253   |markdownReadAndConvert{\markdownEnd}%
2254   {|\markdownEnd}%
2255 |endgroup
```

The macro is exposed in the interface, so that the user can create their own markdown environments. Due to the way the arguments are passed to Lua (see Section 3.2.6), the first argument may not contain the string `]]` (regardless of the category code of the bracket symbol (`]`)).

The `\markdownMode` macro specifies how the plain \TeX implementation interfaces with the Lua interface. The valid values and their meaning are as follows:

- `0` – Shell escape via the 18 output file stream
- `1` – Shell escape via the Lua `os.execute` method
- `2` – Direct Lua access
- `3` – The `lt3luabridge` Lua package

By defining the macro, the user can coerce the package to use a specific mode. If the user does not define the macro prior to loading the plain \TeX implementation, the correct value will be automatically detected. The outcome of changing the value of `\markdownMode` after the implementation has been loaded is undefined.

The `\markdownMode` macro has been deprecated and will be removed in Markdown 3.0.0. The code that corresponds to `\markdownMode` value of `3` will be the only implementation.

```
2256 \ExplSyntaxOn
2257 \cs_if_exist:NF
2258   \markdownMode
2259   {
2260     \file_if_exist:nTF
2261       { lt3luabridge.tex }
2262       {
2263         \cs_new:Npn
2264           \markdownMode
2265           { 3 }
2266       }
2267     {
2268       \cs_if_exist:NTF
2269         \directlua
```

```

2270     {
2271         \cs_new:Npn
2272             \markdownMode
2273             { 2 }
2274     }
2275     {
2276         \cs_new:Npn
2277             \markdownMode
2278             { 0 }
2279     }
2280 }
2281 }
2282 \ExplSyntaxOff

```

The `\markdownLuaRegisterIBCallback` and `\markdownLuaUnregisterIBCallback` macros have been deprecated and will be removed in Markdown 3.0.0:

```

2283 \def\markdownLuaRegisterIBCallback#1{\relax}%
2284 \def\markdownLuaUnregisterIBCallback#1{\relax}%

```

2.3 \LaTeX Interface

The \LaTeX interface provides \LaTeX environments for the typesetting of markdown input from within \LaTeX , facilities for setting Lua, plain \TeX , and \LaTeX options used during the conversion from markdown to plain \TeX , and facilities for changing the way markdown tokens are rendered. The rest of the interface is inherited from the plain \TeX interface (see Section 2.2).

The \LaTeX implementation redefines the plain \TeX logging macros (see Section 3.2.1) to use the \LaTeX `\PackageInfo`, `\PackageWarning`, and `\PackageError` macros.

```

2285 \newcommand\markdownInfo[1]{\PackageInfo{markdown}{#1}}%
2286 \newcommand\markdownWarning[1]{\PackageWarning{markdown}{#1}}%
2287 \newcommand\markdownError[2]{\PackageError{markdown}{#1}{#2.}}%
2288 \input markdown/markdown

```

The \LaTeX interface is implemented by the `markdown.sty` file, which can be loaded from the \LaTeX document preamble as follows:

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

where `<options>` are the \LaTeX interface options (see Section 2.3.2). Note that `<options>` inside the `\usepackage` macro may not set the `markdownRenderers` (see Section 2.3.2.5) and `markdownRendererPrototypes` (see Section 2.3.2.6) keys. 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*` L^AT_EX environments, and redefines the `\markdownInput` command.

The `markdown` and `markdown*` L^AT_EX environments are used to typeset markdown document fragments. The starred version of the `markdown` environment accepts L^AT_EX interface options (see Section 2.3.2) as its only argument. These options will only influence this markdown document fragment.

```
2289 \newenvironment{markdown}\relax\relax
2290 \newenvironment{markdown*}[1]\relax\relax
```

You may prepend your own code to the `\markdown` macro and append your own code to the `\endmarkdown` macro to produce special effects before and after the `markdown` L^AT_EX environment (and likewise for the starred version).

Note that the `markdown` and `markdown*` L^AT_EX environments are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros exposed by the plain T_EX interface.

The following example L^AT_EX code showcases the usage of the `markdown` and `markdown*` environments:

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

The `\markdownInput` macro accepts a single mandatory parameter containing the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain T_EX. Unlike the `\markdownInput` macro provided by the plain T_EX interface, this macro also accepts L^AT_EX interface options (see Section 2.3.2) as its optional argument. These options will only influence this markdown document.

The following example L^AT_EX code showcases the usage of the `\markdownInput` macro:

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

2.3.2 Options

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

Except for the `plain` option described in Section 2.3.2.1, and the \LaTeX themes described in Section 2.3.2.2, and the \LaTeX setup snippets described in Section 2.3.2.3, \LaTeX options map directly to the options recognized by the plain \TeX interface (see Section 2.2.2) and to the markdown token renderers and their prototypes recognized by the plain \TeX interface (see Sections 2.2.3 and 2.2.4).

The \LaTeX options may be specified when loading the \LaTeX package, when using the `markdown*` \LaTeX environment or the `\markdownInput` macro (see Section 2.3), or via the `\markdownSetup` macro. The `\markdownSetup` macro receives the options to set up as its only argument:

```
2291 \ExplSyntaxOn
2292 \cs_new:Nn
2293   \@@_setup:n
2294   {
2295     \keys_set:nn
2296       { markdown/latex-options }
2297       { #1 }
2298   }
2299 \let\markdownSetup=\@@_setup:n
2300 \ExplSyntaxOff
```

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

```
2301 \newcommand\markdownSetupSnippet[2]{%
2302   \markdownIfSnippetExists{#1}%
2303   {%
2304     \markdownWarning
2305       {Redefined setup snippet \markdownLaTeXThemeName#1}%
2306     \csname markdownLaTeXSetupSnippet%
2307       \markdownLaTeXThemeName#1\endcsname={#2}%
2308   }{%
2309     \newtoks\next
2310     \next={#2}%
2311     \expandafter\let\csname markdownLaTeXSetupSnippet%
2312       \markdownLaTeXThemeName#1\endcsname=\next
2313   }}%
```

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

```
2314 \newcommand\markdownIfSnippetExists[3]{%
2315   \@ifundefined
```

```

2316     {markdownLaTeXSetupSnippet\markdownLaTeXThemeName#1}%
2317     {#3}{#2}}%

```

See Section 2.3.2.2 for information on interactions between setup snippets and L^AT_EX themes. See Section 2.3.2.3 for information about invoking the stored setup snippets.

To enable the enumeration of L^AT_EX options, we will maintain the `\g_@@_latex_options_seq` sequence.

```

2318 \ExplSyntaxOn
2319 \seq_new:N \g_@@_latex_options_seq

```

To enable the reflection of default L^AT_EX options and their types, we will maintain the `\g_@@_default_latex_options_prop` and `\g_@@_latex_option_types_prop` property lists, respectively.

```

2320 \prop_new:N \g_@@_latex_option_types_prop
2321 \prop_new:N \g_@@_default_latex_options_prop
2322 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
2323 \seq_gput_right:NV \g_@@_option_layers_seq \c_@@_option_layer_latex_tl
2324 \cs_new:Nn
2325   \@@_add_latex_option:nnn
2326   {
2327     \@@_add_option:Vnnn
2328     \c_@@_option_layer_latex_tl
2329     { #1 }
2330     { #2 }
2331     { #3 }
2332   }

```

2.3.2.1 No default token renderer prototypes Default token renderer prototypes require L^AT_EX packages that may clash with other packages used in a document. Additionally, if we redefine token renderers and renderer prototypes ourselves, the default definitions will bring no benefit to us. Using the `plain` package option, we can keep the default definitions from the plain T_EX implementation (see Section 3.2.2) and prevent the soft L^AT_EX prerequisites in Section 1.1.3 from being loaded: The `plain` option must be set before or when loading the package. Setting the option after loading the package will have no effect.

```

\usepackage[plain]{markdown}

```

```

2333 \@@_add_latex_option:nnn
2334   { plain }
2335   { boolean }
2336   { false }
2337 \ExplSyntaxOff

```


2.3.2.2 L^AT_EX themes User-defined L^AT_EX themes for the Markdown package provide a domain-specific interpretation of Markdown tokens. Similarly to L^AT_EX packages, themes allow the authors to achieve a specific look and other high-level goals without low-level programming.

The L^AT_EX option `theme=<theme name>` loads a L^AT_EX package (further referred to as a *theme*) named `markdowntheme<munged theme name>.sty`, 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, and a value is qualified if and only if it contains at least one forward slash. Themes are inspired by the Beamer L^AT_EX 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 intended for a single L^AT_EX document class or for a single L^AT_EX package. The preferred format of a theme name is `<theme author>/<target LATEX document class or package>/<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, because L^AT_EX packages are identified only by their filenames, not by their pathnames. [9] Therefore, we can't store the qualified theme names directly using directories, but we must encode the individual segments of the qualified theme in the filename. For example, loading a theme named `witiko/beamer/MU` would load a L^AT_EX package named `markdownthemewitiko_beamer_MU.sty`.

If the L^AT_EX option with key `theme` 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^AT_EX 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^AT_EX package, and finally the `markdownthemewitiko_dot.sty` L^AT_EX package:

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

```
2338 \newif\ifmarkdownLaTeXLoaded
2339 \markdownLaTeXLoadedfalse
2340 \AtEndOfPackage{\markdownLaTeXLoadedtrue}
2341 \ExplSyntaxOn
2342 \tl_new:N \markdownLaTeXThemePackageName
```

```

2343 \cs_new:Nn
2344   \@@_set_latex_theme:n
2345   {
2346     \str_if_in:nnF
2347       { #1 }
2348       { / }
2349       {
2350         \markdownError
2351         { Won't~load~theme~with~unqualified~name~#1 }
2352         { Theme~names~must~contain~at~least~one~forward~slash }
2353       }
2354     \str_if_in:nnT
2355       { #1 }
2356       { _ }
2357       {
2358         \markdownError
2359         { Won't~load~theme~with~an~underscore~in~its~name~#1 }
2360         { Theme~names~must~not~contain~underscores~in~their~names }
2361       }
2362     \tl_set:Nn \markdownLaTeXThemePackageName { #1 }
2363     \str_replace_all:Nnn
2364       \markdownLaTeXThemePackageName
2365       { / }
2366       { _ }
2367     \edef\markdownLaTeXThemePackageName{
2368       markdowntheme\markdownLaTeXThemePackageName}
2369     \expandafter\markdownLaTeXThemeLoad\expandafter{
2370       \markdownLaTeXThemePackageName}{#1/}
2371   }
2372 \keys_define:nn
2373   { markdown/latex-options }
2374   {
2375     theme .code:n = { \@@_set_latex_theme:n { #1 } },
2376   }
2377 \ExplSyntaxOff

```

The \LaTeX themes have a useful synergy with the setup snippets (see Section 2.3.2): To make it less likely that different themes will define setup snippets with the same name, we will prepend $\langle theme\ name \rangle/$ before the snippet name and use the result as the snippet name. For example, if the `witiko/dot` theme defines the `product` setup snippet, the setup snippet will be available under the name `witiko/dot/product`. Due to limitations of \LaTeX , themes may not be loaded after the beginning of a \LaTeX document.

```

2378 \ExplSyntaxOn
2379 \@onlypreamble
2380   \@@_set_latex_theme:n
2381 \ExplSyntaxOff

```

Example 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[theme=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.

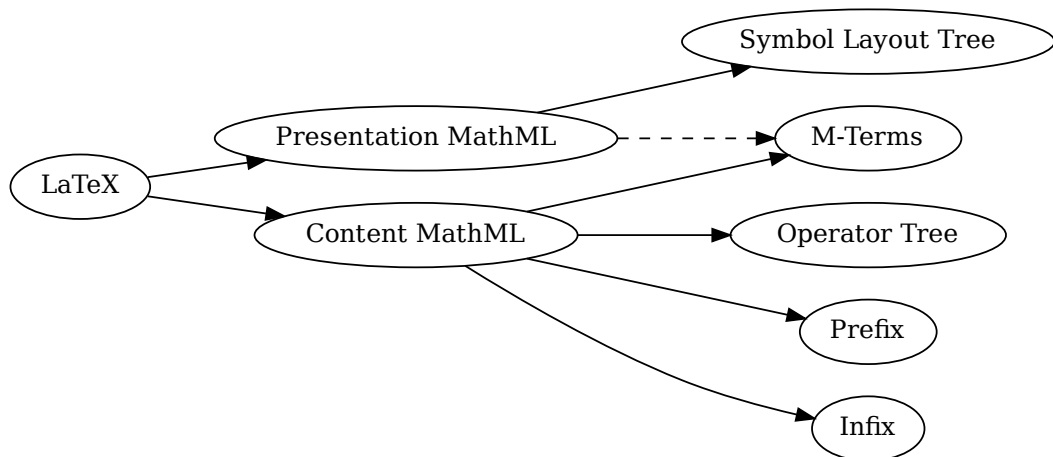


Figure 4: Various formats of mathematical formulae

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.

```
2382 \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[theme=witiko/graphicx/http]{markdown}
\begin{document}
\begin{markdown}
![img](https://github.com/witiko/markdown/raw/main/markdown.png
      "The banner of the Markdown package")
\end{markdown}
\end{document}
  
```

Typesetting the above document produces the output shown in Figure 5. The theme requires the catchfile \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.

```
2383 \ProvidesPackage{markdownthemewitiko_graphicx_http}[2021/03/22]%
```

```

\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

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

```

\documentclass{article}
\usepackage[theme=witiko/tilde]{markdown}
\begin{document}
\begin{markdown}
Bartel~Leendert van~der~Waerden
\end{markdown}
\end{document}

```

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

2384 \ProvidesPackage{markdownthemewitiko_tilde}[2021/03/22]%

Please, see Section 3.3.2.1 for implementation details of the example themes.

2.3.2.3 L^AT_EX setup snippets The L^AT_EX option with key **snippet** invokes a snippet named *<value>*:

2385 \ExplSyntaxOn

```

2386 \keys_define:nn
2387   { markdown/latex-options }
2388   {
2389     snippet .code:n = {
2390       \markdownIfSnippetExists{#1}
2391       {
2392         \expandafter\markdownSetup\expandafter{
2393           \the\csname markdownLaTeXSetupSnippet
2394             \markdownLaTeXThemeName#1\endcsname}
2395         }{
2396           \markdownError
2397             {Can't-invoke-setup~snippet~#1}
2398             {The~setup~snippet~is~undefined}
2399         }
2400       }
2401     }
2402 \ExplSyntaxOff

```

Here is how we can use setup snippets to store options and invoke them later:

```

\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*}

```

2.3.2.4 Plain T_EX Interface Options Here, we automatically define plain T_EX macros and the $\langle key \rangle = \langle value \rangle$ interface for the above L^AT_EX options.

```

2403 \ExplSyntaxOn
2404 \cs_new:Nn \@@_latex_define_option_commands_and_keyvals:
2405 {
2406   \seq_map_inline:Nn
2407     \g_@@_latex_options_seq
2408     {
2409       \@@_plain_tex_define_option_command:n
2410       { ##1 }
2411     }

```

Furthermore, we also define the $\langle key \rangle = \langle value \rangle$ interface for all option macros recognized by the Lua and plain T_EX interfaces.

```

2412   \seq_map_inline:Nn
2413     \g_@@_option_layers_seq
2414     {
2415       \seq_map_inline:cn
2416       { g_@@_ ##1 _options_seq }
2417     }

```

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`.

```

2418       \@@_with_various_cases:nn
2419       { #####1 }
2420     {
2421       \@@_latex_define_option_keyval:nnn
2422       { ##1 }
2423       { #####1 }
2424       { #####1 }
2425     }
2426   }
2427 }
2428 }
2429 \cs_new:Nn \@@_latex_define_option_keyval:nnn
2430 {
2431   \prop_get:cnN
2432     { g_@@_ #1 _option_types_prop }
2433     { #2 }
2434     \l_tmpa_tl
2435   \keys_define:nn
2436     { markdown/latex-options }
2437     {
2438       #3 .code:n = {
2439         \@@_set_option_value:nn
2440         { #2 }
2441         { ##1 }

```

```

2442     },
2443   }
2444   \str_if_eq:VVT
2445     \l_tmpa_tl
2446     \c_@@_option_type_boolean_tl
2447     {
2448       \keys_define:nn
2449         { markdown/latex-options }
2450         {
2451           #3 .default:n = { true },
2452         }
2453     }

```

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.

```

2454   \str_if_eq:VVT
2455     \l_tmpa_tl
2456     \c_@@_option_type_clist_tl
2457     {
2458       \tl_set:Nn
2459         \l_tmpa_tl
2460         { #3 }
2461       \tl_reverse:N
2462         \l_tmpa_tl
2463       \str_if_eq:enF
2464         {
2465           \tl_head:V
2466             \l_tmpa_tl
2467         }
2468         { s }
2469         {
2470           \msg_error:nnn
2471             { @@ }
2472             { malformed-name-for-clist-option }
2473             { #3 }
2474         }
2475       \tl_set:Nx
2476         \l_tmpa_tl
2477         {
2478           \tl_tail:V
2479             \l_tmpa_tl
2480         }
2481       \tl_reverse:N
2482         \l_tmpa_tl

```



```

2483     \tl_put_right:Nn
2484     \l_tmpa_tl
2485     {
2486         .code:n = {
2487             \@@_get_option_value:nN
2488             { #2 }
2489             \l_tmpa_tl
2490             \clist_set:NV
2491             \l_tmpa_clist
2492             { \l_tmpa_tl, { ##1 } }
2493             \@@_set_option_value:nV
2494             { #2 }
2495             \l_tmpa_clist
2496         }
2497     }
2498     \keys_define:nV
2499     { markdown/latex-options }
2500     \l_tmpa_tl
2501 }
2502 }
2503 \cs_generate_variant:Nn
2504 \clist_set:Nn
2505 { NV }
2506 \cs_generate_variant:Nn
2507 \keys_define:nn
2508 { nV }
2509 \cs_generate_variant:Nn
2510 \@@_set_option_value:nn
2511 { nV }
2512 \prg_generate_conditional_variant:Nnn
2513 \str_if_eq:nn
2514 { en }
2515 { F }
2516 \msg_new:nnn
2517 { @@ }
2518 { malformed-name-for-clist-option }
2519 {
2520     Clist~option~name~#1~does~not~end~with~-s.
2521 }
2522 \@@_latex_define_option_commands_and_keyvals:
2523 \ExplSyntaxOff

```

The `finalizeCache` and `frozenCache` plain TeX options are exposed through L^AT_EX options with keys `finalizeCache` and `frozenCache`.

To ensure compatibility with the `minted` package [10, Section 5.1], which supports the `finalizcache` and `frozenscache` package options with similar semantics, the Markdown package also recognizes these as aliases and recognizes them as document

class options. By passing `finalizcache` and `frozencache` as document class options, you may conveniently control the behavior of both packages at once:

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

We hope that other packages will support the `finalizcache` and `frozencache` package options in the future, so that they can become a standard interface for preparing L^AT_EX document sources for distribution.

```
2524 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
2525 \DeclareOption{frozencache}{\markdownSetup{frozenCache}}
```

The following example L^AT_EX code showcases a possible configuration of plain T_EX interface options `hybrid`, `smartEllipses`, and `cacheDir`.

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

2.3.2.5 Plain T_EX Markdown Token Renderers The L^AT_EX interface recognizes an option with the `renderers` key, whose value must be a list of options that map directly to the markdown token renderer macros exposed by the plain T_EX interface (see Section 2.2.3).

```
2526 \ExplSyntaxOn
2527 \cs_new:Nn \@@_latex_define_renderers:
2528   {
2529     \seq_map_function:NN
2530       \g_@@_renderers_seq
2531       \@@_latex_define_renderer:n
2532   }
2533 \cs_new:Nn \@@_latex_define_renderer:n
2534   {
2535     \@@_renderer_tl_to_csname:nN
2536       { #1 }
2537     \l_tmpa_tl
2538     \prop_get:NnN
2539       \g_@@_renderer_arities_prop
2540       { #1 }
2541     \l_tmpb_tl
```

```

2542 \@@_latex_define_renderer:ncV
2543 { #1 }
2544 { \l_tmpa_tl }
2545 \l_tmpb_tl
2546 }
2547 \cs_new:Nn \@@_renderer_tl_to_csname:nN
2548 {
2549 \tl_set:Nn
2550 \l_tmpa_tl
2551 { \str_uppercase:n { #1 } }
2552 \tl_set:Nx
2553 #2
2554 {
2555 markdownRenderer
2556 \tl_head:f { \l_tmpa_tl }
2557 \tl_tail:n { #1 }
2558 }
2559 }
2560 \cs_new:Nn \@@_latex_define_renderer:nNn
2561 {
2562 \@@_with_various_cases:nn
2563 { #1 }
2564 {
2565 \keys_define:nn
2566 { markdown/latex-options/renderers }
2567 {
2568 ##1 .code:n = {
2569 \cs_generate_from_arg_count:NNnn
2570 #2
2571 \cs_set:Npn
2572 { #3 }
2573 { #####1 }
2574 },
2575 }
2576 }
2577 }
2578 \cs_generate_variant:Nn
2579 \@@_latex_define_renderer:nNn
2580 { ncV }
2581 \ExplSyntaxOff

```

The following example L^AT_EX code showcases a possible configuration of the `\markdownRendererLink` and `\markdownRendererEmphasis` markdown token renderers.

```

\markdownSetup{
  renderers = {
    link = {#4}, % Render links as the link title.

```

```

    emphasis = {\emph{#1}},    % Render emphasized text via \emph`.
  }
}

```

2.3.2.6 Plain T_EX Markdown Token Renderer Prototypes The L^AT_EX interface recognizes an option with the `rendererPrototypes` key, whose value must be a list of options that map directly to the markdown token renderer prototype macros exposed by the plain T_EX interface (see Section 2.2.4).

```

2582 \ExplSyntaxOn
2583 \cs_new:Nn \@@_latex_define_renderer_prototypes:
2584 {
2585   \seq_map_function:NN
2586     \g_@@_renderers_seq
2587     \@@_latex_define_renderer_prototype:n
2588 }
2589 \cs_new:Nn \@@_latex_define_renderer_prototype:n
2590 {
2591   \@@_renderer_prototype_tl_to_csname:nN
2592     { #1 }
2593     \l_tmpa_tl
2594   \prop_get:NnN
2595     \g_@@_renderer_arities_prop
2596     { #1 }
2597     \l_tmpb_tl
2598   \@@_latex_define_renderer_prototype:ncV
2599     { #1 }
2600     { \l_tmpa_tl }
2601     \l_tmpb_tl
2602 }
2603 \cs_new:Nn \@@_latex_define_renderer_prototype:nNn
2604 {
2605   \@@_with_various_cases:nn
2606     { #1 }
2607     {
2608       \keys_define:nn
2609         { markdown/latex-options/renderer-prototypes }
2610         {
2611           ##1 .code:n = {
2612             \cs_generate_from_arg_count:NNnn
2613               #2
2614               \cs_set:Npn
2615                 { #3 }
2616                 { #####1 }
2617             },
2618         }

```

```

2619     }
2620   }
2621   \cs_generate_variant:Nn
2622   \@@_latex_define_renderer_prototype:nNn
2623   { ncV }
2624   \ExplSyntaxOff

```

The following example \LaTeX code showcases a possible configuration of the `\markdownRendererImagePrototype` and `\markdownRendererCodeSpanPrototype` markdown token renderer prototypes.

```

\markdownSetup{
  rendererPrototypes = {
    image = {\includegraphics{#2}},
    codeSpan = {\texttt{#1}}, % Render inline code via `texttt`.
  }
}

```

2.4 ConTeXt Interface

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).

```

2625 \writestatus{loading}{ConTeXt User Module / markdown}%
2626 \startmodule[markdown]
2627 \unprotect

```

The ConTeXt implementation redefines the plain \TeX logging macros (see Section 3.2.1) to use the ConTeXt `\writestatus` macro.

```

2628 \def\markdownInfo#1{\writestatus{markdown}{#1.}}%
2629 \def\markdownWarning#1{\writestatus{markdown\space warn}{#1.}}%
2630 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
2631   \do#\do\^\do\_do\%do\~}%
2632 \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` command.

```
2633 \let\startmarkdown\relax
2634 \let\stopmarkdown\relax
2635 \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), or via the `\setupmarkdown` macro. The `\setupmarkdown` macro receives the options to set up as its only argument:

```

2636 \ExplSyntaxOn
2637 \cs_new:Nn
2638   \@@_setup:n
2639   {
2640     \keys_set:nn
2641       { markdown/context-options }
2642       { #1 }
2643   }
2644 \long\def\setupmarkdown[#1]
2645   {
2646     \@@_setup:n
2647       { #1 }
2648   }
2649 \ExplSyntaxOff

```

2.4.2.1 ConTeXt Interface Options We define the $\langle key \rangle = \langle value \rangle$ interface for all option macros recognized by the Lua and plain TeX interfaces.

```

2650 \ExplSyntaxOn
2651 \cs_new:Nn \@@_context_define_option_commands_and_keyvals:
2652   {
2653     \seq_map_inline:Nn
2654       \g_@@_option_layers_seq
2655       {
2656         \seq_map_inline:cn
2657           { g_@@_ ##1 _options_seq }
2658           {

```

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`.

```

2659         \@@_with_various_cases:nn
2660         { #####1 }
2661         {
2662           \@@_context_define_option_keyval:nnn
2663             { ##1 }
2664             { #####1 }
2665             { #####1##1 }
2666         }
2667     }
2668 }

```

```
2669 }
```

Furthermore, we also accept caseless variants of options in line with the style of ConTeXt.

```
2670 \cs_new:Nn \@@_caseless:N
2671 {
2672   \regex_replace_all:nnN
2673     { ([a-z])([A-Z]) }
2674     { \1 \c { str_lowercase:n } \cB\{ \2 \cE\} }
2675     #1
2676   \tl_set:Nx
2677     #1
2678     { #1 }
2679 }
2680 \seq_gput_right:Nn \g_@@_cases_seq { @@_caseless:N }
2681 \cs_new:Nn \@@_context_define_option_keyval:nnn
2682 {
2683   \prop_get:cnN
2684     { g_@@_ #1 _option_types_prop }
2685     { #2 }
2686   \l_tmpa_tl
2687   \keys_define:nn
2688     { markdown/context-options }
2689     {
2690       #3 .code:n = {
2691         \tl_set:Nx
2692           \l_tmpa_tl
2693           {
2694             \str_case:nnF
2695               { ##1 }
2696               {
2697                 { yes } { true }
2698                 { no } { false }
2699               }
2700             { ##1 }
2701           }
2702         \@@_set_option_value:nV
2703           { #2 }
2704         \l_tmpa_tl
2705       },
2706     }
2707   \str_if_eq:VVT
2708     \l_tmpa_tl
2709     \c_@@_option_type_boolean_tl
2710     {
2711       \keys_define:nn
2712         { markdown/context-options }
```



```

2713         {
2714         #3 .default:n = { true },
2715         }
2716     }
2717 }
2718 \cs_generate_variant:Nn
2719   \@@_set_option_value:nn
2720   { nV }
2721 \@@_context_define_option_commands_and_keyvals:
2722 \ExplSyntaxOff

```

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 \TeX *token renderers* is performed by the Lua layer. The plain \TeX layer provides default definitions for the token renderers. The \LaTeX and \ConTeXt layers correct idiosyncrasies of the respective \TeX 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 \TeX , 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 \TeX writer were hidden behind the converter functions exposed by the Lua interface (see Section 2.1).

```

2723 local upper, format, length =
2724   string.upper, string.format, string.len
2725 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, any =
2726   lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
2727   lpeg.Cmt, lpeg.Cc, lpeg.Ct, lpeg.B, lpeg.Cs, lpeg.P(1)

```

3.1.1 Utility Functions

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

```
2728 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.

```
2729 function util.err(msg, exit_code)
2730   io.stderr:write("markdown.lua: " .. msg .. "\n")
2731   os.exit(exit_code or 1)
2732 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.

```
2733 function util.cache(dir, string, salt, transform, suffix)
2734   local digest = md5.sumhexa(string .. (salt or ""))
2735   local name = util.pathname(dir, digest .. suffix)
2736   local file = io.open(name, "r")
2737   if file == nil then -- If no cache entry exists, then create a new one.
2738     file = assert(io.open(name, "w"),
2739       [[Could not open file ]] .. name .. [[ for writing]])
2740     local result = string
2741     if transform ~= nil then
2742       result = transform(result)
2743     end
2744     assert(file:write(result))
2745     assert(file:close())
2746   end
2747   return name
2748 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.

```
2749 function util.cache_verbatim(dir, string)
2750   local name = util.cache(dir, string, nil, nil, ".verbatim")
2751   return name
2752 end
```

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

```
2753 function util.table_copy(t)
2754   local u = { }
2755   for k, v in pairs(t) do u[k] = v end
2756   return setmetatable(u, getmetatable(t))
2757 end
```

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

```
2758 function util.encode_json_string(s)
2759   s = s:gsub([[\\]], [[\\]])
```

```

2760  s = s:gsub([""], [{"\"}])
2761  return [{""]} .. s .. [{""]}
2762 end

```

The `util.lookup_files` method looks up files with filename `f` and returns its path. If the `kpathsea` library is available, it will search for files not only in the current working directory but also in the `TEX` directory structure. Further options for `kpathsea` can be specified in table `options`. [1, Section 10.7.4]

```

2763 util.lookup_files = (function()
2764   local ran_ok, kpse = pcall(require, "kpse")
2765   if ran_ok then
2766     kpse.set_program_name("luatex")
2767   else
2768     kpse = { lookup = function(f, _) return f end }
2769   end
2770
2771   local function lookup_files(f, options)
2772     return kpse.lookup(f, options)
2773   end
2774
2775   return lookup_files
2776 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].

```

2777 function util.expand_tabs_in_line(s, tabstop)
2778   local tab = tabstop or 4
2779   local corr = 0
2780   return (s:gsub("\t", function(p)
2781     local sp = tab - (p - 1 + corr) % tab
2782     corr = corr - 1 + sp
2783     return string.rep(" ", sp)
2784   end))
2785 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.

```

2786 function util.walk(t, f)
2787   local typ = type(t)
2788   if typ == "string" then
2789     f(t)
2790   elseif typ == "table" then
2791     local i = 1

```

```

2792     local n
2793     n = t[i]
2794     while n do
2795         util.walk(n, f)
2796         i = i + 1
2797         n = t[i]
2798     end
2799     elseif typ == "function" then
2800         local ok, val = pcall(t)
2801         if ok then
2802             util.walk(val,f)
2803         end
2804     else
2805         f(tostring(t))
2806     end
2807 end

```

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

```

2808 function util.flatten(ary)
2809     local new = {}
2810     for _,v in ipairs(ary) do
2811         if type(v) == "table" then
2812             for _,w in ipairs(util.flatten(v)) do
2813                 new[#new + 1] = w
2814             end
2815         else
2816             new[#new + 1] = v
2817         end
2818     end
2819     return new
2820 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.

```

2821 function util.rope_to_string(rope)
2822     local buffer = {}
2823     util.walk(rope, function(x) buffer[#buffer + 1] = x end)
2824     return table.concat(buffer)
2825 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.

```

2826 function util.rope_last(rope)
2827     if #rope == 0 then
2828         return nil
2829     else
2830         local l = rope[#rope]

```

```

2831     if type(l) == "table" then
2832         return util.rope_last(l)
2833     else
2834         return l
2835     end
2836 end
2837 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 $1 \leq i \leq \#ary$.

```

2838 function util.intersperse(ary, x)
2839     local new = {}
2840     local l = #ary
2841     for i,v in ipairs(ary) do
2842         local n = #new
2843         new[n + 1] = v
2844         if i ~= l then
2845             new[n + 2] = x
2846         end
2847     end
2848     return new
2849 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 $1 \leq i \leq \#ary$.

```

2850 function util.map(ary, f)
2851     local new = {}
2852     for i,v in ipairs(ary) do
2853         new[i] = f(v)
2854     end
2855     return new
2856 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.

```

2857 function util.escaper(char_escapes, string_escapes)

```

Build a string of escapable characters.

```

2858     local char_escapes_list = ""
2859     for i,_ in pairs(char_escapes) do
2860         char_escapes_list = char_escapes_list .. i
2861     end

```

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

```
2862 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.

```
2863 if string_escapes then
2864   for k,v in pairs(string_escapes) do
2865     escapable = P(k) / v + escapable
2866   end
2867 end
```

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

```
2868 local escape_string = Cs((escapable + any)^0)
```

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

```
2869 return function(s)
2870   return lpeg.match(escape_string, s)
2871 end
2872 end
```

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

```
2873 function util.pathname(dir, file)
2874   if #dir == 0 then
2875     return file
2876   else
2877     return dir .. "/" .. file
2878   end
2879 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.

```
2880 local entities = {}
2881
```

```
2882 local character_entities = {
2883     ["Tab"] = 9,
2884     ["NewLine"] = 10,
2885     ["excl"] = 33,
2886     ["quot"] = 34,
2887     ["QUOT"] = 34,
2888     ["num"] = 35,
2889     ["dollar"] = 36,
2890     ["percent"] = 37,
2891     ["amp"] = 38,
2892     ["AMP"] = 38,
2893     ["apos"] = 39,
2894     ["lpar"] = 40,
2895     ["rpar"] = 41,
2896     ["ast"] = 42,
2897     ["midast"] = 42,
2898     ["plus"] = 43,
2899     ["comma"] = 44,
2900     ["period"] = 46,
2901     ["sol"] = 47,
2902     ["colon"] = 58,
2903     ["semi"] = 59,
2904     ["lt"] = 60,
2905     ["LT"] = 60,
2906     ["equals"] = 61,
2907     ["gt"] = 62,
2908     ["GT"] = 62,
2909     ["quest"] = 63,
2910     ["commat"] = 64,
2911     ["lsqb"] = 91,
2912     ["lbrack"] = 91,
2913     ["bsol"] = 92,
2914     ["rsqb"] = 93,
2915     ["rbrack"] = 93,
2916     ["Hat"] = 94,
2917     ["lowbar"] = 95,
2918     ["grave"] = 96,
2919     ["DiacriticalGrave"] = 96,
2920     ["lcub"] = 123,
2921     ["lbrace"] = 123,
2922     ["verbar"] = 124,
2923     ["vert"] = 124,
2924     ["VerticalLine"] = 124,
2925     ["rcub"] = 125,
2926     ["rbrace"] = 125,
2927     ["nbsp"] = 160,
2928     ["NonBreakingSpace"] = 160,
```

2929 ["iexcl"] = 161,
2930 ["cent"] = 162,
2931 ["pound"] = 163,
2932 ["curren"] = 164,
2933 ["yen"] = 165,
2934 ["brvbar"] = 166,
2935 ["sect"] = 167,
2936 ["Dot"] = 168,
2937 ["die"] = 168,
2938 ["DoubleDot"] = 168,
2939 ["uml"] = 168,
2940 ["copy"] = 169,
2941 ["COPY"] = 169,
2942 ["ordf"] = 170,
2943 ["laquo"] = 171,
2944 ["not"] = 172,
2945 ["shy"] = 173,
2946 ["reg"] = 174,
2947 ["circledR"] = 174,
2948 ["REG"] = 174,
2949 ["macr"] = 175,
2950 ["OverBar"] = 175,
2951 ["strns"] = 175,
2952 ["deg"] = 176,
2953 ["plusmn"] = 177,
2954 ["pm"] = 177,
2955 ["PlusMinus"] = 177,
2956 ["sup2"] = 178,
2957 ["sup3"] = 179,
2958 ["acute"] = 180,
2959 ["DiacriticalAcute"] = 180,
2960 ["micro"] = 181,
2961 ["para"] = 182,
2962 ["middot"] = 183,
2963 ["centerdot"] = 183,
2964 ["CenterDot"] = 183,
2965 ["cedil"] = 184,
2966 ["Cedilla"] = 184,
2967 ["sup1"] = 185,
2968 ["ordm"] = 186,
2969 ["raquo"] = 187,
2970 ["frac14"] = 188,
2971 ["frac12"] = 189,
2972 ["half"] = 189,
2973 ["frac34"] = 190,
2974 ["iquest"] = 191,
2975 ["Agrave"] = 192,

2976 ["Acute"] = 193,
2977 ["Acirc"] = 194,
2978 ["Atilde"] = 195,
2979 ["Auml"] = 196,
2980 ["Aring"] = 197,
2981 ["AElig"] = 198,
2982 ["Ccedil"] = 199,
2983 ["Egrave"] = 200,
2984 ["Eacute"] = 201,
2985 ["Ecirc"] = 202,
2986 ["Euml"] = 203,
2987 ["Igrave"] = 204,
2988 ["Iacute"] = 205,
2989 ["Icirc"] = 206,
2990 ["Iuml"] = 207,
2991 ["ETH"] = 208,
2992 ["Ntilde"] = 209,
2993 ["Ograve"] = 210,
2994 ["Oacute"] = 211,
2995 ["Ocirc"] = 212,
2996 ["Otilde"] = 213,
2997 ["Ouml"] = 214,
2998 ["times"] = 215,
2999 ["Oslash"] = 216,
3000 ["Ugrave"] = 217,
3001 ["Uacute"] = 218,
3002 ["Ucirc"] = 219,
3003 ["Uuml"] = 220,
3004 ["Yacute"] = 221,
3005 ["THORN"] = 222,
3006 ["szlig"] = 223,
3007 ["agrave"] = 224,
3008 ["aacute"] = 225,
3009 ["acirc"] = 226,
3010 ["atilde"] = 227,
3011 ["auml"] = 228,
3012 ["aring"] = 229,
3013 ["aelig"] = 230,
3014 ["ccedil"] = 231,
3015 ["egrave"] = 232,
3016 ["eacute"] = 233,
3017 ["ecirc"] = 234,
3018 ["euml"] = 235,
3019 ["igrave"] = 236,
3020 ["iacute"] = 237,
3021 ["icirc"] = 238,
3022 ["iuml"] = 239,

3023 ["eth"] = 240,
3024 ["ntilde"] = 241,
3025 ["ograve"] = 242,
3026 ["oacute"] = 243,
3027 ["ocirc"] = 244,
3028 ["otilde"] = 245,
3029 ["ouml"] = 246,
3030 ["divide"] = 247,
3031 ["div"] = 247,
3032 ["oslash"] = 248,
3033 ["ugrave"] = 249,
3034 ["uacute"] = 250,
3035 ["ucirc"] = 251,
3036 ["uuml"] = 252,
3037 ["yacute"] = 253,
3038 ["thorn"] = 254,
3039 ["yuml"] = 255,
3040 ["Amacr"] = 256,
3041 ["amacr"] = 257,
3042 ["Abreve"] = 258,
3043 ["abreve"] = 259,
3044 ["Aogon"] = 260,
3045 ["aogon"] = 261,
3046 ["Cacute"] = 262,
3047 ["cacute"] = 263,
3048 ["Ccirc"] = 264,
3049 ["ccirc"] = 265,
3050 ["Cdot"] = 266,
3051 ["cdot"] = 267,
3052 ["Ccaron"] = 268,
3053 ["ccaron"] = 269,
3054 ["Dcaron"] = 270,
3055 ["dcaron"] = 271,
3056 ["Dstrok"] = 272,
3057 ["dstrok"] = 273,
3058 ["Emacr"] = 274,
3059 ["emacr"] = 275,
3060 ["Edot"] = 278,
3061 ["edot"] = 279,
3062 ["Eogon"] = 280,
3063 ["eogon"] = 281,
3064 ["Ecaron"] = 282,
3065 ["ecaron"] = 283,
3066 ["Gcirc"] = 284,
3067 ["gcirc"] = 285,
3068 ["Gbreve"] = 286,
3069 ["gbreve"] = 287,

3070 ["Gdot"] = 288,
3071 ["gdot"] = 289,
3072 ["Gcedil"] = 290,
3073 ["Hcirc"] = 292,
3074 ["hcirc"] = 293,
3075 ["Hstrook"] = 294,
3076 ["hstrook"] = 295,
3077 ["Itilde"] = 296,
3078 ["itilde"] = 297,
3079 ["Imacr"] = 298,
3080 ["imacr"] = 299,
3081 ["Iogon"] = 302,
3082 ["iogon"] = 303,
3083 ["Idot"] = 304,
3084 ["imath"] = 305,
3085 ["inodot"] = 305,
3086 ["IJlig"] = 306,
3087 ["ijlig"] = 307,
3088 ["Jcirc"] = 308,
3089 ["jcirc"] = 309,
3090 ["Kcedil"] = 310,
3091 ["kcedil"] = 311,
3092 ["kgreen"] = 312,
3093 ["Lacute"] = 313,
3094 ["lacute"] = 314,
3095 ["Lcedil"] = 315,
3096 ["lcedil"] = 316,
3097 ["Lcaron"] = 317,
3098 ["lcaron"] = 318,
3099 ["Lmidot"] = 319,
3100 ["lmidot"] = 320,
3101 ["Lstrook"] = 321,
3102 ["lstrook"] = 322,
3103 ["Nacute"] = 323,
3104 ["nacute"] = 324,
3105 ["Ncedil"] = 325,
3106 ["ncedil"] = 326,
3107 ["Ncaron"] = 327,
3108 ["ncaron"] = 328,
3109 ["napos"] = 329,
3110 ["ENG"] = 330,
3111 ["eng"] = 331,
3112 ["Omacr"] = 332,
3113 ["omacr"] = 333,
3114 ["Odblac"] = 336,
3115 ["odblac"] = 337,
3116 ["OElig"] = 338,

3117 ["oelig"] = 339,
3118 ["Racute"] = 340,
3119 ["racute"] = 341,
3120 ["Rcedil"] = 342,
3121 ["rcedil"] = 343,
3122 ["Rcaron"] = 344,
3123 ["rcaron"] = 345,
3124 ["Sacute"] = 346,
3125 ["sacute"] = 347,
3126 ["Scirc"] = 348,
3127 ["scirc"] = 349,
3128 ["Scedil"] = 350,
3129 ["scedil"] = 351,
3130 ["Scaron"] = 352,
3131 ["scaron"] = 353,
3132 ["Tcedil"] = 354,
3133 ["tcedil"] = 355,
3134 ["Tcaron"] = 356,
3135 ["tcaron"] = 357,
3136 ["Tstrok"] = 358,
3137 ["tstrok"] = 359,
3138 ["Utilde"] = 360,
3139 ["utilde"] = 361,
3140 ["Umacr"] = 362,
3141 ["umacr"] = 363,
3142 ["Ubreve"] = 364,
3143 ["ubreve"] = 365,
3144 ["Uring"] = 366,
3145 ["uring"] = 367,
3146 ["Udblac"] = 368,
3147 ["udblac"] = 369,
3148 ["Uogon"] = 370,
3149 ["uogon"] = 371,
3150 ["Wcirc"] = 372,
3151 ["wcirc"] = 373,
3152 ["Ycirc"] = 374,
3153 ["ycirc"] = 375,
3154 ["Yuml"] = 376,
3155 ["Zacute"] = 377,
3156 ["zacute"] = 378,
3157 ["Zdot"] = 379,
3158 ["zdot"] = 380,
3159 ["Zcaron"] = 381,
3160 ["zcaron"] = 382,
3161 ["fnof"] = 402,
3162 ["imped"] = 437,
3163 ["gacute"] = 501,

3164 ["jmath"] = 567,
3165 ["circ"] = 710,
3166 ["caron"] = 711,
3167 ["Hacek"] = 711,
3168 ["breve"] = 728,
3169 ["Breve"] = 728,
3170 ["dot"] = 729,
3171 ["DiacriticalDot"] = 729,
3172 ["ring"] = 730,
3173 ["ogon"] = 731,
3174 ["tilde"] = 732,
3175 ["DiacriticalTilde"] = 732,
3176 ["dblac"] = 733,
3177 ["DiacriticalDoubleAcute"] = 733,
3178 ["DownBreve"] = 785,
3179 ["UnderBar"] = 818,
3180 ["Alpha"] = 913,
3181 ["Beta"] = 914,
3182 ["Gamma"] = 915,
3183 ["Delta"] = 916,
3184 ["Epsilon"] = 917,
3185 ["Zeta"] = 918,
3186 ["Eta"] = 919,
3187 ["Theta"] = 920,
3188 ["Iota"] = 921,
3189 ["Kappa"] = 922,
3190 ["Lambda"] = 923,
3191 ["Mu"] = 924,
3192 ["Nu"] = 925,
3193 ["Xi"] = 926,
3194 ["Omicron"] = 927,
3195 ["Pi"] = 928,
3196 ["Rho"] = 929,
3197 ["Sigma"] = 931,
3198 ["Tau"] = 932,
3199 ["Upsilon"] = 933,
3200 ["Phi"] = 934,
3201 ["Chi"] = 935,
3202 ["Psi"] = 936,
3203 ["Omega"] = 937,
3204 ["alpha"] = 945,
3205 ["beta"] = 946,
3206 ["gamma"] = 947,
3207 ["delta"] = 948,
3208 ["epsiv"] = 949,
3209 ["varepsilon"] = 949,
3210 ["epsilon"] = 949,

3211 ["zeta"] = 950,
 3212 ["eta"] = 951,
 3213 ["theta"] = 952,
 3214 ["iota"] = 953,
 3215 ["kappa"] = 954,
 3216 ["lambda"] = 955,
 3217 ["mu"] = 956,
 3218 ["nu"] = 957,
 3219 ["xi"] = 958,
 3220 ["omicron"] = 959,
 3221 ["pi"] = 960,
 3222 ["rho"] = 961,
 3223 ["sigmav"] = 962,
 3224 ["varsigma"] = 962,
 3225 ["sigmaf"] = 962,
 3226 ["sigma"] = 963,
 3227 ["tau"] = 964,
 3228 ["upsi"] = 965,
 3229 ["upsilon"] = 965,
 3230 ["phi"] = 966,
 3231 ["phiv"] = 966,
 3232 ["varphi"] = 966,
 3233 ["chi"] = 967,
 3234 ["psi"] = 968,
 3235 ["omega"] = 969,
 3236 ["thetav"] = 977,
 3237 ["vartheta"] = 977,
 3238 ["thetasym"] = 977,
 3239 ["Upsi"] = 978,
 3240 ["upsih"] = 978,
 3241 ["straightphi"] = 981,
 3242 ["piv"] = 982,
 3243 ["varpi"] = 982,
 3244 ["Gammad"] = 988,
 3245 ["gammad"] = 989,
 3246 ["digamma"] = 989,
 3247 ["kappav"] = 1008,
 3248 ["varkappa"] = 1008,
 3249 ["rhov"] = 1009,
 3250 ["varrho"] = 1009,
 3251 ["epsi"] = 1013,
 3252 ["straightepsilon"] = 1013,
 3253 ["bepsi"] = 1014,
 3254 ["backepsilon"] = 1014,
 3255 ["IOcy"] = 1025,
 3256 ["DJcy"] = 1026,
 3257 ["GJcy"] = 1027,

3258 ["Jukcy"] = 1028,
3259 ["DScy"] = 1029,
3260 ["Iukcy"] = 1030,
3261 ["YIcy"] = 1031,
3262 ["Jsercy"] = 1032,
3263 ["LJcy"] = 1033,
3264 ["NJcy"] = 1034,
3265 ["TSHcy"] = 1035,
3266 ["KJcy"] = 1036,
3267 ["Ubrcy"] = 1038,
3268 ["DZcy"] = 1039,
3269 ["Acy"] = 1040,
3270 ["Bcy"] = 1041,
3271 ["Vcy"] = 1042,
3272 ["Gcy"] = 1043,
3273 ["Dcy"] = 1044,
3274 ["IEcy"] = 1045,
3275 ["ZHcy"] = 1046,
3276 ["Zcy"] = 1047,
3277 ["Icy"] = 1048,
3278 ["Jcy"] = 1049,
3279 ["Kcy"] = 1050,
3280 ["Lcy"] = 1051,
3281 ["Mcy"] = 1052,
3282 ["Ncy"] = 1053,
3283 ["Ocy"] = 1054,
3284 ["Pcy"] = 1055,
3285 ["Rcy"] = 1056,
3286 ["Scy"] = 1057,
3287 ["Tcy"] = 1058,
3288 ["Ucy"] = 1059,
3289 ["Fcy"] = 1060,
3290 ["KHcy"] = 1061,
3291 ["TScy"] = 1062,
3292 ["CHcy"] = 1063,
3293 ["SHcy"] = 1064,
3294 ["SHCHcy"] = 1065,
3295 ["HARDcy"] = 1066,
3296 ["Ycy"] = 1067,
3297 ["SOFTcy"] = 1068,
3298 ["Ecy"] = 1069,
3299 ["YUcy"] = 1070,
3300 ["YAcy"] = 1071,
3301 ["acy"] = 1072,
3302 ["bcy"] = 1073,
3303 ["vcy"] = 1074,
3304 ["gcy"] = 1075,

3305 ["dcy"] = 1076,
3306 ["iecy"] = 1077,
3307 ["zhcy"] = 1078,
3308 ["zcy"] = 1079,
3309 ["icy"] = 1080,
3310 ["jcy"] = 1081,
3311 ["kcy"] = 1082,
3312 ["lcy"] = 1083,
3313 ["mcy"] = 1084,
3314 ["ncy"] = 1085,
3315 ["ocy"] = 1086,
3316 ["pcy"] = 1087,
3317 ["rcy"] = 1088,
3318 ["scy"] = 1089,
3319 ["tcy"] = 1090,
3320 ["ucy"] = 1091,
3321 ["fcy"] = 1092,
3322 ["khcy"] = 1093,
3323 ["tscy"] = 1094,
3324 ["chcy"] = 1095,
3325 ["shcy"] = 1096,
3326 ["shchcy"] = 1097,
3327 ["hardcy"] = 1098,
3328 ["ycy"] = 1099,
3329 ["softcy"] = 1100,
3330 ["ecy"] = 1101,
3331 ["yucy"] = 1102,
3332 ["yacy"] = 1103,
3333 ["iocy"] = 1105,
3334 ["djcy"] = 1106,
3335 ["gjcy"] = 1107,
3336 ["jukcy"] = 1108,
3337 ["dscy"] = 1109,
3338 ["iukcy"] = 1110,
3339 ["yicy"] = 1111,
3340 ["jsercy"] = 1112,
3341 ["ljcy"] = 1113,
3342 ["njcy"] = 1114,
3343 ["tshcy"] = 1115,
3344 ["kjcy"] = 1116,
3345 ["ubrcy"] = 1118,
3346 ["dzcy"] = 1119,
3347 ["ensp"] = 8194,
3348 ["emsp"] = 8195,
3349 ["emsp13"] = 8196,
3350 ["emsp14"] = 8197,
3351 ["numsp"] = 8199,

3352 ["puncsp"] = 8200,
3353 ["thinsp"] = 8201,
3354 ["ThinSpace"] = 8201,
3355 ["hairsp"] = 8202,
3356 ["VeryThinSpace"] = 8202,
3357 ["ZeroWidthSpace"] = 8203,
3358 ["NegativeVeryThinSpace"] = 8203,
3359 ["NegativeThinSpace"] = 8203,
3360 ["NegativeMediumSpace"] = 8203,
3361 ["NegativeThickSpace"] = 8203,
3362 ["zwnj"] = 8204,
3363 ["zwj"] = 8205,
3364 ["lrm"] = 8206,
3365 ["rlm"] = 8207,
3366 ["hyphen"] = 8208,
3367 ["dash"] = 8208,
3368 ["ndash"] = 8211,
3369 ["mdash"] = 8212,
3370 ["horbar"] = 8213,
3371 ["Verbar"] = 8214,
3372 ["Vert"] = 8214,
3373 ["lsquo"] = 8216,
3374 ["OpenCurlyQuote"] = 8216,
3375 ["rsquo"] = 8217,
3376 ["rsquor"] = 8217,
3377 ["CloseCurlyQuote"] = 8217,
3378 ["lsquor"] = 8218,
3379 ["sbquo"] = 8218,
3380 ["ldquo"] = 8220,
3381 ["OpenCurlyDoubleQuote"] = 8220,
3382 ["rdquo"] = 8221,
3383 ["rdquor"] = 8221,
3384 ["CloseCurlyDoubleQuote"] = 8221,
3385 ["ldquor"] = 8222,
3386 ["bdquo"] = 8222,
3387 ["dagger"] = 8224,
3388 ["Dagger"] = 8225,
3389 ["ddagger"] = 8225,
3390 ["bullet"] = 8226,
3391 ["bullet"] = 8226,
3392 ["nldr"] = 8229,
3393 ["hellip"] = 8230,
3394 ["mldr"] = 8230,
3395 ["permil"] = 8240,
3396 ["pertenk"] = 8241,
3397 ["prime"] = 8242,
3398 ["Prime"] = 8243,

3399 ["tprime"] = 8244,
3400 ["bprime"] = 8245,
3401 ["backprime"] = 8245,
3402 ["lsaquo"] = 8249,
3403 ["rsaquo"] = 8250,
3404 ["oline"] = 8254,
3405 ["caret"] = 8257,
3406 ["hybull"] = 8259,
3407 ["frasl"] = 8260,
3408 ["bsemi"] = 8271,
3409 ["qprime"] = 8279,
3410 ["MediumSpace"] = 8287,
3411 ["NoBreak"] = 8288,
3412 ["ApplyFunction"] = 8289,
3413 ["af"] = 8289,
3414 ["InvisibleTimes"] = 8290,
3415 ["it"] = 8290,
3416 ["InvisibleComma"] = 8291,
3417 ["ic"] = 8291,
3418 ["euro"] = 8364,
3419 ["tdot"] = 8411,
3420 ["TripleDot"] = 8411,
3421 ["DotDot"] = 8412,
3422 ["Copf"] = 8450,
3423 ["complexes"] = 8450,
3424 ["incare"] = 8453,
3425 ["gscr"] = 8458,
3426 ["hamilt"] = 8459,
3427 ["HilbertSpace"] = 8459,
3428 ["Hscr"] = 8459,
3429 ["Hfr"] = 8460,
3430 ["Poincareplane"] = 8460,
3431 ["quaternions"] = 8461,
3432 ["Hopf"] = 8461,
3433 ["planckh"] = 8462,
3434 ["planck"] = 8463,
3435 ["hbar"] = 8463,
3436 ["plankv"] = 8463,
3437 ["hslash"] = 8463,
3438 ["Iscr"] = 8464,
3439 ["imagline"] = 8464,
3440 ["image"] = 8465,
3441 ["Im"] = 8465,
3442 ["imagpart"] = 8465,
3443 ["Ifr"] = 8465,
3444 ["Lscr"] = 8466,
3445 ["lagran"] = 8466,

3446 ["Laplacetrif"] = 8466,
3447 ["ell"] = 8467,
3448 ["Nopf"] = 8469,
3449 ["naturals"] = 8469,
3450 ["numero"] = 8470,
3451 ["copysr"] = 8471,
3452 ["weierp"] = 8472,
3453 ["wp"] = 8472,
3454 ["Popf"] = 8473,
3455 ["primes"] = 8473,
3456 ["rationals"] = 8474,
3457 ["Qopf"] = 8474,
3458 ["Rscr"] = 8475,
3459 ["realine"] = 8475,
3460 ["real"] = 8476,
3461 ["Re"] = 8476,
3462 ["realpart"] = 8476,
3463 ["Rfr"] = 8476,
3464 ["reals"] = 8477,
3465 ["Ropf"] = 8477,
3466 ["rx"] = 8478,
3467 ["trade"] = 8482,
3468 ["TRADE"] = 8482,
3469 ["integers"] = 8484,
3470 ["Zopf"] = 8484,
3471 ["ohm"] = 8486,
3472 ["mho"] = 8487,
3473 ["Zfr"] = 8488,
3474 ["zeetrif"] = 8488,
3475 ["iiota"] = 8489,
3476 ["angst"] = 8491,
3477 ["bernou"] = 8492,
3478 ["Bernoullis"] = 8492,
3479 ["Bscr"] = 8492,
3480 ["Cfr"] = 8493,
3481 ["Cayleys"] = 8493,
3482 ["escr"] = 8495,
3483 ["Escr"] = 8496,
3484 ["expectation"] = 8496,
3485 ["Fscr"] = 8497,
3486 ["Fouriertrif"] = 8497,
3487 ["phmmat"] = 8499,
3488 ["Mellintrif"] = 8499,
3489 ["Mscr"] = 8499,
3490 ["order"] = 8500,
3491 ["orderof"] = 8500,
3492 ["oscr"] = 8500,

3493 ["alefsym"] = 8501,
 3494 ["aleph"] = 8501,
 3495 ["beth"] = 8502,
 3496 ["gimel"] = 8503,
 3497 ["daleth"] = 8504,
 3498 ["CapitalDifferentialD"] = 8517,
 3499 ["DD"] = 8517,
 3500 ["DifferentialD"] = 8518,
 3501 ["dd"] = 8518,
 3502 ["ExponentialE"] = 8519,
 3503 ["exponentiale"] = 8519,
 3504 ["ee"] = 8519,
 3505 ["ImaginaryI"] = 8520,
 3506 ["ii"] = 8520,
 3507 ["frac13"] = 8531,
 3508 ["frac23"] = 8532,
 3509 ["frac15"] = 8533,
 3510 ["frac25"] = 8534,
 3511 ["frac35"] = 8535,
 3512 ["frac45"] = 8536,
 3513 ["frac16"] = 8537,
 3514 ["frac56"] = 8538,
 3515 ["frac18"] = 8539,
 3516 ["frac38"] = 8540,
 3517 ["frac58"] = 8541,
 3518 ["frac78"] = 8542,
 3519 ["larr"] = 8592,
 3520 ["leftarrow"] = 8592,
 3521 ["LeftArrow"] = 8592,
 3522 ["slarr"] = 8592,
 3523 ["ShortLeftArrow"] = 8592,
 3524 ["uarr"] = 8593,
 3525 ["uparrow"] = 8593,
 3526 ["UpArrow"] = 8593,
 3527 ["ShortUpArrow"] = 8593,
 3528 ["rarr"] = 8594,
 3529 ["rightarrow"] = 8594,
 3530 ["RightArrow"] = 8594,
 3531 ["srarr"] = 8594,
 3532 ["ShortRightArrow"] = 8594,
 3533 ["darr"] = 8595,
 3534 ["downarrow"] = 8595,
 3535 ["DownArrow"] = 8595,
 3536 ["ShortDownArrow"] = 8595,
 3537 ["harr"] = 8596,
 3538 ["leftrightarrow"] = 8596,
 3539 ["LeftRightArrow"] = 8596,

```

3540 ["varr"] = 8597,
3541 ["updownarrow"] = 8597,
3542 ["UpDownArrow"] = 8597,
3543 ["nwarr"] = 8598,
3544 ["UpperLeftArrow"] = 8598,
3545 ["nwarrow"] = 8598,
3546 ["nearr"] = 8599,
3547 ["UpperRightArrow"] = 8599,
3548 ["nearrow"] = 8599,
3549 ["searr"] = 8600,
3550 ["searrow"] = 8600,
3551 ["LowerRightArrow"] = 8600,
3552 ["swarr"] = 8601,
3553 ["swarrow"] = 8601,
3554 ["LowerLeftArrow"] = 8601,
3555 ["nlarr"] = 8602,
3556 ["nleftarrow"] = 8602,
3557 ["nrarr"] = 8603,
3558 ["nrightarrow"] = 8603,
3559 ["rarrw"] = 8605,
3560 ["rightsquigarrow"] = 8605,
3561 ["Larr"] = 8606,
3562 ["twoheadleftarrow"] = 8606,
3563 ["Uarr"] = 8607,
3564 ["Rarr"] = 8608,
3565 ["twoheadrightarrow"] = 8608,
3566 ["Darr"] = 8609,
3567 ["larrtl"] = 8610,
3568 ["leftarrowtail"] = 8610,
3569 ["rarrtl"] = 8611,
3570 ["rightarrowtail"] = 8611,
3571 ["LeftTeeArrow"] = 8612,
3572 ["mapstoleft"] = 8612,
3573 ["UpTeeArrow"] = 8613,
3574 ["mapstoup"] = 8613,
3575 ["map"] = 8614,
3576 ["RightTeeArrow"] = 8614,
3577 ["mapsto"] = 8614,
3578 ["DownTeeArrow"] = 8615,
3579 ["mapstodown"] = 8615,
3580 ["larrhk"] = 8617,
3581 ["hookleftarrow"] = 8617,
3582 ["rarrhk"] = 8618,
3583 ["hookrightarrow"] = 8618,
3584 ["larrlp"] = 8619,
3585 ["looparrowleft"] = 8619,
3586 ["rarrlp"] = 8620,

```

3587 ["looparrowright"] = 8620,
 3588 ["harrw"] = 8621,
 3589 ["leftrightsquigarrow"] = 8621,
 3590 ["nharr"] = 8622,
 3591 ["nletrightarrow"] = 8622,
 3592 ["lsh"] = 8624,
 3593 ["Lsh"] = 8624,
 3594 ["rsh"] = 8625,
 3595 ["Rsh"] = 8625,
 3596 ["ldsh"] = 8626,
 3597 ["rdsh"] = 8627,
 3598 ["crarr"] = 8629,
 3599 ["cularr"] = 8630,
 3600 ["curvearrowleft"] = 8630,
 3601 ["curarr"] = 8631,
 3602 ["curvearrowright"] = 8631,
 3603 ["olarr"] = 8634,
 3604 ["circlearrowleft"] = 8634,
 3605 ["orarr"] = 8635,
 3606 ["circlearrowright"] = 8635,
 3607 ["lharu"] = 8636,
 3608 ["LeftVector"] = 8636,
 3609 ["leftharpoonup"] = 8636,
 3610 ["lhard"] = 8637,
 3611 ["leftharpoondown"] = 8637,
 3612 ["DownLeftVector"] = 8637,
 3613 ["uharr"] = 8638,
 3614 ["upharpoonright"] = 8638,
 3615 ["RightUpVector"] = 8638,
 3616 ["uharl"] = 8639,
 3617 ["upharpoonleft"] = 8639,
 3618 ["LeftUpVector"] = 8639,
 3619 ["rharu"] = 8640,
 3620 ["RightVector"] = 8640,
 3621 ["rightharpoonup"] = 8640,
 3622 ["rhard"] = 8641,
 3623 ["rightharpoondown"] = 8641,
 3624 ["DownRightVector"] = 8641,
 3625 ["dharr"] = 8642,
 3626 ["RightDownVector"] = 8642,
 3627 ["downharpoonright"] = 8642,
 3628 ["dharl"] = 8643,
 3629 ["LeftDownVector"] = 8643,
 3630 ["downharpoonleft"] = 8643,
 3631 ["rlarr"] = 8644,
 3632 ["rightleftarrows"] = 8644,
 3633 ["RightArrowLeftArrow"] = 8644,

3634 ["udarr"] = 8645,
 3635 ["UpArrowDownArrow"] = 8645,
 3636 ["lrarr"] = 8646,
 3637 ["leftrightarrows"] = 8646,
 3638 ["LeftArrowRightArrow"] = 8646,
 3639 ["llarr"] = 8647,
 3640 ["leftleftarrows"] = 8647,
 3641 ["uuarr"] = 8648,
 3642 ["upuparrows"] = 8648,
 3643 ["rrarr"] = 8649,
 3644 ["rightrightarrows"] = 8649,
 3645 ["ddarr"] = 8650,
 3646 ["downdownarrows"] = 8650,
 3647 ["lrhar"] = 8651,
 3648 ["ReverseEquilibrium"] = 8651,
 3649 ["leftrightharpoons"] = 8651,
 3650 ["rlhar"] = 8652,
 3651 ["rightleftharpoons"] = 8652,
 3652 ["Equilibrium"] = 8652,
 3653 ["nlArr"] = 8653,
 3654 ["nLeftarrow"] = 8653,
 3655 ["nhArr"] = 8654,
 3656 ["nLeftrightarrow"] = 8654,
 3657 ["nrArr"] = 8655,
 3658 ["nRightarrow"] = 8655,
 3659 ["lArr"] = 8656,
 3660 ["Leftarrow"] = 8656,
 3661 ["DoubleLeftArrow"] = 8656,
 3662 ["uArr"] = 8657,
 3663 ["Uparrow"] = 8657,
 3664 ["DoubleUpArrow"] = 8657,
 3665 ["rArr"] = 8658,
 3666 ["Rightarrow"] = 8658,
 3667 ["Implies"] = 8658,
 3668 ["DoubleRightArrow"] = 8658,
 3669 ["dArr"] = 8659,
 3670 ["Downarrow"] = 8659,
 3671 ["DoubleDownArrow"] = 8659,
 3672 ["hArr"] = 8660,
 3673 ["Leftrightarrow"] = 8660,
 3674 ["DoubleLeftRightArrow"] = 8660,
 3675 ["iff"] = 8660,
 3676 ["vArr"] = 8661,
 3677 ["Updownarrow"] = 8661,
 3678 ["DoubleUpDownArrow"] = 8661,
 3679 ["nwArr"] = 8662,
 3680 ["neArr"] = 8663,

3681 ["seArr"] = 8664,
3682 ["swArr"] = 8665,
3683 ["lAarr"] = 8666,
3684 ["Lleftarrow"] = 8666,
3685 ["rAarr"] = 8667,
3686 ["Rrightarrow"] = 8667,
3687 ["zigrarr"] = 8669,
3688 ["larrb"] = 8676,
3689 ["LeftArrowBar"] = 8676,
3690 ["rarrb"] = 8677,
3691 ["RightArrowBar"] = 8677,
3692 ["duarr"] = 8693,
3693 ["DownArrowUpArrow"] = 8693,
3694 ["loarr"] = 8701,
3695 ["roarr"] = 8702,
3696 ["hoarr"] = 8703,
3697 ["forall"] = 8704,
3698 ["ForAll"] = 8704,
3699 ["comp"] = 8705,
3700 ["complement"] = 8705,
3701 ["part"] = 8706,
3702 ["PartialD"] = 8706,
3703 ["exist"] = 8707,
3704 ["Exists"] = 8707,
3705 ["nexist"] = 8708,
3706 ["NotExists"] = 8708,
3707 ["nexists"] = 8708,
3708 ["empty"] = 8709,
3709 ["emptyset"] = 8709,
3710 ["emptyv"] = 8709,
3711 ["varnothing"] = 8709,
3712 ["nabla"] = 8711,
3713 ["Del"] = 8711,
3714 ["isin"] = 8712,
3715 ["isinv"] = 8712,
3716 ["Element"] = 8712,
3717 ["in"] = 8712,
3718 ["notin"] = 8713,
3719 ["NotElement"] = 8713,
3720 ["notinva"] = 8713,
3721 ["niv"] = 8715,
3722 ["ReverseElement"] = 8715,
3723 ["ni"] = 8715,
3724 ["SuchThat"] = 8715,
3725 ["notni"] = 8716,
3726 ["notniva"] = 8716,
3727 ["NotReverseElement"] = 8716,

3728 ["prod"] = 8719,
3729 ["Product"] = 8719,
3730 ["coprod"] = 8720,
3731 ["Coproduct"] = 8720,
3732 ["sum"] = 8721,
3733 ["Sum"] = 8721,
3734 ["minus"] = 8722,
3735 ["mnplus"] = 8723,
3736 ["mp"] = 8723,
3737 ["MinusPlus"] = 8723,
3738 ["plusdo"] = 8724,
3739 ["dotplus"] = 8724,
3740 ["setmn"] = 8726,
3741 ["setminus"] = 8726,
3742 ["Backslash"] = 8726,
3743 ["ssetmn"] = 8726,
3744 ["smallsetminus"] = 8726,
3745 ["lowast"] = 8727,
3746 ["compfn"] = 8728,
3747 ["SmallCircle"] = 8728,
3748 ["radic"] = 8730,
3749 ["Sqrt"] = 8730,
3750 ["prop"] = 8733,
3751 ["propto"] = 8733,
3752 ["Proportional"] = 8733,
3753 ["vprop"] = 8733,
3754 ["varpropto"] = 8733,
3755 ["infin"] = 8734,
3756 ["angrt"] = 8735,
3757 ["ang"] = 8736,
3758 ["angle"] = 8736,
3759 ["angmsd"] = 8737,
3760 ["measuredangle"] = 8737,
3761 ["angsph"] = 8738,
3762 ["mid"] = 8739,
3763 ["VerticalBar"] = 8739,
3764 ["smid"] = 8739,
3765 ["shortmid"] = 8739,
3766 ["nmid"] = 8740,
3767 ["NotVerticalBar"] = 8740,
3768 ["nsmid"] = 8740,
3769 ["nshortmid"] = 8740,
3770 ["par"] = 8741,
3771 ["parallel"] = 8741,
3772 ["DoubleVerticalBar"] = 8741,
3773 ["spar"] = 8741,
3774 ["shortparallel"] = 8741,

3775 ["npar"] = 8742,
3776 ["nparallel"] = 8742,
3777 ["NotDoubleVerticalBar"] = 8742,
3778 ["nspar"] = 8742,
3779 ["nshortparallel"] = 8742,
3780 ["and"] = 8743,
3781 ["wedge"] = 8743,
3782 ["or"] = 8744,
3783 ["vee"] = 8744,
3784 ["cap"] = 8745,
3785 ["cup"] = 8746,
3786 ["int"] = 8747,
3787 ["Integral"] = 8747,
3788 ["Int"] = 8748,
3789 ["tint"] = 8749,
3790 ["iiint"] = 8749,
3791 ["conint"] = 8750,
3792 ["oint"] = 8750,
3793 ["ContourIntegral"] = 8750,
3794 ["Conint"] = 8751,
3795 ["DoubleContourIntegral"] = 8751,
3796 ["Cconint"] = 8752,
3797 ["cwint"] = 8753,
3798 ["cwconint"] = 8754,
3799 ["ClockwiseContourIntegral"] = 8754,
3800 ["awconint"] = 8755,
3801 ["CounterClockwiseContourIntegral"] = 8755,
3802 ["there4"] = 8756,
3803 ["therefore"] = 8756,
3804 ["Therefore"] = 8756,
3805 ["because"] = 8757,
3806 ["because"] = 8757,
3807 ["Because"] = 8757,
3808 ["ratio"] = 8758,
3809 ["Colon"] = 8759,
3810 ["Proportion"] = 8759,
3811 ["minusd"] = 8760,
3812 ["dotminus"] = 8760,
3813 ["mDDot"] = 8762,
3814 ["homtht"] = 8763,
3815 ["sim"] = 8764,
3816 ["Tilde"] = 8764,
3817 ["thksim"] = 8764,
3818 ["thicksim"] = 8764,
3819 ["bsim"] = 8765,
3820 ["backsim"] = 8765,
3821 ["ac"] = 8766,

3822 ["mstpos"] = 8766,
 3823 ["acd"] = 8767,
 3824 ["wreath"] = 8768,
 3825 ["VerticalTilde"] = 8768,
 3826 ["wr"] = 8768,
 3827 ["nsim"] = 8769,
 3828 ["NotTilde"] = 8769,
 3829 ["esim"] = 8770,
 3830 ["EqualTilde"] = 8770,
 3831 ["eqsim"] = 8770,
 3832 ["sime"] = 8771,
 3833 ["TildeEqual"] = 8771,
 3834 ["simeq"] = 8771,
 3835 ["nsime"] = 8772,
 3836 ["nsimeq"] = 8772,
 3837 ["NotTildeEqual"] = 8772,
 3838 ["cong"] = 8773,
 3839 ["TildeFullEqual"] = 8773,
 3840 ["simne"] = 8774,
 3841 ["ncong"] = 8775,
 3842 ["NotTildeFullEqual"] = 8775,
 3843 ["asymp"] = 8776,
 3844 ["ap"] = 8776,
 3845 ["TildeTilde"] = 8776,
 3846 ["approx"] = 8776,
 3847 ["thkap"] = 8776,
 3848 ["thickapprox"] = 8776,
 3849 ["nap"] = 8777,
 3850 ["NotTildeTilde"] = 8777,
 3851 ["napprox"] = 8777,
 3852 ["ape"] = 8778,
 3853 ["approxpeq"] = 8778,
 3854 ["apid"] = 8779,
 3855 ["bcong"] = 8780,
 3856 ["backcong"] = 8780,
 3857 ["asympeq"] = 8781,
 3858 ["CupCap"] = 8781,
 3859 ["bump"] = 8782,
 3860 ["HumpDownHump"] = 8782,
 3861 ["Bumpeq"] = 8782,
 3862 ["bumpe"] = 8783,
 3863 ["HumpEqual"] = 8783,
 3864 ["bumpeq"] = 8783,
 3865 ["esdot"] = 8784,
 3866 ["DotEqual"] = 8784,
 3867 ["doteq"] = 8784,
 3868 ["eDot"] = 8785,

3869 ["doteqdot"] = 8785,
 3870 ["efDot"] = 8786,
 3871 ["fallingdotseq"] = 8786,
 3872 ["erDot"] = 8787,
 3873 ["risingdotseq"] = 8787,
 3874 ["colone"] = 8788,
 3875 ["coloneq"] = 8788,
 3876 ["Assign"] = 8788,
 3877 ["ecolon"] = 8789,
 3878 ["eqcolon"] = 8789,
 3879 ["ecir"] = 8790,
 3880 ["eqcirc"] = 8790,
 3881 ["cire"] = 8791,
 3882 ["circeq"] = 8791,
 3883 ["wedgeq"] = 8793,
 3884 ["veeeq"] = 8794,
 3885 ["trie"] = 8796,
 3886 ["triangleq"] = 8796,
 3887 ["equest"] = 8799,
 3888 ["questeq"] = 8799,
 3889 ["ne"] = 8800,
 3890 ["NotEqual"] = 8800,
 3891 ["equiv"] = 8801,
 3892 ["Congruent"] = 8801,
 3893 ["nequiv"] = 8802,
 3894 ["NotCongruent"] = 8802,
 3895 ["le"] = 8804,
 3896 ["leq"] = 8804,
 3897 ["ge"] = 8805,
 3898 ["GreaterEqual"] = 8805,
 3899 ["geq"] = 8805,
 3900 ["lE"] = 8806,
 3901 ["LessFullEqual"] = 8806,
 3902 ["leqq"] = 8806,
 3903 ["gE"] = 8807,
 3904 ["GreaterFullEqual"] = 8807,
 3905 ["geqq"] = 8807,
 3906 ["lnE"] = 8808,
 3907 ["lneqq"] = 8808,
 3908 ["gnE"] = 8809,
 3909 ["gneqq"] = 8809,
 3910 ["Lt"] = 8810,
 3911 ["NestedLessLess"] = 8810,
 3912 ["ll"] = 8810,
 3913 ["Gt"] = 8811,
 3914 ["NestedGreaterGreater"] = 8811,
 3915 ["gg"] = 8811,

3916 ["twixt"] = 8812,
 3917 ["between"] = 8812,
 3918 ["NotCupCap"] = 8813,
 3919 ["nlt"] = 8814,
 3920 ["NotLess"] = 8814,
 3921 ["nless"] = 8814,
 3922 ["ngt"] = 8815,
 3923 ["NotGreater"] = 8815,
 3924 ["ngtr"] = 8815,
 3925 ["nle"] = 8816,
 3926 ["NotLessEqual"] = 8816,
 3927 ["nleq"] = 8816,
 3928 ["nge"] = 8817,
 3929 ["NotGreaterEqual"] = 8817,
 3930 ["ngeq"] = 8817,
 3931 ["lsim"] = 8818,
 3932 ["LessTilde"] = 8818,
 3933 ["lesssim"] = 8818,
 3934 ["gsim"] = 8819,
 3935 ["gtrsim"] = 8819,
 3936 ["GreaterTilde"] = 8819,
 3937 ["nlsim"] = 8820,
 3938 ["NotLessTilde"] = 8820,
 3939 ["ngsim"] = 8821,
 3940 ["NotGreaterTilde"] = 8821,
 3941 ["lg"] = 8822,
 3942 ["lessgtr"] = 8822,
 3943 ["LessGreater"] = 8822,
 3944 ["gl"] = 8823,
 3945 ["gtrless"] = 8823,
 3946 ["GreaterLess"] = 8823,
 3947 ["ntlg"] = 8824,
 3948 ["NotLessGreater"] = 8824,
 3949 ["ntgl"] = 8825,
 3950 ["NotGreaterLess"] = 8825,
 3951 ["pr"] = 8826,
 3952 ["Precedes"] = 8826,
 3953 ["prec"] = 8826,
 3954 ["sc"] = 8827,
 3955 ["Succeeds"] = 8827,
 3956 ["succ"] = 8827,
 3957 ["prcue"] = 8828,
 3958 ["PrecedesSlantEqual"] = 8828,
 3959 ["preccurlyeq"] = 8828,
 3960 ["sccue"] = 8829,
 3961 ["SucceedsSlantEqual"] = 8829,
 3962 ["succcurlyeq"] = 8829,

3963 ["prsim"] = 8830,
3964 ["precsim"] = 8830,
3965 ["PrecedesTilde"] = 8830,
3966 ["scsim"] = 8831,
3967 ["succsim"] = 8831,
3968 ["SucceedsTilde"] = 8831,
3969 ["npr"] = 8832,
3970 ["nprec"] = 8832,
3971 ["NotPrecedes"] = 8832,
3972 ["nsc"] = 8833,
3973 ["nsucc"] = 8833,
3974 ["NotSucceeds"] = 8833,
3975 ["sub"] = 8834,
3976 ["subset"] = 8834,
3977 ["sup"] = 8835,
3978 ["supset"] = 8835,
3979 ["Superset"] = 8835,
3980 ["nsub"] = 8836,
3981 ["nsup"] = 8837,
3982 ["sube"] = 8838,
3983 ["SubsetEqual"] = 8838,
3984 ["subseteq"] = 8838,
3985 ["supe"] = 8839,
3986 ["supseteq"] = 8839,
3987 ["SupersetEqual"] = 8839,
3988 ["nsube"] = 8840,
3989 ["nsubseteq"] = 8840,
3990 ["NotSubsetEqual"] = 8840,
3991 ["nsupe"] = 8841,
3992 ["nsupseteq"] = 8841,
3993 ["NotSupersetEqual"] = 8841,
3994 ["subne"] = 8842,
3995 ["subsetneq"] = 8842,
3996 ["supne"] = 8843,
3997 ["supsetneq"] = 8843,
3998 ["cupdot"] = 8845,
3999 ["uplus"] = 8846,
4000 ["UnionPlus"] = 8846,
4001 ["sqsub"] = 8847,
4002 ["SquareSubset"] = 8847,
4003 ["sqsubset"] = 8847,
4004 ["sqsup"] = 8848,
4005 ["SquareSuperset"] = 8848,
4006 ["sqsupset"] = 8848,
4007 ["sqsube"] = 8849,
4008 ["SquareSubsetEqual"] = 8849,
4009 ["sqsubseteq"] = 8849,

```

4010 ["sqsupe"] = 8850,
4011 ["SquareSupersetEqual"] = 8850,
4012 ["sqsupseteq"] = 8850,
4013 ["sqcap"] = 8851,
4014 ["SquareIntersection"] = 8851,
4015 ["sqcup"] = 8852,
4016 ["SquareUnion"] = 8852,
4017 ["oplus"] = 8853,
4018 ["CirclePlus"] = 8853,
4019 ["ominus"] = 8854,
4020 ["CircleMinus"] = 8854,
4021 ["otimes"] = 8855,
4022 ["CircleTimes"] = 8855,
4023 ["osol"] = 8856,
4024 ["odot"] = 8857,
4025 ["CircleDot"] = 8857,
4026 ["ocir"] = 8858,
4027 ["circledcirc"] = 8858,
4028 ["oast"] = 8859,
4029 ["circledast"] = 8859,
4030 ["odash"] = 8861,
4031 ["circleddash"] = 8861,
4032 ["plusb"] = 8862,
4033 ["boxplus"] = 8862,
4034 ["minusb"] = 8863,
4035 ["boxminus"] = 8863,
4036 ["timesb"] = 8864,
4037 ["boxtimes"] = 8864,
4038 ["sdotb"] = 8865,
4039 ["dotsquare"] = 8865,
4040 ["vdash"] = 8866,
4041 ["RightTee"] = 8866,
4042 ["dashv"] = 8867,
4043 ["LeftTee"] = 8867,
4044 ["top"] = 8868,
4045 ["DownTee"] = 8868,
4046 ["bottom"] = 8869,
4047 ["bot"] = 8869,
4048 ["perp"] = 8869,
4049 ["UpTee"] = 8869,
4050 ["models"] = 8871,
4051 ["vDash"] = 8872,
4052 ["DoubleRightTee"] = 8872,
4053 ["Vdash"] = 8873,
4054 ["Vvdash"] = 8874,
4055 ["VDash"] = 8875,
4056 ["nvdash"] = 8876,

```

4057 ["nvDash"] = 8877,
 4058 ["nVdash"] = 8878,
 4059 ["nVDash"] = 8879,
 4060 ["prurel"] = 8880,
 4061 ["vltri"] = 8882,
 4062 ["vartriangleleft"] = 8882,
 4063 ["LeftTriangle"] = 8882,
 4064 ["vrtri"] = 8883,
 4065 ["vartriangleright"] = 8883,
 4066 ["RightTriangle"] = 8883,
 4067 ["ltrie"] = 8884,
 4068 ["trianglelefteq"] = 8884,
 4069 ["LeftTriangleEqual"] = 8884,
 4070 ["rtrie"] = 8885,
 4071 ["trianglerighteq"] = 8885,
 4072 ["RightTriangleEqual"] = 8885,
 4073 ["origof"] = 8886,
 4074 ["imof"] = 8887,
 4075 ["mumap"] = 8888,
 4076 ["multimap"] = 8888,
 4077 ["hercon"] = 8889,
 4078 ["intcal"] = 8890,
 4079 ["intercal"] = 8890,
 4080 ["veebar"] = 8891,
 4081 ["barvee"] = 8893,
 4082 ["angrtvb"] = 8894,
 4083 ["ltri"] = 8895,
 4084 ["xwedge"] = 8896,
 4085 ["Wedge"] = 8896,
 4086 ["bigwedge"] = 8896,
 4087 ["xvee"] = 8897,
 4088 ["Vee"] = 8897,
 4089 ["bigvee"] = 8897,
 4090 ["xcap"] = 8898,
 4091 ["Intersection"] = 8898,
 4092 ["bigcap"] = 8898,
 4093 ["xcup"] = 8899,
 4094 ["Union"] = 8899,
 4095 ["bigcup"] = 8899,
 4096 ["diam"] = 8900,
 4097 ["diamond"] = 8900,
 4098 ["Diamond"] = 8900,
 4099 ["sdot"] = 8901,
 4100 ["sstarf"] = 8902,
 4101 ["Star"] = 8902,
 4102 ["divonx"] = 8903,
 4103 ["divideontimes"] = 8903,

4104 ["bowtie"] = 8904,
 4105 ["ltimes"] = 8905,
 4106 ["rtimes"] = 8906,
 4107 ["lthree"] = 8907,
 4108 ["leftthreetimes"] = 8907,
 4109 ["rthree"] = 8908,
 4110 ["rightthreetimes"] = 8908,
 4111 ["bsime"] = 8909,
 4112 ["backsimeq"] = 8909,
 4113 ["cuvee"] = 8910,
 4114 ["curlyvee"] = 8910,
 4115 ["cuwed"] = 8911,
 4116 ["curlywedge"] = 8911,
 4117 ["Sub"] = 8912,
 4118 ["Subset"] = 8912,
 4119 ["Sup"] = 8913,
 4120 ["Supset"] = 8913,
 4121 ["Cap"] = 8914,
 4122 ["Cup"] = 8915,
 4123 ["fork"] = 8916,
 4124 ["pitchfork"] = 8916,
 4125 ["epar"] = 8917,
 4126 ["ltdot"] = 8918,
 4127 ["lessdot"] = 8918,
 4128 ["gtdot"] = 8919,
 4129 ["gtrdot"] = 8919,
 4130 ["Ll"] = 8920,
 4131 ["Gg"] = 8921,
 4132 ["ggg"] = 8921,
 4133 ["leg"] = 8922,
 4134 ["LessEqualGreater"] = 8922,
 4135 ["lesseqgtr"] = 8922,
 4136 ["gel"] = 8923,
 4137 ["gtreqless"] = 8923,
 4138 ["GreaterEqualLess"] = 8923,
 4139 ["cuepr"] = 8926,
 4140 ["curlyeqprec"] = 8926,
 4141 ["cuesc"] = 8927,
 4142 ["curlyeqsucc"] = 8927,
 4143 ["nprcue"] = 8928,
 4144 ["NotPrecedesSlantEqual"] = 8928,
 4145 ["nsccue"] = 8929,
 4146 ["NotSucceedsSlantEqual"] = 8929,
 4147 ["nsqsube"] = 8930,
 4148 ["NotSquareSubsetEqual"] = 8930,
 4149 ["nsqsupe"] = 8931,
 4150 ["NotSquareSupersetEqual"] = 8931,

4151 ["lnsim"] = 8934,
 4152 ["gnsim"] = 8935,
 4153 ["prnsim"] = 8936,
 4154 ["precnsim"] = 8936,
 4155 ["scnsim"] = 8937,
 4156 ["succnsim"] = 8937,
 4157 ["nltri"] = 8938,
 4158 ["ntriangleleft"] = 8938,
 4159 ["NotLeftTriangle"] = 8938,
 4160 ["nrtri"] = 8939,
 4161 ["ntriangleright"] = 8939,
 4162 ["NotRightTriangle"] = 8939,
 4163 ["nltrie"] = 8940,
 4164 ["ntrianglelefteq"] = 8940,
 4165 ["NotLeftTriangleEqual"] = 8940,
 4166 ["nrtrie"] = 8941,
 4167 ["ntrianglerighteq"] = 8941,
 4168 ["NotRightTriangleEqual"] = 8941,
 4169 ["vellip"] = 8942,
 4170 ["ctdot"] = 8943,
 4171 ["utdot"] = 8944,
 4172 ["dtdot"] = 8945,
 4173 ["disin"] = 8946,
 4174 ["isinsv"] = 8947,
 4175 ["isins"] = 8948,
 4176 ["isindot"] = 8949,
 4177 ["notinvc"] = 8950,
 4178 ["notinvb"] = 8951,
 4179 ["isinE"] = 8953,
 4180 ["nisd"] = 8954,
 4181 ["xnis"] = 8955,
 4182 ["nis"] = 8956,
 4183 ["notnivc"] = 8957,
 4184 ["notnivb"] = 8958,
 4185 ["barwed"] = 8965,
 4186 ["barwedge"] = 8965,
 4187 ["Barwed"] = 8966,
 4188 ["doublebarwedge"] = 8966,
 4189 ["lceil"] = 8968,
 4190 ["LeftCeiling"] = 8968,
 4191 ["rceil"] = 8969,
 4192 ["RightCeiling"] = 8969,
 4193 ["lfloor"] = 8970,
 4194 ["LeftFloor"] = 8970,
 4195 ["rfloor"] = 8971,
 4196 ["RightFloor"] = 8971,
 4197 ["drcrop"] = 8972,

4198 ["dlcrop"] = 8973,
4199 ["urcrop"] = 8974,
4200 ["ulcrop"] = 8975,
4201 ["bnot"] = 8976,
4202 ["proflin"] = 8978,
4203 ["profsurf"] = 8979,
4204 ["telrec"] = 8981,
4205 ["target"] = 8982,
4206 ["ulcorn"] = 8988,
4207 ["ulcorner"] = 8988,
4208 ["urcorn"] = 8989,
4209 ["urcorner"] = 8989,
4210 ["dlcorn"] = 8990,
4211 ["llcorner"] = 8990,
4212 ["drcorn"] = 8991,
4213 ["lrcorn"] = 8991,
4214 ["frown"] = 8994,
4215 ["sfrown"] = 8994,
4216 ["smile"] = 8995,
4217 ["ssmile"] = 8995,
4218 ["cylcty"] = 9005,
4219 ["profalar"] = 9006,
4220 ["topbot"] = 9014,
4221 ["ovbar"] = 9021,
4222 ["solbar"] = 9023,
4223 ["angzarr"] = 9084,
4224 ["lmoust"] = 9136,
4225 ["lmoustache"] = 9136,
4226 ["rmoust"] = 9137,
4227 ["rmoustache"] = 9137,
4228 ["tbrk"] = 9140,
4229 ["OverBracket"] = 9140,
4230 ["bbrk"] = 9141,
4231 ["UnderBracket"] = 9141,
4232 ["bbrktbrk"] = 9142,
4233 ["OverParenthesis"] = 9180,
4234 ["UnderParenthesis"] = 9181,
4235 ["OverBrace"] = 9182,
4236 ["UnderBrace"] = 9183,
4237 ["trpezium"] = 9186,
4238 ["elinters"] = 9191,
4239 ["blank"] = 9251,
4240 ["oS"] = 9416,
4241 ["circledS"] = 9416,
4242 ["boxh"] = 9472,
4243 ["HorizontalLine"] = 9472,
4244 ["boxv"] = 9474,

4245 ["boxdr"] = 9484,
4246 ["boxdl"] = 9488,
4247 ["boxur"] = 9492,
4248 ["boxul"] = 9496,
4249 ["boxvr"] = 9500,
4250 ["boxvl"] = 9508,
4251 ["boxhd"] = 9516,
4252 ["boxhu"] = 9524,
4253 ["boxvh"] = 9532,
4254 ["boxH"] = 9552,
4255 ["boxV"] = 9553,
4256 ["boxdR"] = 9554,
4257 ["boxDr"] = 9555,
4258 ["boxDR"] = 9556,
4259 ["boxdL"] = 9557,
4260 ["boxDl"] = 9558,
4261 ["boxDL"] = 9559,
4262 ["boxuR"] = 9560,
4263 ["boxUr"] = 9561,
4264 ["boxUR"] = 9562,
4265 ["boxuL"] = 9563,
4266 ["boxUl"] = 9564,
4267 ["boxUL"] = 9565,
4268 ["boxvR"] = 9566,
4269 ["boxVr"] = 9567,
4270 ["boxVR"] = 9568,
4271 ["boxvL"] = 9569,
4272 ["boxVl"] = 9570,
4273 ["boxVL"] = 9571,
4274 ["boxHd"] = 9572,
4275 ["boxhD"] = 9573,
4276 ["boxHD"] = 9574,
4277 ["boxHu"] = 9575,
4278 ["boxhU"] = 9576,
4279 ["boxHU"] = 9577,
4280 ["boxvH"] = 9578,
4281 ["boxVh"] = 9579,
4282 ["boxVH"] = 9580,
4283 ["uhblk"] = 9600,
4284 ["lhblk"] = 9604,
4285 ["block"] = 9608,
4286 ["blk14"] = 9617,
4287 ["blk12"] = 9618,
4288 ["blk34"] = 9619,
4289 ["squ"] = 9633,
4290 ["square"] = 9633,
4291 ["Square"] = 9633,

4292 ["squf"] = 9642,
4293 ["squarf"] = 9642,
4294 ["blacksquare"] = 9642,
4295 ["FilledVerySmallSquare"] = 9642,
4296 ["EmptyVerySmallSquare"] = 9643,
4297 ["rect"] = 9645,
4298 ["marker"] = 9646,
4299 ["fltns"] = 9649,
4300 ["xutri"] = 9651,
4301 ["bigtriangleup"] = 9651,
4302 ["utri"] = 9652,
4303 ["blacktriangle"] = 9652,
4304 ["utri"] = 9653,
4305 ["triangle"] = 9653,
4306 ["rtri"] = 9656,
4307 ["blacktriangleright"] = 9656,
4308 ["rtri"] = 9657,
4309 ["triangleright"] = 9657,
4310 ["xdtri"] = 9661,
4311 ["bigtriangledown"] = 9661,
4312 ["dtri"] = 9662,
4313 ["blacktriangledown"] = 9662,
4314 ["dtri"] = 9663,
4315 ["triangledown"] = 9663,
4316 ["ltrif"] = 9666,
4317 ["blacktriangleleft"] = 9666,
4318 ["ltrif"] = 9667,
4319 ["triangleleft"] = 9667,
4320 ["loz"] = 9674,
4321 ["lozenge"] = 9674,
4322 ["cir"] = 9675,
4323 ["tridot"] = 9708,
4324 ["xcirc"] = 9711,
4325 ["bigcirc"] = 9711,
4326 ["ultri"] = 9720,
4327 ["urtri"] = 9721,
4328 ["lltri"] = 9722,
4329 ["EmptySmallSquare"] = 9723,
4330 ["FilledSmallSquare"] = 9724,
4331 ["starf"] = 9733,
4332 ["bigstar"] = 9733,
4333 ["star"] = 9734,
4334 ["phone"] = 9742,
4335 ["female"] = 9792,
4336 ["male"] = 9794,
4337 ["spades"] = 9824,
4338 ["spadesuit"] = 9824,

4339 ["clubs"] = 9827,
 4340 ["clubsuit"] = 9827,
 4341 ["hearts"] = 9829,
 4342 ["heartsuit"] = 9829,
 4343 ["diams"] = 9830,
 4344 ["diamondsuit"] = 9830,
 4345 ["sung"] = 9834,
 4346 ["flat"] = 9837,
 4347 ["natur"] = 9838,
 4348 ["natural"] = 9838,
 4349 ["sharp"] = 9839,
 4350 ["check"] = 10003,
 4351 ["checkmark"] = 10003,
 4352 ["cross"] = 10007,
 4353 ["malt"] = 10016,
 4354 ["maltese"] = 10016,
 4355 ["sext"] = 10038,
 4356 ["VerticalSeparator"] = 10072,
 4357 ["lbrk"] = 10098,
 4358 ["rbrk"] = 10099,
 4359 ["lobrk"] = 10214,
 4360 ["LeftDoubleBracket"] = 10214,
 4361 ["robrk"] = 10215,
 4362 ["RightDoubleBracket"] = 10215,
 4363 ["lang"] = 10216,
 4364 ["LeftAngleBracket"] = 10216,
 4365 ["langle"] = 10216,
 4366 ["rang"] = 10217,
 4367 ["RightAngleBracket"] = 10217,
 4368 ["rangle"] = 10217,
 4369 ["Lang"] = 10218,
 4370 ["Rang"] = 10219,
 4371 ["loang"] = 10220,
 4372 ["roang"] = 10221,
 4373 ["xlarr"] = 10229,
 4374 ["longleftarrow"] = 10229,
 4375 ["LongLeftArrow"] = 10229,
 4376 ["xrarr"] = 10230,
 4377 ["longrightarrow"] = 10230,
 4378 ["LongRightArrow"] = 10230,
 4379 ["xharr"] = 10231,
 4380 ["longleftrightarrow"] = 10231,
 4381 ["LongLeftRightArrow"] = 10231,
 4382 ["xlArr"] = 10232,
 4383 ["Longleftarrow"] = 10232,
 4384 ["DoubleLongLeftArrow"] = 10232,
 4385 ["xrArr"] = 10233,

4386 ["Longrightarrow"] = 10233,
 4387 ["DoubleLongRightArrow"] = 10233,
 4388 ["xhArr"] = 10234,
 4389 ["Longleftrightarrow"] = 10234,
 4390 ["DoubleLongLeftRightArrow"] = 10234,
 4391 ["xmap"] = 10236,
 4392 ["longmapsto"] = 10236,
 4393 ["dzigrarr"] = 10239,
 4394 ["nvlArr"] = 10498,
 4395 ["nvrArr"] = 10499,
 4396 ["nvHarr"] = 10500,
 4397 ["Map"] = 10501,
 4398 ["lbarr"] = 10508,
 4399 ["rbarr"] = 10509,
 4400 ["bkarow"] = 10509,
 4401 ["lBarr"] = 10510,
 4402 ["rBarr"] = 10511,
 4403 ["dbkarow"] = 10511,
 4404 ["RBarr"] = 10512,
 4405 ["drbkarow"] = 10512,
 4406 ["DDottrahd"] = 10513,
 4407 ["UpArrowBar"] = 10514,
 4408 ["DownArrowBar"] = 10515,
 4409 ["Rarrtl"] = 10518,
 4410 ["latail"] = 10521,
 4411 ["ratail"] = 10522,
 4412 ["lAtail"] = 10523,
 4413 ["rAtail"] = 10524,
 4414 ["larrfs"] = 10525,
 4415 ["rarrfs"] = 10526,
 4416 ["larrbfs"] = 10527,
 4417 ["rarrbfs"] = 10528,
 4418 ["nwarhk"] = 10531,
 4419 ["nearhk"] = 10532,
 4420 ["searhk"] = 10533,
 4421 ["hksearow"] = 10533,
 4422 ["swarhk"] = 10534,
 4423 ["hkswarow"] = 10534,
 4424 ["nwnear"] = 10535,
 4425 ["nesear"] = 10536,
 4426 ["toea"] = 10536,
 4427 ["seswar"] = 10537,
 4428 ["tosa"] = 10537,
 4429 ["swnwar"] = 10538,
 4430 ["rarrc"] = 10547,
 4431 ["cudarr"] = 10549,
 4432 ["ldca"] = 10550,

4433 ["rdca"] = 10551,
4434 ["cudarrl"] = 10552,
4435 ["larrpl"] = 10553,
4436 ["curarrm"] = 10556,
4437 ["cularrp"] = 10557,
4438 ["rarrpl"] = 10565,
4439 ["harrcir"] = 10568,
4440 ["Uarrocir"] = 10569,
4441 ["lurdshar"] = 10570,
4442 ["ldrushar"] = 10571,
4443 ["LeftRightVector"] = 10574,
4444 ["RightUpDownVector"] = 10575,
4445 ["DownLeftRightVector"] = 10576,
4446 ["LeftUpDownVector"] = 10577,
4447 ["LeftVectorBar"] = 10578,
4448 ["RightVectorBar"] = 10579,
4449 ["RightUpVectorBar"] = 10580,
4450 ["RightDownVectorBar"] = 10581,
4451 ["DownLeftVectorBar"] = 10582,
4452 ["DownRightVectorBar"] = 10583,
4453 ["LeftUpVectorBar"] = 10584,
4454 ["LeftDownVectorBar"] = 10585,
4455 ["LeftTeeVector"] = 10586,
4456 ["RightTeeVector"] = 10587,
4457 ["RightUpTeeVector"] = 10588,
4458 ["RightDownTeeVector"] = 10589,
4459 ["DownLeftTeeVector"] = 10590,
4460 ["DownRightTeeVector"] = 10591,
4461 ["LeftUpTeeVector"] = 10592,
4462 ["LeftDownTeeVector"] = 10593,
4463 ["lHar"] = 10594,
4464 ["uHar"] = 10595,
4465 ["rHar"] = 10596,
4466 ["dHar"] = 10597,
4467 ["luruhar"] = 10598,
4468 ["ldrdhar"] = 10599,
4469 ["ruluhar"] = 10600,
4470 ["rdldhar"] = 10601,
4471 ["lharul"] = 10602,
4472 ["llhard"] = 10603,
4473 ["rharul"] = 10604,
4474 ["lrhard"] = 10605,
4475 ["udhar"] = 10606,
4476 ["UpEquilibrium"] = 10606,
4477 ["duhar"] = 10607,
4478 ["ReverseUpEquilibrium"] = 10607,
4479 ["RoundImplies"] = 10608,

4480 ["erarr"] = 10609,
4481 ["simrarr"] = 10610,
4482 ["larrsim"] = 10611,
4483 ["rarrsim"] = 10612,
4484 ["rarrap"] = 10613,
4485 ["ltlarr"] = 10614,
4486 ["gtrarr"] = 10616,
4487 ["subrarr"] = 10617,
4488 ["suplarr"] = 10619,
4489 ["lfisht"] = 10620,
4490 ["rfisht"] = 10621,
4491 ["ufisht"] = 10622,
4492 ["dfisht"] = 10623,
4493 ["lopar"] = 10629,
4494 ["ropar"] = 10630,
4495 ["lbrke"] = 10635,
4496 ["rbrke"] = 10636,
4497 ["lbrkslu"] = 10637,
4498 ["rbrksld"] = 10638,
4499 ["lbrksld"] = 10639,
4500 ["rbrkslu"] = 10640,
4501 ["langd"] = 10641,
4502 ["rangd"] = 10642,
4503 ["lparlt"] = 10643,
4504 ["rpargt"] = 10644,
4505 ["gtlPar"] = 10645,
4506 ["ltrPar"] = 10646,
4507 ["vzigzag"] = 10650,
4508 ["vangrt"] = 10652,
4509 ["angrtvbd"] = 10653,
4510 ["ange"] = 10660,
4511 ["range"] = 10661,
4512 ["dwangle"] = 10662,
4513 ["uwangle"] = 10663,
4514 ["angmsdaa"] = 10664,
4515 ["angmsdab"] = 10665,
4516 ["angmsdac"] = 10666,
4517 ["angmsdad"] = 10667,
4518 ["angmsdae"] = 10668,
4519 ["angmsdaf"] = 10669,
4520 ["angmsdag"] = 10670,
4521 ["angmsdah"] = 10671,
4522 ["bemptyv"] = 10672,
4523 ["demptyv"] = 10673,
4524 ["cemptyv"] = 10674,
4525 ["raemptyv"] = 10675,
4526 ["laemptyv"] = 10676,

4527 ["ohbar"] = 10677,
 4528 ["omid"] = 10678,
 4529 ["opar"] = 10679,
 4530 ["operp"] = 10681,
 4531 ["olcross"] = 10683,
 4532 ["odsold"] = 10684,
 4533 ["olcir"] = 10686,
 4534 ["ofcir"] = 10687,
 4535 ["olt"] = 10688,
 4536 ["ogt"] = 10689,
 4537 ["cirscir"] = 10690,
 4538 ["cirE"] = 10691,
 4539 ["solb"] = 10692,
 4540 ["bsolb"] = 10693,
 4541 ["boxbox"] = 10697,
 4542 ["trisb"] = 10701,
 4543 ["rtriltri"] = 10702,
 4544 ["LeftTriangleBar"] = 10703,
 4545 ["RightTriangleBar"] = 10704,
 4546 ["race"] = 10714,
 4547 ["iinfin"] = 10716,
 4548 ["infintie"] = 10717,
 4549 ["nvinfin"] = 10718,
 4550 ["eparsl"] = 10723,
 4551 ["smeparsl"] = 10724,
 4552 ["eqvparsl"] = 10725,
 4553 ["lozf"] = 10731,
 4554 ["blacklozenge"] = 10731,
 4555 ["RuleDelayed"] = 10740,
 4556 ["dsol"] = 10742,
 4557 ["xodot"] = 10752,
 4558 ["bigodot"] = 10752,
 4559 ["xoplus"] = 10753,
 4560 ["bigoplus"] = 10753,
 4561 ["xotime"] = 10754,
 4562 ["bigotimes"] = 10754,
 4563 ["xuplus"] = 10756,
 4564 ["biguplus"] = 10756,
 4565 ["xsqcup"] = 10758,
 4566 ["bigsqcup"] = 10758,
 4567 ["qint"] = 10764,
 4568 ["iiiint"] = 10764,
 4569 ["fpartint"] = 10765,
 4570 ["cirfnint"] = 10768,
 4571 ["awint"] = 10769,
 4572 ["rppolint"] = 10770,
 4573 ["scpolint"] = 10771,

4574 ["npolint"] = 10772,
4575 ["pointint"] = 10773,
4576 ["quatint"] = 10774,
4577 ["intlarhk"] = 10775,
4578 ["pluscir"] = 10786,
4579 ["plusacir"] = 10787,
4580 ["simplus"] = 10788,
4581 ["plusdu"] = 10789,
4582 ["plussim"] = 10790,
4583 ["plustwo"] = 10791,
4584 ["mcomma"] = 10793,
4585 ["minusdu"] = 10794,
4586 ["loplus"] = 10797,
4587 ["roplus"] = 10798,
4588 ["Cross"] = 10799,
4589 ["timesd"] = 10800,
4590 ["timesbar"] = 10801,
4591 ["smashp"] = 10803,
4592 ["lotimes"] = 10804,
4593 ["rotimes"] = 10805,
4594 ["otimesas"] = 10806,
4595 ["Otimes"] = 10807,
4596 ["odiv"] = 10808,
4597 ["triplus"] = 10809,
4598 ["triminus"] = 10810,
4599 ["tritime"] = 10811,
4600 ["iproduct"] = 10812,
4601 ["intprod"] = 10812,
4602 ["amalg"] = 10815,
4603 ["capdot"] = 10816,
4604 ["ncup"] = 10818,
4605 ["ncap"] = 10819,
4606 ["capand"] = 10820,
4607 ["cupor"] = 10821,
4608 ["cupcap"] = 10822,
4609 ["capcup"] = 10823,
4610 ["cupbrcap"] = 10824,
4611 ["capbrcup"] = 10825,
4612 ["cupcup"] = 10826,
4613 ["capcap"] = 10827,
4614 ["ccups"] = 10828,
4615 ["ccaps"] = 10829,
4616 ["ccupssm"] = 10832,
4617 ["And"] = 10835,
4618 ["Or"] = 10836,
4619 ["andand"] = 10837,
4620 ["oror"] = 10838,

4621 ["orslope"] = 10839,
 4622 ["andslope"] = 10840,
 4623 ["andv"] = 10842,
 4624 ["orv"] = 10843,
 4625 ["andd"] = 10844,
 4626 ["ord"] = 10845,
 4627 ["wedbar"] = 10847,
 4628 ["sdote"] = 10854,
 4629 ["simdot"] = 10858,
 4630 ["congdote"] = 10861,
 4631 ["easter"] = 10862,
 4632 ["apacir"] = 10863,
 4633 ["apE"] = 10864,
 4634 ["eplus"] = 10865,
 4635 ["pluse"] = 10866,
 4636 ["Esim"] = 10867,
 4637 ["Colone"] = 10868,
 4638 ["Equal"] = 10869,
 4639 ["eDDot"] = 10871,
 4640 ["ddotseq"] = 10871,
 4641 ["equivDD"] = 10872,
 4642 ["ltcir"] = 10873,
 4643 ["gtcir"] = 10874,
 4644 ["ltquest"] = 10875,
 4645 ["gtquest"] = 10876,
 4646 ["les"] = 10877,
 4647 ["LessSlantEqual"] = 10877,
 4648 ["leqslant"] = 10877,
 4649 ["ges"] = 10878,
 4650 ["GreaterSlantEqual"] = 10878,
 4651 ["geqslant"] = 10878,
 4652 ["lesdot"] = 10879,
 4653 ["gesdot"] = 10880,
 4654 ["lesdoto"] = 10881,
 4655 ["gesdoto"] = 10882,
 4656 ["lesdotor"] = 10883,
 4657 ["gesdoto1"] = 10884,
 4658 ["lap"] = 10885,
 4659 ["lessapprox"] = 10885,
 4660 ["gap"] = 10886,
 4661 ["gtrapprox"] = 10886,
 4662 ["lne"] = 10887,
 4663 ["lneq"] = 10887,
 4664 ["gne"] = 10888,
 4665 ["gneq"] = 10888,
 4666 ["lnap"] = 10889,
 4667 ["lnapprox"] = 10889,

4668 ["gnap"] = 10890,
4669 ["gnapprox"] = 10890,
4670 ["lEg"] = 10891,
4671 ["lesseqgtr"] = 10891,
4672 ["gEl"] = 10892,
4673 ["gtreqqless"] = 10892,
4674 ["lsime"] = 10893,
4675 ["gsime"] = 10894,
4676 ["lsimg"] = 10895,
4677 ["gsiml"] = 10896,
4678 ["lgE"] = 10897,
4679 ["glE"] = 10898,
4680 ["lesges"] = 10899,
4681 ["gesles"] = 10900,
4682 ["els"] = 10901,
4683 ["eqslantless"] = 10901,
4684 ["egs"] = 10902,
4685 ["eqslantgtr"] = 10902,
4686 ["elsdot"] = 10903,
4687 ["egsdot"] = 10904,
4688 ["el"] = 10905,
4689 ["eg"] = 10906,
4690 ["siml"] = 10909,
4691 ["simg"] = 10910,
4692 ["simlE"] = 10911,
4693 ["simgE"] = 10912,
4694 ["LessLess"] = 10913,
4695 ["GreaterGreater"] = 10914,
4696 ["glj"] = 10916,
4697 ["gla"] = 10917,
4698 ["ltcc"] = 10918,
4699 ["gtcc"] = 10919,
4700 ["lescc"] = 10920,
4701 ["gescc"] = 10921,
4702 ["smt"] = 10922,
4703 ["lat"] = 10923,
4704 ["smtE"] = 10924,
4705 ["late"] = 10925,
4706 ["bumpE"] = 10926,
4707 ["pre"] = 10927,
4708 ["preceq"] = 10927,
4709 ["PrecedesEqual"] = 10927,
4710 ["sce"] = 10928,
4711 ["succeq"] = 10928,
4712 ["SucceedsEqual"] = 10928,
4713 ["prE"] = 10931,
4714 ["scE"] = 10932,

4715 ["prnE"] = 10933,
 4716 ["precneqq"] = 10933,
 4717 ["scnE"] = 10934,
 4718 ["succneqq"] = 10934,
 4719 ["prap"] = 10935,
 4720 ["precapprox"] = 10935,
 4721 ["scap"] = 10936,
 4722 ["succapprox"] = 10936,
 4723 ["prnap"] = 10937,
 4724 ["precnapprox"] = 10937,
 4725 ["scnap"] = 10938,
 4726 ["succnapprox"] = 10938,
 4727 ["Pr"] = 10939,
 4728 ["Sc"] = 10940,
 4729 ["subdot"] = 10941,
 4730 ["supdot"] = 10942,
 4731 ["subplus"] = 10943,
 4732 ["supplus"] = 10944,
 4733 ["submult"] = 10945,
 4734 ["supmult"] = 10946,
 4735 ["subedot"] = 10947,
 4736 ["supedot"] = 10948,
 4737 ["subE"] = 10949,
 4738 ["subseteqq"] = 10949,
 4739 ["supE"] = 10950,
 4740 ["supseteqq"] = 10950,
 4741 ["subsim"] = 10951,
 4742 ["supsim"] = 10952,
 4743 ["subnE"] = 10955,
 4744 ["subsetneqq"] = 10955,
 4745 ["supnE"] = 10956,
 4746 ["supsetneqq"] = 10956,
 4747 ["csub"] = 10959,
 4748 ["csup"] = 10960,
 4749 ["csube"] = 10961,
 4750 ["csupe"] = 10962,
 4751 ["subsup"] = 10963,
 4752 ["supsub"] = 10964,
 4753 ["subsub"] = 10965,
 4754 ["supsup"] = 10966,
 4755 ["suphsub"] = 10967,
 4756 ["supdsub"] = 10968,
 4757 ["forkv"] = 10969,
 4758 ["topfork"] = 10970,
 4759 ["mlcp"] = 10971,
 4760 ["Dashv"] = 10980,
 4761 ["DoubleLeftTee"] = 10980,

4762 ["Vdashl"] = 10982,
4763 ["Barv"] = 10983,
4764 ["vBar"] = 10984,
4765 ["vBarv"] = 10985,
4766 ["Vbar"] = 10987,
4767 ["Not"] = 10988,
4768 ["bNot"] = 10989,
4769 ["rnmid"] = 10990,
4770 ["cirmid"] = 10991,
4771 ["midcir"] = 10992,
4772 ["topcir"] = 10993,
4773 ["nhpar"] = 10994,
4774 ["parsim"] = 10995,
4775 ["parsl"] = 11005,
4776 ["fflig"] = 64256,
4777 ["filig"] = 64257,
4778 ["fllig"] = 64258,
4779 ["ffilig"] = 64259,
4780 ["ffllig"] = 64260,
4781 ["Ascr"] = 119964,
4782 ["Cscr"] = 119966,
4783 ["Dscr"] = 119967,
4784 ["Gscr"] = 119970,
4785 ["Jscr"] = 119973,
4786 ["Kscr"] = 119974,
4787 ["Nscr"] = 119977,
4788 ["Oscr"] = 119978,
4789 ["Pscr"] = 119979,
4790 ["Qscr"] = 119980,
4791 ["Sscr"] = 119982,
4792 ["Tscr"] = 119983,
4793 ["Uscr"] = 119984,
4794 ["Vscr"] = 119985,
4795 ["Wscr"] = 119986,
4796 ["Xscr"] = 119987,
4797 ["Yscr"] = 119988,
4798 ["Zscr"] = 119989,
4799 ["ascr"] = 119990,
4800 ["bscr"] = 119991,
4801 ["cscr"] = 119992,
4802 ["dscr"] = 119993,
4803 ["fscr"] = 119995,
4804 ["hscr"] = 119997,
4805 ["iscr"] = 119998,
4806 ["jscr"] = 119999,
4807 ["kscr"] = 120000,
4808 ["lscr"] = 120001,

4809 ["mscr"] = 120002,
4810 ["nscr"] = 120003,
4811 ["pscr"] = 120005,
4812 ["qscr"] = 120006,
4813 ["rscr"] = 120007,
4814 ["sscr"] = 120008,
4815 ["tscr"] = 120009,
4816 ["uscr"] = 120010,
4817 ["vscr"] = 120011,
4818 ["wscr"] = 120012,
4819 ["xscr"] = 120013,
4820 ["yscr"] = 120014,
4821 ["zscr"] = 120015,
4822 ["Afr"] = 120068,
4823 ["Bfr"] = 120069,
4824 ["Dfr"] = 120071,
4825 ["Efr"] = 120072,
4826 ["Ffr"] = 120073,
4827 ["Gfr"] = 120074,
4828 ["Jfr"] = 120077,
4829 ["Kfr"] = 120078,
4830 ["Lfr"] = 120079,
4831 ["Mfr"] = 120080,
4832 ["Nfr"] = 120081,
4833 ["Ofr"] = 120082,
4834 ["Pfr"] = 120083,
4835 ["Qfr"] = 120084,
4836 ["Sfr"] = 120086,
4837 ["Tfr"] = 120087,
4838 ["Ufr"] = 120088,
4839 ["Vfr"] = 120089,
4840 ["Wfr"] = 120090,
4841 ["Xfr"] = 120091,
4842 ["Yfr"] = 120092,
4843 ["afr"] = 120094,
4844 ["bfr"] = 120095,
4845 ["cfr"] = 120096,
4846 ["dfr"] = 120097,
4847 ["efr"] = 120098,
4848 ["ffr"] = 120099,
4849 ["gfr"] = 120100,
4850 ["hfr"] = 120101,
4851 ["ifr"] = 120102,
4852 ["jfr"] = 120103,
4853 ["kfr"] = 120104,
4854 ["lfr"] = 120105,
4855 ["mfr"] = 120106,

4856 ["nfr"] = 120107,
4857 ["ofr"] = 120108,
4858 ["pfr"] = 120109,
4859 ["qfr"] = 120110,
4860 ["rfr"] = 120111,
4861 ["sfr"] = 120112,
4862 ["tfr"] = 120113,
4863 ["ufr"] = 120114,
4864 ["vfr"] = 120115,
4865 ["wfr"] = 120116,
4866 ["xfr"] = 120117,
4867 ["yfr"] = 120118,
4868 ["zfr"] = 120119,
4869 ["Aopf"] = 120120,
4870 ["Bopf"] = 120121,
4871 ["Dopf"] = 120123,
4872 ["Eopf"] = 120124,
4873 ["Fopf"] = 120125,
4874 ["Gopf"] = 120126,
4875 ["Iopf"] = 120128,
4876 ["Jopf"] = 120129,
4877 ["Kopf"] = 120130,
4878 ["Lopf"] = 120131,
4879 ["Mopf"] = 120132,
4880 ["Oopf"] = 120134,
4881 ["Sopf"] = 120138,
4882 ["Topf"] = 120139,
4883 ["Uopf"] = 120140,
4884 ["Vopf"] = 120141,
4885 ["Wopf"] = 120142,
4886 ["Xopf"] = 120143,
4887 ["Yopf"] = 120144,
4888 ["aopf"] = 120146,
4889 ["bopf"] = 120147,
4890 ["copf"] = 120148,
4891 ["dopf"] = 120149,
4892 ["eopf"] = 120150,
4893 ["fopf"] = 120151,
4894 ["gopf"] = 120152,
4895 ["hopf"] = 120153,
4896 ["iopf"] = 120154,
4897 ["jopf"] = 120155,
4898 ["kopf"] = 120156,
4899 ["lopf"] = 120157,
4900 ["mopf"] = 120158,
4901 ["nopf"] = 120159,
4902 ["oopf"] = 120160,

```

4903 ["popf"] = 120161,
4904 ["qopf"] = 120162,
4905 ["ropf"] = 120163,
4906 ["sopf"] = 120164,
4907 ["topf"] = 120165,
4908 ["uopf"] = 120166,
4909 ["vopf"] = 120167,
4910 ["wopf"] = 120168,
4911 ["xopf"] = 120169,
4912 ["yopf"] = 120170,
4913 ["zopf"] = 120171,
4914 }

```

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

```

4915 function entities.dec_entity(s)
4916     return unicode.utf8.char(tonumber(s))
4917 end

```

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

```

4918 function entities.hex_entity(s)
4919     return unicode.utf8.char(tonumber("0x"..s))
4920 end

```

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

```

4921 function entities.char_entity(s)
4922     local n = character_entities[s]
4923     if n == nil then
4924         return "&" .. s .. ";"
4925     end
4926     return unicode.utf8.char(n)
4927 end

```

3.1.3 Plain T_EX Writer

This section documents the `writer` object, which implements the routines for producing the T_EX output. The object is an amalgamate of the generic, T_EX, L^AT_EX 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.

```

4928 M.writer = {}

```

The `writer.new` method creates and returns a new TeX 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.

```
4929 function M.writer.new(options)
4930   local self = {}
```

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

```
4931   self.options = options
```

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

```
4932   local slice_specifiers = {}
4933   for specifier in options.slice:gmatch("[^%s]+") do
4934     table.insert(slice_specifiers, specifier)
4935   end
4936
4937   if #slice_specifiers == 2 then
4938     self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
4939     local slice_begin_type = self.slice_begin:sub(1, 1)
4940     if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
4941       self.slice_begin = "^" .. self.slice_begin
4942     end
4943     local slice_end_type = self.slice_end:sub(1, 1)
4944     if slice_end_type ~= "^" and slice_end_type ~= "$" then
4945       self.slice_end = "$" .. self.slice_end
4946     end
4947   elseif #slice_specifiers == 1 then
4948     self.slice_begin = "^" .. slice_specifiers[1]
4949     self.slice_end = "$" .. slice_specifiers[1]
4950   end
4951
4952   self.slice_begin_type = self.slice_begin:sub(1, 1)
4953   self.slice_begin_identifier = self.slice_begin:sub(2) or ""
4954   self.slice_end_type = self.slice_end:sub(1, 1)
4955   self.slice_end_identifier = self.slice_end:sub(2) or ""
4956
4957   if self.slice_begin == "^" and self.slice_end ~= "^" then
4958     self.is_writing = true
4959   else
4960     self.is_writing = false
4961   end
```

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

```
4962 self.suffix = ".tex"
```

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

```
4963 self.space = " "
```

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

```
4964 self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
4965 function self.plain(s)
4966   return s
4967 end
```

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

```
4968 function self.paragraph(s)
4969   if not self.is_writing then return "" end
4970   return s
4971 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.

```
4972 function self.pack(name)
4973   return [[\input{]] .. name .. [[]\relax]]
4974 end
```

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

```
4975 function self.interblocksep()
4976   if not self.is_writing then return "" end
4977   return "\\markdownRendererInterblockSeparator\n{}"
4978 end
```

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

```
4979 self.hard_line_break = "\\markdownRendererHardLineBreak\n{}"
```

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

```
4980 self.ellipsis = "\\markdownRendererEllipsis{}"
```

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

```
4981 function self.thematic_break()
4982   if not self.is_writing then return "" end
4983   return "\\markdownRendererThematicBreak{}"
4984 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.

```
4985 self.escaped_uri_chars = {
```

```

4986     [{""] = "\\markdownRendererLeftBrace{}",
4987     ["}"] = "\\markdownRendererRightBrace{}",
4988     [{"\\"} = "\\markdownRendererBackslash{}",
4989     }
4990     self.escaped_minimal_strings = {
4991         ["^"] = "\\markdownRendererCircumflex\\markdownRendererCircumflex ",
4992         [{"☒"} = "\\markdownRendererTickedBox{}",
4993         [{"☐"} = "\\markdownRendererHalfTickedBox{}",
4994         [{"□"} = "\\markdownRendererUntickedBox{}",
4995         [entities.hex_entity('FFFD')] = "\\markdownRendererReplacementCharacter{}",
4996     }

```

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

```

4997     self.escaped_strings = util.table_copy(self.escaped_minimal_strings)
4998     self.escaped_strings[entities.hex_entity('00A0')] = self.nbsp

```

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

```

4999     self.escaped_chars = {
5000         [{""] = "\\markdownRendererLeftBrace{}",
5001         ["}"] = "\\markdownRendererRightBrace{}",
5002         [{"%"} = "\\markdownRendererPercentSign{}",
5003         [{"\\"} = "\\markdownRendererBackslash{}",
5004         [{"#"} = "\\markdownRendererHash{}",
5005         [{"$"} = "\\markdownRendererDollarSign{}",
5006         [{"&"} = "\\markdownRendererAmpersand{}",
5007         [{"_"} = "\\markdownRendererUnderscore{}",
5008         [{"^"} = "\\markdownRendererCircumflex{}",
5009         [{"~"} = "\\markdownRendererTilde{}",
5010         [{"|"} = "\\markdownRendererPipe{}",
5011         [entities.hex_entity('0000')] = "\\markdownRendererReplacementCharacter{}",
5012     }

```

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.

```

5013     local escape_typographic_text = util.escaper(
5014         self.escaped_chars, self.escaped_strings)
5015     local escape_programmatic_text = util.escaper(
5016         self.escaped_uri_chars, self.escaped_minimal_strings)
5017     local escape_minimal = util.escaper(
5018         {}, 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.

```

5019 self.escape = escape_typographic_text
5020 self.math = escape_minimal
5021 if options.hybrid then
5022     self.identifier = escape_minimal
5023     self.string = escape_minimal
5024     self.uri = escape_minimal
5025 else
5026     self.identifier = escape_programmatic_text
5027     self.string = escape_typographic_text
5028     self.uri = escape_programmatic_text
5029 end

```

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

```

5030 function self.code(s)
5031     return {"\\markdownRendererCodeSpan{" ,self.escape(s),"}"}
5032 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, and `tit` to the title of the link.

```

5033 function self.link(lab,src,tit)
5034     return {"\\markdownRendererLink{" ,lab,"} ",
5035           "{" ,self.escape(src),"} ",
5036           "{" ,self.uri(src),"} ",
5037           "{" ,self.string(tit or ""),""}
5038 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, and `tit` to the title of the image.

```

5039 function self.image(lab,src,tit)
5040     return {"\\markdownRendererImage{" ,lab,"} ",
5041           "{" ,self.string(src),"} ",
5042           "{" ,self.uri(src),"} ",
5043           "{" ,self.string(tit or ""),""}
5044 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.

```

5045 function self.bulletlist(items,tight)
5046   if not self.is_writing then return "" end
5047   local buffer = {}
5048   for _,item in ipairs(items) do
5049     buffer[#buffer + 1] = self.bulletitem(item)
5050   end
5051   local contents = util.intersperse(buffer,"\n")
5052   if tight and options.tightLists then
5053     return {"\\markdownRendererUlBeginTight\n",contents,
5054           "\n\\markdownRendererUlEndTight "}
5055   else
5056     return {"\\markdownRendererUlBegin\n",contents,
5057           "\n\\markdownRendererUlEnd "}
5058   end
5059 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.

```

5060 function self.bulletitem(s)
5061   return {"\\markdownRendererUlItem ",s,
5062         "\\markdownRendererUlItemEnd "}
5063 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.

```

5064 function self.orderedlist(items,tight,startnum)
5065   if not self.is_writing then return "" end
5066   local buffer = {}
5067   local num = startnum
5068   for _,item in ipairs(items) do
5069     buffer[#buffer + 1] = self.ordereditem(item,num)
5070     if num ~= nil then
5071       num = num + 1
5072     end
5073   end
5074   local contents = util.intersperse(buffer,"\n")
5075   if tight and options.tightLists then
5076     return {"\\markdownRendererOlBeginTight\n",contents,
5077           "\n\\markdownRendererOlEndTight "}
5078   else
5079     return {"\\markdownRendererOlBegin\n",contents,
5080           "\n\\markdownRendererOlEnd "}
5081   end
5082 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.

```
5083 function self.ordereditem(s,num)
5084   if num ~= nil then
5085     return {"\\markdownRendererOItemWithNumber{" ,num,"} ",s,
5086           "\\markdownRendererOItemEnd "}
5087   else
5088     return {"\\markdownRendererOItem ",s,
5089           "\\markdownRendererOItemEnd "}
5090   end
5091 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.

```
5092 function self.inline_html_comment(contents)
5093   return {"\\markdownRendererInlineHtmlComment{" ,contents,"}"}
5094 end
```

Define `writer->block_html_comment` as a function that will transform the contents of a block HTML comment, to the output format, where `contents` are the contents of the HTML comment.

```
5095 function self.block_html_comment(contents)
5096   if not self.is_writing then return "" end
5097   return {"\\markdownRendererBlockHtmlCommentBegin\n",contents,
5098         "\n\\markdownRendererBlockHtmlCommentEnd "}
5099 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.

```
5100 function self.inline_html_tag(contents)
5101   return {"\\markdownRendererInlineHtmlTag{" ,self.string(contents),"}"}
5102 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.

```
5103 function self.block_html_element(s)
5104   if not self.is_writing then return "" end
5105   local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
5106   return {"\\markdownRendererInputBlockHtmlElement{" ,name,"}"}
5107 end
```

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


```

5108 function self.emphasis(s)
5109     return {"\\markdownRendererEmphasis{" ,s,"}"}
5110 end

```

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

```

5111 function self.checkbox(f)
5112     if f == 1.0 then
5113         return "☒ "
5114     elseif f == 0.0 then
5115         return "☐ "
5116     else
5117         return "◻ "
5118     end
5119 end

```

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

```

5120 function self.strong(s)
5121     return {"\\markdownRendererStrongEmphasis{" ,s,"}"}
5122 end

```

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

```

5123 function self.blockquote(s)
5124     if #util.rope_to_string(s) == 0 then return "" end
5125     return {"\\markdownRendererBlockQuoteBegin\n",s,
5126           "\n\\markdownRendererBlockQuoteEnd "}
5127 end

```

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

```

5128 function self.verbatim(s)
5129     if not self.is_writing then return "" end
5130     s = s:gsub("\\n$", "")
5131     local name = util.cache_verbatim(options.cacheDir, s)
5132     return {"\\markdownRendererInputVerbatim{" ,name,"}"}
5133 end

```

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

```

5134 function self.document(d)
5135     local buf = {"\\markdownRendererDocumentBegin\n", d}
5136
5137     -- pop all attributes
5138     table.insert(buf, self.pop_attributes())
5139
5140     table.insert(buf, "\\markdownRendererDocumentEnd")

```

```

5141
5142     return buf
5143 end

```

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

```

5144 function self.attributes(attr)
5145     local buf = {}
5146
5147     table.sort(attr)
5148     local key, value
5149     for i = 1, #attr do
5150         if attr[i]:sub(1, 1) == "#" then
5151             table.insert(buf, {"\\markdownRendererAttributeIdentifier{" ,
5152                                 attr[i]:sub(2), "}"})
5153         elseif attr[i]:sub(1, 1) == "." then
5154             table.insert(buf, {"\\markdownRendererAttributeName{" ,
5155                                 attr[i]:sub(2), "}"})
5156         else
5157             key, value = attr[i]:match("(^[^= ]+)%s*=%s*(.*)")
5158             table.insert(buf, {"\\markdownRendererAttributeKeyValue{" ,
5159                                 key, "}{", value, "}"})
5160         end
5161     end
5162
5163     return buf
5164 end

```

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

```

5165     self.active_attributes = {}

```

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`.

```

5166 local function apply_attributes()
5167     local buf = {}
5168     for i = 1, #self.active_attributes do
5169         local start_output = self.active_attributes[i][3]
5170         if start_output ~= nil then
5171             table.insert(buf, start_output)
5172         end
5173     end
5174     return buf
5175 end
5176
5177 local function tear_down_attributes()
5178     local buf = {}

```

```

5179     for i = #self.active_attributes, 1, -1 do
5180         local end_output = self.active_attributes[i][4]
5181         if end_output ~= nil then
5182             table.insert(buf, end_output)
5183         end
5184     end
5185     return buf
5186 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.

```

5187     function self.push_attributes(attribute_type, attributes,
5188                                 start_output, end_output)
5189         -- index attributes in a hash table for easy lookup
5190         attributes = attributes or {}
5191         for i = 1, #attributes do
5192             attributes[attributes[i]] = true
5193         end
5194
5195         local buf = {}
5196         -- handle slicing
5197         if attributes["#" .. self.slice_end_identifer] ~= nil and
5198            self.slice_end_type == "^" then
5199             if self.is_writing then
5200                 table.insert(buf, tear_down_attributes())
5201             end
5202             self.is_writing = false
5203         end
5204         if attributes["#" .. self.slice_begin_identifer] ~= nil and
5205            self.slice_begin_type == "^" then
5206             self.is_writing = true
5207             table.insert(buf, apply_attributes())
5208             self.is_writing = true
5209         end
5210         if self.is_writing and start_output ~= nil then
5211             table.insert(buf, start_output)
5212         end
5213         table.insert(self.active_attributes,
5214                     {attribute_type, attributes,
5215                      start_output, end_output})
5216     return buf
5217 end
5218

```

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`).

```

5219 function self.pop_attributes(attribute_type)
5220   local buf = {}
5221   -- pop attributes until we find attributes of correct type
5222   -- or until no attributes remain
5223   local current_attribute_type = false
5224   while current_attribute_type ~= attribute_type and
5225     #self.active_attributes > 0 do
5226     local attributes, _, end_output
5227     current_attribute_type, attributes, _, end_output = table.unpack(
5228       self.active_attributes[#self.active_attributes])
5229     if self.is_writing and end_output ~= nil then
5230       table.insert(buf, end_output)
5231     end
5232     table.remove(self.active_attributes, #self.active_attributes)
5233     -- handle slicing
5234     if attributes["#" .. self.slice_end_identifier] ~= nil
5235       and self.slice_end_type == "$" then
5236       if self.is_writing then
5237         table.insert(buf, tear_down_attributes())
5238       end
5239       self.is_writing = false
5240     end
5241     if attributes["#" .. self.slice_begin_identifier] ~= nil and
5242       self.slice_begin_type == "$" then
5243       self.is_writing = true
5244       table.insert(buf, apply_attributes())
5245     end
5246   end
5247   return buf
5248 end

```

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

```

5249 local current_heading_level = 0
5250 function self.heading(s, level, attributes)
5251   local buf = {}
5252
5253   -- push empty attributes for implied sections
5254   while current_heading_level < level - 1 do
5255     table.insert(buf,
5256       self.push_attributes("heading",

```

```

5257             nil,
5258             "\\markdownRendererSectionBegin\n",
5259             "\n\\markdownRendererSectionEnd ")
5260     current_heading_level = current_heading_level + 1
5261 end
5262
5263 -- pop attributes for sections that have ended
5264 while current_heading_level >= level do
5265     table.insert(buf, self.pop_attributes("heading"))
5266     current_heading_level = current_heading_level - 1
5267 end
5268
5269 -- push attributes for the new section
5270 local start_output = {}
5271 local end_output = {}
5272 table.insert(start_output, "\\markdownRendererSectionBegin\n")
5273 if options.headerAttributes and attributes ~= nil and #attributes > 0 then
5274     table.insert(start_output,
5275                 "\\markdownRendererHeaderAttributeContextBegin\n")
5276     table.insert(start_output, self.attributes(attributes))
5277     table.insert(end_output,
5278                 "\n\\markdownRendererHeaderAttributeContextEnd ")
5279 end
5280 table.insert(end_output, "\n\\markdownRendererSectionEnd ")
5281
5282 table.insert(buf, self.push_attributes("heading",
5283                                     attributes,
5284                                     start_output,
5285                                     end_output))
5286 current_heading_level = current_heading_level + 1
5287 assert(current_heading_level == level)
5288
5289 -- produce the renderer
5290 local cmd
5291 level = level + options.shiftHeadings
5292 if level <= 1 then
5293     cmd = "\\markdownRendererHeadingOne"
5294 elseif level == 2 then
5295     cmd = "\\markdownRendererHeadingTwo"
5296 elseif level == 3 then
5297     cmd = "\\markdownRendererHeadingThree"
5298 elseif level == 4 then
5299     cmd = "\\markdownRendererHeadingFour"
5300 elseif level == 5 then
5301     cmd = "\\markdownRendererHeadingFive"
5302 elseif level >= 6 then
5303     cmd = "\\markdownRendererHeadingSix"

```

```

5304     else
5305         cmd = ""
5306     end
5307     if self.is_writing then
5308         table.insert(buf, {cmd, "{", s, "}"})
5309     end
5310
5311     return buf
5312 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.

```

5313 function self.get_state()
5314     return {
5315         is_writing=self.is_writing,
5316         active_attributes={table.unpack(self.active_attributes)},
5317     }
5318 end

```

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

```

5319 function self.set_state(s)
5320     local previous_state = self.get_state()
5321     for key, value in pairs(s) do
5322         self[key] = value
5323     end
5324     return previous_state
5325 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`.

```

5326 function self.defer_call(f)
5327     local previous_state = self.get_state()
5328     return function(...)
5329         local state = self.set_state(previous_state)
5330         local return_value = f(...)
5331         self.set_state(state)
5332         return return_value
5333     end
5334 end
5335
5336 return self
5337 end

```

3.1.4 Parsers

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

```
5338 local parsers = {}
```

3.1.4.1 Basic Parsers

```
5339 parsers.percent = P("%")
5340 parsers.at = P("@")
5341 parsers.comma = P(",")
5342 parsers.asterisk = P("*")
5343 parsers.dash = P("-")
5344 parsers.plus = P("+")
5345 parsers.underscore = P("_")
5346 parsers.period = P(".")
5347 parsers.hash = P("#")
5348 parsers.dollar = P("$")
5349 parsers.ampersand = P("&")
5350 parsers.backtick = P("`")
5351 parsers.less = P("<")
5352 parsers.more = P(">")
5353 parsers.space = P(" ")
5354 parsers.squote = P("'")
5355 parsers.dquote = P('"')
5356 parsers.lparent = P("(")
5357 parsers.rparent = P(")")
5358 parsers.lbracket = P("[")
5359 parsers.rbracket = P("]")
5360 parsers.lbrace = P("{")
5361 parsers.rbrace = P("}")
5362 parsers.circumflex = P("^")
5363 parsers.slash = P("/")
5364 parsers.equal = P("=")
5365 parsers.colon = P(":")
5366 parsers.semicolon = P(";")
5367 parsers.exclamation = P("!")
5368 parsers.pipe = P("|")
5369 parsers.tilde = P("~")
5370 parsers.backslash = P("\\")
5371 parsers.tab = P("\t")
5372 parsers.newline = P("\n")
5373 parsers.tightblocksep = P("\001")
5374
5375 parsers.digit = R("09")
5376 parsers.hexdigit = R("09", "af", "AF")
5377 parsers.letter = R("AZ", "az")
```

```

5378 parsers.alphanumeric      = R("AZ","az","09")
5379 parsers.keyword          = parsers.letter
5380                          * parsers.alphanumeric^0
5381 parsers.internal_punctuation = S(":,;,.?")
5382
5383 parsers.doubleasterisks     = P("**")
5384 parsers.doubleunderscores   = P("__")
5385 parsers.doubletildes       = P("~~")
5386 parsers.fourspace         = P("    ")
5387
5388 parsers.any                 = P(1)
5389 parsers.succeed            = P(true)
5390 parsers.fail                = P(false)
5391
5392 parsers.escapable          = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")
5393 parsers.anyescaped         = parsers.backslash / "\"" * parsers.escapable
5394                          + parsers.any
5395
5396 parsers.spacechar          = S("\t ")
5397 parsers.spacing            = S(" \n\r\t")
5398 parsers.nospacechar        = parsers.any - parsers.spacing
5399 parsers.optionalspace     = parsers.spacechar^0
5400
5401 parsers.normalchar         = parsers.any - (V("SpecialChar")
5402                          + parsers.spacing
5403                          + parsers.tightblocksep)
5404 parsers.eof                 = -parsers.any
5405 parsers.nonindentspace     = parsers.space^-3 * -parsers.spacechar
5406 parsers.indent             = parsers.space^-3 * parsers.tab
5407                          + parsers.fourspace / "\""
5408 parsers.linechar           = P(1 - parsers.newline)
5409
5410 parsers.blankline          = parsers.optionalspace
5411                          * parsers.newline / "\n"
5412 parsers.blanklines         = parsers.blankline^0
5413 parsers.skipblanklines     = (parsers.optionalspace * parsers.newline)^0
5414 parsers.indentedline       = parsers.indent / "\""
5415                          * C(parsers.linechar^1 * parsers.newline^-
5416                          1)
5416 parsers.optionallyindentedline = parsers.indent^-1 / "\""
5417                          * C(parsers.linechar^1 * parsers.newline^-
5418                          1)
5418 parsers.sp                  = parsers.spacing^0
5419 parsers.spnl                = parsers.optionalspace
5420                          * (parsers.newline * parsers.optionalspace)^-
5421                          1
5421 parsers.line                = parsers.linechar^0 * parsers.newline

```



```
5422 parsers.nonemptyline = parsers.line - parsers.blankline
```

The `parsers.commented_line1` parser recognizes the regular language of T_EX comments, see an equivalent finite automaton in Figure 6.

```
5423 parsers.commented_line_letter = parsers.linechar
5424 + parsers.newline
5425 - parsers.backslash
5426 - parsers.percent
5427 parsers.commented_line = Cg(Cc(""), "backslashes")
5428 * ((#(parsers.commented_line_letter
5429     - parsers.newline)
5430    * Cb("backslashes")
5431    * Cs(parsers.commented_line_letter
5432        - parsers.newline)1 -- initial
5433    * Cg(Cc(""), "backslashes"))
5434 + #(parsers.backslash * parsers.backslash
5435    * Cg((parsers.backslash -- even backslash
5436        * parsers.backslash)1, "backslashes")
5437 + (parsers.backslash
5438    * (#parsers.percent
5439      * Cb("backslashes")
5440      / function(backslashes)
5441        return string.rep("\\", #backslashes / 2)
5442      end
5443      * C(parsers.percent)
5444      + #parsers.commented_line_letter
5445      * Cb("backslashes")
5446      * Cc("\\")
5447      * C(parsers.commented_line_letter))
5448    * Cg(Cc(""), "backslashes"))0
5449 * (#parsers.percent
5450   * Cb("backslashes")
5451   / function(backslashes)
5452     return string.rep("\\", #backslashes / 2)
5453   end
5454 * ((parsers.percent -- comment
5455     * parsers.line
5456     * #parsers.blankline) -- blank line
5457   / "\n"
5458   + parsers.percent -- comment
5459   * parsers.line
5460   * parsers.optionalspace) -- leading tabs and space
5461 + #(parsers.newline)
5462 * Cb("backslashes")
5463 * C(parsers.newline))
5464
5465 parsers.chunk = parsers.line * (parsers.optionallyindentedline
```

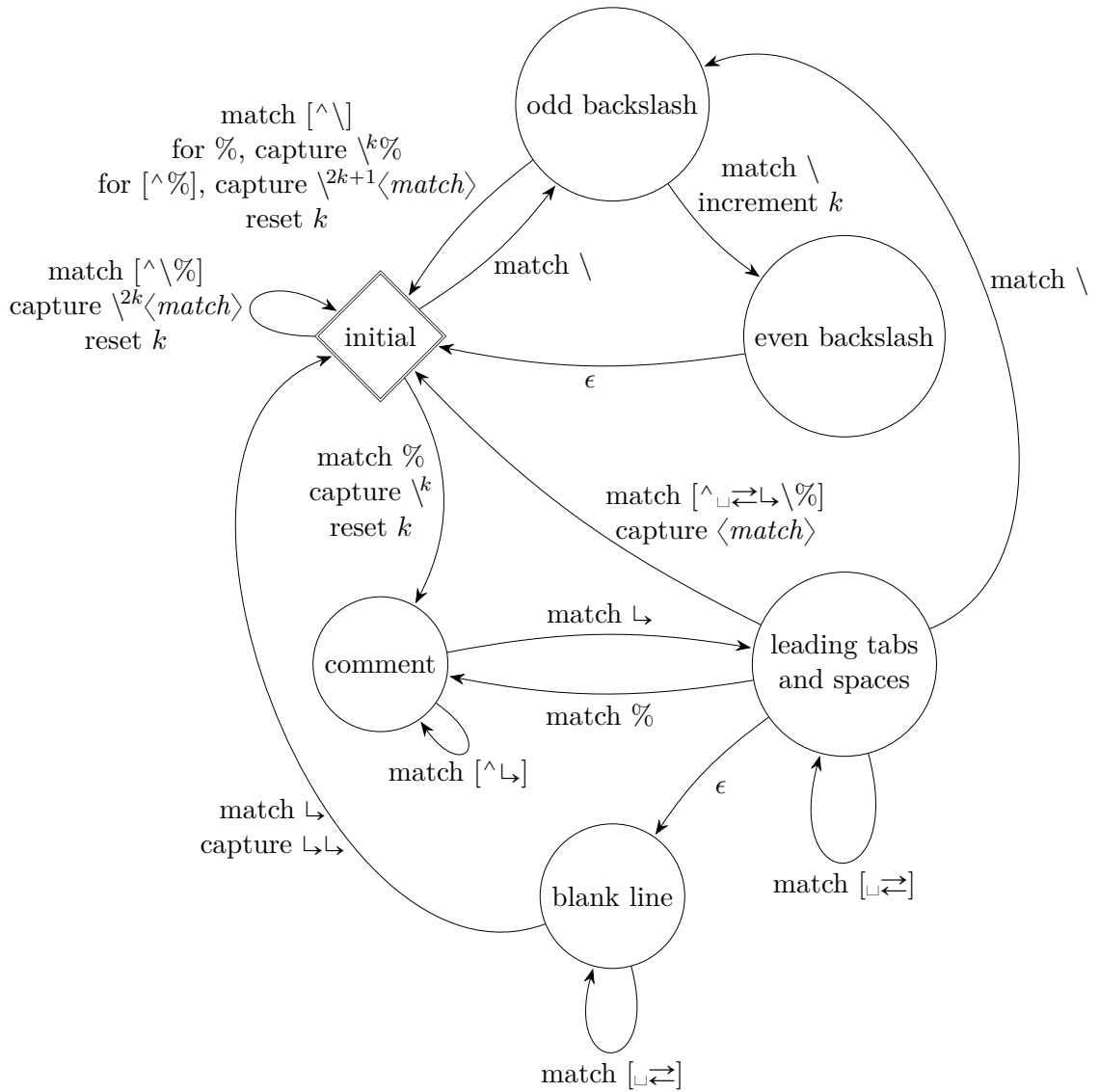


Figure 6: A pushdown automaton that recognizes TeX comments

```

5466                                     - parsers.blankline)^0
5467
5468 parsers.attribute_key_char      = parsers.alphanumeric + S("_-")
5469 parsers.attribute_key           = (parsers.attribute_key_char
5470                                   - parsers.dash - parsers.digit)
5471                                   * parsers.attribute_key_char^0
5472 parsers.attribute_value         = ( (parsers.dquote / "'")
5473                                   * (parsers.anyescaped - parsers.dquote)^0
5474                                   * (parsers.dquote / "'"))
5475                                   + ( parsers.anyescaped - parsers.dquote - parsers.rbra
5476                                   - parsers.space)^0
5477
5478 parsers.attribute = (parsers.dash * Cc(".unnumbered"))
5479                   + C((parsers.hash + parsers.period)
5480                       * parsers.attribute_key)
5481                   + Cs( parsers.attribute_key
5482                       * parsers.optionalspace * parsers.equal * parsers.optionalspace
5483                       * parsers.attribute_value)
5484 parsers.attributes = parsers.lbrace
5485                   * parsers.optionalspace
5486                   * parsers.attribute
5487                   * (parsers.spacechar^1
5488                   * parsers.attribute)^0
5489                   * parsers.optionalspace
5490                   * parsers.rbrace
5491
5492
5493 parsers.raw_attribute = parsers.lbrace
5494                   * parsers.optionalspace
5495                   * parsers.equal
5496                   * C(parsers.attribute_key)
5497                   * parsers.optionalspace
5498                   * parsers.rbrace
5499
5500 -- block followed by 0 or more optionally
5501 -- indented blocks with first line indented.
5502 parsers.indented_blocks = function(bl)
5503   return Cs( bl
5504             * (parsers.blankline^1 * parsers.indent * -parsers.blankline * bl)^0
5505             * (parsers.blankline^1 + parsers.eof) )
5506 end

```

3.1.4.2 Parsers Used for Markdown Lists

```

5507 parsers.bulletchar = C(parsers.plus + parsers.asterisk + parsers.dash)
5508
5509 parsers.bullet = ( parsers.bulletchar * #parsers.spacing

```

```

5510             * (parsers.tab + parsers.space^-
3)
5511         + parsers.space * parsers.bulletchar * #parsers.spacing
5512             * (parsers.tab + parsers.space^-2)
5513         + parsers.space * parsers.space * parsers.bulletchar
5514             * #parsers.spacing
5515             * (parsers.tab + parsers.space^-1)
5516         + parsers.space * parsers.space * parsers.space
5517             * parsers.bulletchar * #parsers.spacing
5518     )
5519
5520 local function tickbox(interior)
5521     return parsers.optionalspace * parsers.lbracket
5522         * interior * parsers.rbracket * parsers.spacechar^1
5523 end
5524
5525 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
5526 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
5527 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
5528

```

3.1.4.3 Parsers Used for Markdown Code Spans

```

5529 parsers.openticks = Cg(parsers.backtick^1, "ticks")
5530
5531 local function captures_equal_length(_,i,a,b)
5532     return #a == #b and i
5533 end
5534
5535 parsers.closeticks = parsers.space^-1
5536                 * Cmt(C(parsers.backtick^1)
5537                     * Cb("ticks"), captures_equal_length)
5538
5539 parsers.intickschar = (parsers.any - S("\n\r`"))
5540                 + (parsers.newline * -parsers.blankline)
5541                 + (parsers.space - parsers.closeticks)
5542                 + (parsers.backtick^1 - parsers.closeticks)
5543
5544 parsers.inticks = parsers.openticks * parsers.space^-1
5545                 * C(parsers.intickschar^0) * parsers.closeticks

```

3.1.4.4 Parsers Used for Markdown Tags and Links

```

5546 parsers.leader = parsers.space^-3
5547
5548 -- content in balanced brackets, parentheses, or quotes:
5549 parsers.bracketed = P{ parsers.lbracket
5550                 * (( parsers.backslash / '"' * parsers.rbracket

```

```

5551             + parsers.any - (parsers.lbracket
5552                               + parsers.rbracket
5553                               + parsers.blankline^2)
5554             ) + V(1))^0
5555             * parsers.rbracket }
5556
5557 parsers.inparens = P{ parsers.lparent
5558                       * ((parsers.anyescaped - (parsers.lparent
5559                                                         + parsers.rparent
5560                                                         + parsers.blankline^2)
5561                       ) + V(1))^0
5562                       * parsers.rparent }
5563
5564 parsers.squoted   = P{ parsers.squote * parsers.alphanumeric
5565                       * ((parsers.anyescaped - (parsers.squote
5566                                                         + parsers.blankline^2)
5567                       ) + V(1))^0
5568                       * parsers.squote }
5569
5570 parsers.dquoted   = P{ parsers.dquote * parsers.alphanumeric
5571                       * ((parsers.anyescaped - (parsers.dquote
5572                                                         + parsers.blankline^2)
5573                       ) + V(1))^0
5574                       * parsers.dquote }
5575
5576 -- bracketed tag for markdown links, allowing nested brackets:
5577 parsers.tag       = parsers.lbracket
5578                   * Cs((parsers.alphanumeric^1
5579                         + parsers.bracketed
5580                         + parsers.inticks
5581                         + ( parsers.backslash / "\"" * parsers.rbracket
5582                         + parsers.any
5583                         - (parsers.rbracket + parsers.blankline^2))))^0
5584                   * parsers.rbracket
5585
5586 -- url for markdown links, allowing nested brackets:
5587 parsers.url       = parsers.less * Cs((parsers.anyescaped
5588                                       - parsers.more)^0)
5589                                       * parsers.more
5590                   + Cs((parsers.inparens + (parsers.anyescaped
5591                                       - parsers.spacing
5592                                       - parsers.rparent))^1)
5593
5594 -- quoted text, possibly with nested quotes:
5595 parsers.title_s   = parsers.squote * Cs(((parsers.anyescaped-parsers.squote)
5596                                       + parsers.squoted)^0)
5597                                       * parsers.squote

```

```

5598
5599 parsers.title_d      = parsers.dquote * Cs(((parsers.anyescaped-parsers.dquote)
5600                               + parsers.dquoted)^0)
5601                               * parsers.dquote
5602
5603 parsers.title_p      = parsers.lparent
5604                               * Cs((parsers.inparens + (parsers.anyescaped-parsers.rparent))^0)
5605                               * parsers.rparent
5606
5607 parsers.title        = parsers.title_d + parsers.title_s + parsers.title_p
5608
5609 parsers.optionaltitle
5610                       = parsers.spnl * parsers.title * parsers.spacechar^0
5611                       + Cc("")

```

3.1.4.5 Parsers Used for HTML

```

5612 -- case-insensitive match (we assume s is lowercase). must be single byte encoding
5613 parsers.keyword_exact = function(s)
5614   local parser = P(0)
5615   for i=1,#s do
5616     local c = s:sub(i,i)
5617     local m = c .. upper(c)
5618     parser = parser * S(m)
5619   end
5620   return parser
5621 end
5622
5623 parsers.block_keyword =
5624   parsers.keyword_exact("address") + parsers.keyword_exact("blockquote") +
5625   parsers.keyword_exact("center") + parsers.keyword_exact("del") +
5626   parsers.keyword_exact("dir") + parsers.keyword_exact("div") +
5627   parsers.keyword_exact("p") + parsers.keyword_exact("pre") +
5628   parsers.keyword_exact("li") + parsers.keyword_exact("ol") +
5629   parsers.keyword_exact("ul") + parsers.keyword_exact("dl") +
5630   parsers.keyword_exact("dd") + parsers.keyword_exact("form") +
5631   parsers.keyword_exact("fieldset") + parsers.keyword_exact("isindex") +
5632   parsers.keyword_exact("ins") + parsers.keyword_exact("menu") +
5633   parsers.keyword_exact("noframes") + parsers.keyword_exact("frameset") +
5634   parsers.keyword_exact("h1") + parsers.keyword_exact("h2") +
5635   parsers.keyword_exact("h3") + parsers.keyword_exact("h4") +
5636   parsers.keyword_exact("h5") + parsers.keyword_exact("h6") +
5637   parsers.keyword_exact("hr") + parsers.keyword_exact("script") +
5638   parsers.keyword_exact("noscript") + parsers.keyword_exact("table") +
5639   parsers.keyword_exact("tbody") + parsers.keyword_exact("tfoot") +
5640   parsers.keyword_exact("thead") + parsers.keyword_exact("th") +
5641   parsers.keyword_exact("td") + parsers.keyword_exact("tr")

```

```

5642
5643 -- There is no reason to support bad html, so we expect quoted attributes
5644 parsers.htmlattributevalue
5645         = parsers.squote * (parsers.any - (parsers.blankline
5646                                           + parsers.squote))^0
5647           * parsers.squote
5648         + parsers.dquote * (parsers.any - (parsers.blankline
5649                                           + parsers.dquote))^0
5650           * parsers.dquote
5651
5652 parsers.htmlattribute = parsers.spacing^1
5653                       * (parsers.alphanumeric + S("_-"))^1
5654                       * parsers.sp * parsers.equal * parsers.sp
5655                       * parsers.htmlattributevalue
5656
5657 parsers.htmlcomment  = P("<!--")
5658                       * parsers.optionalspace
5659                       * Cs((parsers.any - parsers.optionalspace * P("-->"))^0)
5660                       * parsers.optionalspace
5661                       * P("-->")
5662
5663 parsers.htmlinstruction = P("<?") * (parsers.any - P("?>"))^0 * P("?>")
5664
5665 parsers.openelt_any = parsers.less * parsers.keyword * parsers.htmlattribute^0
5666                   * parsers.sp * parsers.more
5667
5668 parsers.openelt_exact = function(s)
5669   return parsers.less * parsers.sp * parsers.keyword_exact(s)
5670         * parsers.htmlattribute^0 * parsers.sp * parsers.more
5671 end
5672
5673 parsers.openelt_block = parsers.sp * parsers.block_keyword
5674                   * parsers.htmlattribute^0 * parsers.sp * parsers.more
5675
5676 parsers.closeelt_any = parsers.less * parsers.sp * parsers.slash
5677                   * parsers.keyword * parsers.sp * parsers.more
5678
5679 parsers.closeelt_exact = function(s)
5680   return parsers.less * parsers.sp * parsers.slash * parsers.keyword_exact(s)
5681         * parsers.sp * parsers.more
5682 end
5683
5684 parsers.emptyelt_any = parsers.less * parsers.sp * parsers.keyword
5685                   * parsers.htmlattribute^0 * parsers.sp * parsers.slash
5686                   * parsers.more
5687
5688 parsers.emptyelt_block = parsers.less * parsers.sp * parsers.block_keyword

```

```

5689             * parsers.htmlattribute^0 * parsers.sp * parsers.slash
5690             * parsers.more
5691
5692 parsers.displaytext = (parsers.any - parsers.less)^1
5693
5694 -- return content between two matched HTML tags
5695 parsers.in_matched = function(s)
5696   return { parsers.openelt_exact(s)
5697           * (V(1) + parsers.displaytext
5698             + (parsers.less - parsers.closeelt_exact(s)))^0
5699           * parsers.closeelt_exact(s) }
5700 end
5701
5702 local function parse_matched_tags(s,pos)
5703   local t = string.lower(lpeg.match(C(parsers.keyword),s,pos))
5704   return lpeg.match(parsers.in_matched(t),s,pos-1)
5705 end
5706
5707 parsers.in_matched_block_tags = parsers.less
5708                               * Cmt(#parsers.openelt_block, parse_matched_tags)
5709

```

3.1.4.6 Parsers Used for HTML Entities

```

5710 parsers.hexentity = parsers.ampersand * parsers.hash * S("Xx")
5711                   * C(parsers.hexdigit^1) * parsers.semicolon
5712 parsers.decentity = parsers.ampersand * parsers.hash
5713                   * C(parsers.digit^1) * parsers.semicolon
5714 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
5715                   * parsers.semicolon

```

3.1.4.7 Helpers for References

```

5716 -- parse a reference definition: [foo]: /bar "title"
5717 parsers.define_reference_parser = parsers.leader * parsers.tag * parsers.colon
5718                               * parsers.spacechar^0 * parsers.url
5719                               * parsers.optionaltitle * parsers.blankline^1

```

3.1.4.8 Inline Elements

```

5720 parsers.Inline      = V("Inline")
5721 parsers.IndentedInline = V("IndentedInline")
5722
5723 -- parse many p between starter and ender
5724 parsers.between = function(p, starter, ender)
5725   local ender2 = B(parsers.nonspacechar) * ender
5726   return (starter * #parsers.nonspacechar * Ct(p * (p - ender2)^0) * ender2)
5727 end

```



```

5728
5729 parsers.urlchar      = parsers.anyescaped - parsers.newline - parsers.more

```

3.1.4.9 Block Elements

```

5730 parsers.lineof = function(c)
5731     return (parsers.leader * (P(c) * parsers.optionalspace)^3
5732           * (parsers.newline * parsers.blankline^1
5733             + parsers.newline^-1 * parsers.eof))
5734 end

```

3.1.4.10 Headings

```

5735 -- parse Atx heading start and return level
5736 parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
5737                       * -parsers.hash / length
5738
5739 -- parse setext header ending and return level
5740 parsers.heading_level = parsers.equal^1 * Cc(1) + parsers.dash^1 * Cc(2)
5741
5742 local function strip_atx_end(s)
5743     return s:gsub("#%s*\n$", "")
5744 end

```

3.1.5 Markdown Reader

This section documents the `reader` object, which implements the routines for parsing the markdown input. The object corresponds to the markdown reader object that was located in the `lunamark/reader/markdown.lua` file in the Lunamark Lua module.

The `reader.new` method creates and returns a new `TeX` reader object associated with the Lua interface options (see Section 2.1.3) `options` and with a writer object `writer`. When `options` are unspecified, it is assumed that an empty table was passed to the method.

The objects produced by the `reader.new` method expose instance methods and variables of their own. As a convention, I will refer to these *member*s as `reader-><member>`.

```

5745 M.reader = {}
5746 function M.reader.new(writer, options)
5747     local self = {}

```

Make the `writer` and `options` parameters available as `reader->writer` and `reader->options`, respectively, so that they are accessible from extensions.

```

5748     self.writer = writer
5749     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.

```

5750 self.parsers = {}
5751 (function(parsers)
5752     setmetatable(self.parsers, {
5753         __index = function (_, key)
5754             return parsers[key]
5755         end
5756     })
5757 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.

```

5758 local parsers = self.parsers

```

3.1.5.1 Top-Level Helper Functions Define `reader->normalize_tag` as a function that normalizes a markdown reference tag by lowercasing it, and by collapsing any adjacent whitespace characters.

```

5759 function self.normalize_tag(tag)
5760     tag = util.ropetostring(tag)
5761     tag = tag:gsub("[\n\r\t]+", " ")
5762     tag = tag:gsub("^ ", ""):gsub(" $", "")
5763     tag = uni_case.casefold(tag, true, false)
5764     return tag
5765 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.

```

5766 local function iterlines(s, f)
5767     local rope = lpeg.match(Ct((parsers.line / f)^1), s)
5768     return util.ropetostring(rope)
5769 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.

```

5770 if options.preserveTabs then
5771     self.expandtabs = function(s) return s end
5772 else
5773     self.expandtabs = function(s)
5774         if s:find("\t") then
5775             return iterlines(s, util.expand_tabs_in_line)
5776         else
5777             return s
5778         end
5779     end
5780 end

```

3.1.5.2 High-Level Parser Functions Create a `reader->parser_functions` hash table that stores high-level parser functions. Define `reader->create_parser` as a function that will create a high-level parser function `reader->parser_functions.name`, that matches input using grammar `grammar`. If `toplevel` is true, the input is expected to come straight from the user, not from a recursive call, and will be preprocessed.

```
5781 self.parser_functions = {}
5782 self.create_parser = function(name, grammar, toplevel)
5783     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.

```
5784     if toplevel and options.stripIndent then
5785         local min_prefix_length, min_prefix = nil, ''
5786         str = iterlines(str, function(line)
5787             if lpeg.match(parsers.nonemptyline, line) == nil then
5788                 return line
5789             end
5790             line = util.expand_tabs_in_line(line)
5791             local prefix = lpeg.match(C(parsers.optionalspace), line)
5792             local prefix_length = #prefix
5793             local is_shorter = min_prefix_length == nil
5794             is_shorter = is_shorter or prefix_length < min_prefix_length
5795             if is_shorter then
5796                 min_prefix_length, min_prefix = prefix_length, prefix
5797             end
5798             return line
5799         end)
5800         str = str:gsub('^' .. min_prefix, '')
5801     end
```

If the parser is top-level and the `texComments` or `hybrid` Lua options are enabled, we will strip all plain T_EX comments from the input string `str` together with the trailing newline characters.

```
5802     if toplevel and (options.texComments or options.hybrid) then
5803         str = lpeg.match(Ct(parsers.commented_line^1), str)
5804         str = util.rope_to_string(str)
5805     end
5806     local res = lpeg.match(grammar(), str)
5807     if res == nil then
5808         error(format("%s failed on:\n%s", name, str:sub(1,20)))
5809     else
5810         return res
5811     end
```

```

5812     end
5813 end
5814
5815 self.create_parser("parse_blocks",
5816                   function()
5817                     return parsers.blocks
5818                   end, true)
5819
5820 self.create_parser("parse_blocks_nested",
5821                   function()
5822                     return parsers.blocks_nested
5823                   end, false)
5824
5825 self.create_parser("parse_inlines",
5826                   function()
5827                     return parsers.inlines
5828                   end, false)
5829
5830 self.create_parser("parse_inlines_no_link",
5831                   function()
5832                     return parsers.inlines_no_link
5833                   end, false)
5834
5835 self.create_parser("parse_inlines_no_inline_note",
5836                   function()
5837                     return parsers.inlines_no_inline_note
5838                   end, false)
5839
5840 self.create_parser("parse_inlines_no_html",
5841                   function()
5842                     return parsers.inlines_no_html
5843                   end, false)
5844
5845 self.create_parser("parse_inlines_nbsp",
5846                   function()
5847                     return parsers.inlines_nbsp
5848                   end, false)

```

3.1.5.3 Parsers Used for Markdown Lists (local)

```

5849 if options.hashEnumerators then
5850   parsers.dig = parsers.digit + parsers.hash
5851 else
5852   parsers.dig = parsers.digit
5853 end
5854
5855 parsers.enumerator = C(parsers.dig^3 * parsers.period) * #parsers.spacing

```

```

5856         + C(parsers.dig^2 * parsers.period) * #parsers.spacing
5857           * (parsers.tab + parsers.space^1)
5858         + C(parsers.dig * parsers.period) * #parsers.spacing
5859           * (parsers.tab + parsers.space^-2)
5860         + parsers.space * C(parsers.dig^2 * parsers.period)
5861           * #parsers.spacing
5862         + parsers.space * C(parsers.dig * parsers.period)
5863           * #parsers.spacing
5864           * (parsers.tab + parsers.space^-1)
5865         + parsers.space * parsers.space * C(parsers.dig^1
5866           * parsers.period) * #parsers.spacing

```

3.1.5.4 Parsers Used for Blockquotes (local)

```

5867 -- strip off leading > and indents, and run through blocks
5868 parsers.blockquote_body = ((parsers.leader * parsers.more * parsers.space^-
5869 1)/""
5870                               * parsers.linechar^0 * parsers.newline)^1
5871                               * (-V("BlockquoteExceptions") * parsers.linechar^1
5872                               * parsers.newline)^0
5873 if not options.breakableBlockquotes then
5874     parsers.blockquote_body = parsers.blockquote_body
5875                               * (parsers.blankline^0 / "")
5876 end

```

3.1.5.5 Helpers for Links and References (local)

```

5877 -- List of references defined in the document
5878 local references
5879
5880 -- add a reference to the list
5881 local function register_link(tag,url,title)
5882     references[self.normalize_tag(tag)] = { url = url, title = title }
5883     return ""
5884 end
5885
5886 -- lookup link reference and return either
5887 -- the link or nil and fallback text.
5888 local function lookup_reference(label,sps,tag)
5889     local tagpart
5890     if not tag then
5891         tag = label
5892         tagpart = ""
5893     elseif tag == "" then
5894         tag = label
5895         tagpart = "[]"
5896     else

```

```

5897         tagpart = {"[",
5898             self.parser_functions.parse_inlines(tag),
5899             "]" }
5900     end
5901     if sps then
5902         tagpart = {sps, tagpart}
5903     end
5904     local r = references[self.normalize_tag(tag)]
5905     if r then
5906         return r
5907     else
5908         return nil, {"[",
5909             self.parser_functions.parse_inlines(label),
5910             "]", tagpart}
5911     end
5912 end
5913
5914 -- lookup link reference and return a link, if the reference is found,
5915 -- or a bracketed label otherwise.
5916 local function indirect_link(label,sps,tag)
5917     return writer.defer_call(function()
5918         local r,fallback = lookup_reference(label,sps,tag)
5919         if r then
5920             return writer.link(
5921                 self.parser_functions.parse_inlines_no_link(label),
5922                 r.url, r.title)
5923         else
5924             return fallback
5925         end
5926     end)
5927 end
5928
5929 -- lookup image reference and return an image, if the reference is found,
5930 -- or a bracketed label otherwise.
5931 local function indirect_image(label,sps,tag)
5932     return writer.defer_call(function()
5933         local r,fallback = lookup_reference(label,sps,tag)
5934         if r then
5935             return writer.image(writer.string(label), r.url, r.title)
5936         else
5937             return {"!", fallback}
5938         end
5939     end)
5940 end

```

3.1.5.6 Inline Elements (local)

```

5941 parsers.Str      = (parsers.normalchar * (parsers.normalchar + parsers.at)^0)
5942                  / writer.string
5943
5944 parsers.Symbol    = (V("SpecialChar") - parsers.tightblocksep)
5945                  / writer.string
5946
5947 parsers.Ellipsis  = P("...") / writer.ellipsis
5948
5949 parsers.Smart     = parsers.Ellipsis
5950
5951 parsers.Code      = parsers.inticks / writer.code
5952
5953 if options.blankBeforeBlockquote then
5954   parsers.bqstart = parsers.fail
5955 else
5956   parsers.bqstart = parsers.more
5957 end
5958
5959 if options.blankBeforeHeading then
5960   parsers.headerstart = parsers.fail
5961 else
5962   parsers.headerstart = parsers.hash
5963                       + (parsers.line * (parsers.equal^1 + parsers.dash^1)
5964                       * parsers.optionalspace * parsers.newline)
5965 end
5966
5967 parsers.EndlineExceptions
5968         = parsers.blankline -- paragraph break
5969         + parsers.tightblocksep -- nested list
5970         + parsers.eof        -- end of document
5971         + parsers.bqstart
5972         + parsers.headerstart
5973
5974 parsers.Endline = parsers.newline
5975                 * -V("EndlineExceptions")
5976                 * parsers.spacechar^0
5977                 / (options.hardLineBreaks and writer.hard_line_break
5978                   or writer.space)
5979
5980 parsers.OptionalIndent
5981         = parsers.spacechar^1 / writer.space
5982
5983 parsers.Space = parsers.spacechar^2 * parsers.Endline / writer.hard_line_break
5984               + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
5985               + parsers.spacechar^1 * parsers.Endline
5986                   * parsers.optionalspace
5987               / (options.hardLineBreaks

```

```

5988             and writer.hard_line_break
5989             or writer.space)
5990         + parsers.spacechar^1 * parsers.optionalspace
5991           / writer.space
5992
5993 parsers.NonbreakingEndline
5994     = parsers.newline
5995     * -V("EndlineExceptions")
5996     * parsers.spacechar^0
5997     / (options.hardLineBreaks and writer.hard_line_break
5998       or writer.nbsp)
5999
6000 parsers.NonbreakingSpace
6001     = parsers.spacechar^2 * parsers.Endline / writer.hard_line_break
6002     + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
6003     + parsers.spacechar^1 * parsers.Endline
6004       * parsers.optionalspace
6005       / (options.hardLineBreaks
6006         and writer.hard_line_break
6007         or writer.nbsp)
6008     + parsers.spacechar^1 * parsers.optionalspace
6009       / writer.nbsp
6010
6011 if options.underscores then
6012     parsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,
6013                                     parsers.doubleasterisks)
6014                     + parsers.between(parsers.Inline, parsers.doubleunderscores,
6015                                     parsers.doubleunderscores)
6016                     ) / writer.strong
6017
6018     parsers.Emph   = ( parsers.between(parsers.Inline, parsers.asterisk,
6019                                     parsers.asterisk)
6020                     + parsers.between(parsers.Inline, parsers.underscore,
6021                                     parsers.underscore)
6022                     ) / writer.emphasis
6023 else
6024     parsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,
6025                                     parsers.doubleasterisks)
6026                     ) / writer.strong
6027
6028     parsers.Emph   = ( parsers.between(parsers.Inline, parsers.asterisk,
6029                                     parsers.asterisk)
6030                     ) / writer.emphasis
6031 end
6032
6033 parsers.AutoLinkUrl = parsers.less
6034                   * C(parsers.alphanumeric^1 * P("://") * parsers.urlchar^1)

```



```

6035         * parsers.more
6036         / function(url)
6037             return writer.link(writer.escape(url), url)
6038         end
6039
6040 parsers.AutoLinkEmail = parsers.less
6041     * C((parsers.alphanumeric + S("-._+"))^1
6042     * P("@") * parsers.urlchar^1)
6043     * parsers.more
6044     / function(email)
6045         return writer.link(writer.escape(email),
6046                             "mailto:".email)
6047     end
6048
6049 parsers.AutoLinkRelativeReference
6050     = parsers.less
6051     * C(parsers.urlchar^1)
6052     * parsers.more
6053     / function(url)
6054         return writer.link(writer.escape(url), url)
6055     end
6056
6057 parsers.DirectLink     = (parsers.tag / self.parser_functions.parse_inlines_no_link)
6058     * parsers.spnl
6059     * parsers.lparent
6060     * (parsers.url + C("")) -- link can be empty [foo]()
6061     * parsers.optionaltitle
6062     * parsers.rparent
6063     / writer.link
6064
6065 parsers.IndirectLink  = parsers.tag * (C(parsers.spnl) * parsers.tag)^-
6066     / indirect_link
6067
6068 -- parse a link or image (direct or indirect)
6069 parsers.Link          = parsers.DirectLink + parsers.IndirectLink
6070
6071 parsers.DirectImage   = parsers.exclamation
6072     * (parsers.tag / self.parser_functions.parse_inlines)
6073     * parsers.spnl
6074     * parsers.lparent
6075     * (parsers.url + C("")) -- link can be empty [foo]()
6076     * parsers.optionaltitle
6077     * parsers.rparent
6078     / writer.image
6079
6080 parsers.IndirectImage = parsers.exclamation * parsers.tag

```

```

6081             * (C(parsers.spnl) * parsers.tag)^-1 / indirect_image
6082
6083 parsers.Image      = parsers.DirectImage + parsers.IndirectImage
6084
6085 -- avoid parsing long strings of * or _ as emph/strong
6086 parsers.UlOrStarLine = parsers.asterisk^4 + parsers.underscore^4
6087                    / writer.string
6088
6089 parsers.EscapedChar = parsers.backslash * C(parsers.escapable) / writer.string
6090
6091 parsers.InlineHtml  = parsers.emptyelt_any / writer.inline_html_tag
6092                    + (parsers.htmlcomment / self.parser_functions.parse_inlines
6093                    / writer.inline_html_comment
6094                    + parsers.htmlinstruction
6095                    + parsers.openelt_any / writer.inline_html_tag
6096                    + parsers.closeelt_any / writer.inline_html_tag
6097
6098 parsers.HtmlEntity  = parsers.hexentity / entities.hex_entity / writer.string
6099                    + parsers.decentity / entities.dec_entity / writer.string
6100                    + parsers.tagentity / entities.char_entity / writer.string

```

3.1.5.7 Block Elements (local)

```

6101 parsers.DisplayHtml = (parsers.htmlcomment / self.parser_functions.parse_blocks_nested
6102                    / writer.block_html_comment
6103                    + parsers.emptyelt_block / writer.block_html_element
6104                    + parsers.openelt_exact("hr") / writer.block_html_element
6105                    + parsers.in_matched_block_tags / writer.block_html_element
6106                    + parsers.htmlinstruction
6107
6108 parsers.Verbatim    = Cs( (parsers.blanklines
6109                    * ((parsers.indentedline - parsers.blankline))^1)^1
6110                    ) / self.expandtabs / writer.verbatim
6111
6112 parsers.BlockquoteExceptions = parsers.leader * parsers.more
6113                    + parsers.blankline
6114
6115 parsers.Blockquote   = Cs(parsers.blockquote_body^1)
6116                    / self.parser_functions.parse_blocks_nested
6117                    / writer.blockquote
6118
6119 parsers.ThematicBreak = ( parsers.lineof(parsers.asterisk)
6120                    + parsers.lineof(parsers.dash)
6121                    + parsers.lineof(parsers.underscore)
6122                    ) / writer.thematic_break
6123
6124 parsers.Reference    = parsers.define_reference_parser / register_link

```

```

6125
6126 parsers.Paragraph = parsers.nonindentspace * Ct(parsers.Inline~1)
6127 * ( parsers.newline
6128 * ( parsers.blankline~1
6129 * #V("EndlineExceptions")
6130 )
6131 + parsers.eof)
6132 / writer.paragraph
6133
6134 parsers.Plain = parsers.nonindentspace * Ct(parsers.Inline~1)
6135 / writer.plain

```

3.1.5.8 Lists (local)

```

6136 parsers.starter = parsers.bullet + parsers.enumerator
6137
6138 if options.taskLists then
6139   parsers.tickbox = ( parsers.ticked_box
6140 * parsers.halfticked_box
6141 + parsers.unticked_box
6142 ) / writer.tickbox
6143 else
6144   parsers.tickbox = parsers.fail
6145 end
6146
6147 -- we use \001 as a separator between a tight list item and a
6148 -- nested list under it.
6149 parsers.NestedList = Cs((parsers.optionallyindentedline
6150 - parsers.starter)^1)
6151 / function(a) return "\001"..a end
6152
6153 parsers.ListBlockLine = parsers.optionallyindentedline
6154 - parsers.blankline - (parsers.indent~1
6155 * parsers.starter)
6156
6157 parsers.ListBlock = parsers.line * parsers.ListBlockLine~0
6158
6159 parsers.ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
6160 * parsers.ListBlock
6161
6162 parsers.TightListItem = function(starter)
6163   return -parsers.ThematicBreak
6164 * (Cs(starter / "" * parsers.tickbox~-1 * parsers.ListBlock * parsers.Ne
6165 1)
6166 / self.parser_functions.parse_blocks_nested)
6167 * -(parsers.blanklines * parsers.indent)

```

```

6167 end
6168
6169 parsers.LooseListItem = function(starter)
6170     return -parsers.ThematicBreak
6171         * Cs( starter / "" * parsers.checkbox^-1 * parsers.ListBlock * Cc("\n")
6172             * (parsers.NestedList + parsers.ListContinuationBlock^0)
6173             * (parsers.blanklines / "\n\n")
6174             ) / self.parser_functions.parse_blocks_nested
6175 end
6176
6177 parsers.BulletList = ( Ct(parsers.TightListItem(parsers.bullet)^1) * Cc(true)
6178                       * parsers.skipblanklines * -parsers.bullet
6179                       + Ct(parsers.LooseListItem(parsers.bullet)^1) * Cc(false)
6180                       * parsers.skipblanklines )
6181                       / writer.bulletlist
6182
6183 local function ordered_list(items,tight,startnum)
6184     if options.startNumber then
6185         startnum = tonumber(startnum) or 1 -- fallback for '#'
6186         if startnum ~= nil then
6187             startnum = math.floor(startnum)
6188         end
6189     else
6190         startnum = nil
6191     end
6192     return writer.orderedlist(items,tight,startnum)
6193 end
6194
6195 parsers.OrderedList = Cg(parsers.enumerator, "listtype") *
6196     ( Ct(parsers.TightListItem(Cb("listtype")))
6197       * parsers.TightListItem(parsers.enumerator)^0)
6198     * Cc(true) * parsers.skipblanklines * -parsers.enumerator
6199     + Ct(parsers.LooseListItem(Cb("listtype")))
6200       * parsers.LooseListItem(parsers.enumerator)^0)
6201     * Cc(false) * parsers.skipblanklines
6202     ) * Cb("listtype") / ordered_list

```

3.1.5.9 Blank (local)

```

6203 parsers.Blank           = parsers.blankline / ""
6204                       + parsers.Reference
6205                       + (parsers.tightblocksep / "\n")

```

3.1.5.10 Headings (local)

```

6206 -- parse atx header
6207 parsers.AtxHeading = Cg(parsers.heading_start, "level")
6208                   * parsers.optionalspace

```

```

6209         * (C(parsers.line)
6210           / strip_atx_end
6211           / self.parser_functions.parse_inlines)
6212         * Cb("level")
6213         / writer.heading
6214
6215     parsers.SettextHeading = #(parsers.line * S("--"))
6216         * Ct(parsers.linechar~1
6217           / self.parser_functions.parse_inlines)
6218         * parsers.newline
6219         * parsers.heading_level
6220         * parsers.optionalspace
6221         * parsers.newline
6222         / writer.heading
6223
6224     parsers.Heading = parsers.AtxHeading + parsers.SettextHeading

```

3.1.5.11 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.

```

6225     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.

```

6226     local walkable_syntax = (function(global_walkable_syntax)
6227       local local_walkable_syntax = {}
6228       for lhs, rule in pairs(global_walkable_syntax) do
6229         local_walkable_syntax[lhs] = util.table_copy(rule)
6230       end
6231       return local_walkable_syntax
6232     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.

```

6233     local current_extension_name = nil
6234     self.insert_pattern = function(selector, pattern, pattern_name)
6235       assert(pattern_name == nil or type(pattern_name) == "string")
6236       local _, _, lhs, pos, rhs = selector:find("^([a+])%s+([%a%s]+%a+)%s+([a+])$")
6237       assert(lhs ~= nil,
6238         [[Expected selector in form "LHS (before|after|instead of) RHS", not "]]
6239         .. selector .. [[]])

```

```

6240     assert(walkable_syntax[lhs] ~= nil,
6241            [[Rule ]] .. lhs .. [[ -> ... does not exist in markdown grammar]])
6242     assert(pos == "before" or pos == "after" or pos == "instead of",
6243            [[Expected positional specifier "before", "after", or "instead of", not "]]
6244            .. pos .. [[]])
6245     local rule = walkable_syntax[lhs]
6246     local index = nil
6247     for current_index, current_rhs in ipairs(rule) do
6248         if type(current_rhs) == "string" and current_rhs == rhs then
6249             index = current_index
6250             if pos == "after" then
6251                 index = index + 1
6252             end
6253             break
6254         end
6255     end
6256     assert(index ~= nil,
6257            [[Rule ]] .. lhs .. [[ -> ]] .. rhs
6258            .. [[ does not exist in markdown grammar]])
6259     local accountable_pattern
6260     if current_extension_name then
6261         accountable_pattern = { pattern, current_extension_name, pattern_name }
6262     else
6263         assert(type(pattern) == "string",
6264                [[reader->insert_pattern() was called outside an extension with ]]
6265                .. [[a PEG pattern instead of a rule name]])
6266         accountable_pattern = pattern
6267     end
6268     if pos == "instead of" then
6269         rule[index] = accountable_pattern
6270     else
6271         table.insert(rule, index, accountable_pattern)
6272     end
6273 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.

```

6274     local syntax =
6275         { "Blocks",
6276
6277           Blocks
6278             = V("InitializeState")
6279             * ( V("ExpectedJekyllData")
6280                * (V("Blank")^0 / writer.interblocksep))^1
6281
6282             * V("Blank")^0
6283             * V("Block")^-1
6284             * ( V("Blank")^0 / writer.interblocksep

```

```

6283             * V("Block"))^0
6284             * V("Blank")^0 * parsers.eof,
6285
6286     ExpectedJekyllData = parsers.fail,
6287
6288     Blank = parsers.Blank,
6289
6290     Blockquote = parsers.Blockquote,
6291     Verbatim = parsers.Verbatim,
6292     ThematicBreak = parsers.ThematicBreak,
6293     BulletList = parsers.BulletList,
6294     OrderedList = parsers.OrderedList,
6295     Heading = parsers.Heading,
6296     DisplayHtml = parsers.DisplayHtml,
6297     Paragraph = parsers.Paragraph,
6298     Plain = parsers.Plain,
6299
6300     EndlineExceptions = parsers.EndlineExceptions,
6301     BlockquoteExceptions = parsers.BlockquoteExceptions,
6302
6303     Str = parsers.Str,
6304     Space = parsers.Space,
6305     OptionalIndent = parsers.OptionalIndent,
6306     Endline = parsers.Endline,
6307     U1OrStarLine = parsers.U1OrStarLine,
6308     Strong = parsers.Strong,
6309     Emph = parsers.Emph,
6310     Link = parsers.Link,
6311     Image = parsers.Image,
6312     Code = parsers.Code,
6313     AutoLinkUrl = parsers.AutoLinkUrl,
6314     AutoLinkEmail = parsers.AutoLinkEmail,
6315     AutoLinkRelativeReference
6316             = parsers.AutoLinkRelativeReference,
6317     InlineHtml = parsers.InlineHtml,
6318     HtmlEntity = parsers.HtmlEntity,
6319     EscapedChar = parsers.EscapedChar,
6320     Smart = parsers.Smart,
6321     Symbol = parsers.Symbol,
6322     SpecialChar = parsers.fail,
6323     InitializeState = parsers.succeed,
6324 }

```

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].

```
6325 self.update_rule = function(rule_name, get_pattern)
6326   assert(current_extension_name ~= nil)
6327   assert(syntax[rule_name] ~= nil,
6328     [[Rule ]] .. rule_name .. [[ -> ... does not exist in markdown grammar]])
6329   local previous_pattern
6330   local extension_name
6331   if walkable_syntax[rule_name] then
6332     local previous_accountable_pattern = walkable_syntax[rule_name][1]
6333     previous_pattern = previous_accountable_pattern[1]
6334     extension_name = previous_accountable_pattern[2] .. ", " .. current_extension_name
6335   else
6336     previous_pattern = nil
6337     extension_name = current_extension_name
6338   end
6339   local pattern = get_pattern(previous_pattern)
6340   local accountable_pattern = { pattern, extension_name, rule_name }
6341   walkable_syntax[rule_name] = { accountable_pattern }
6342 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.

```
6343 local special_characters = {}
6344 self.add_special_character = function(c)
6345   table.insert(special_characters, c)
6346   syntax.SpecialChar = S(table.concat(special_characters, ""))
6347 end
6348
6349 self.add_special_character("*")
6350 self.add_special_character("[")
6351 self.add_special_character("]")
6352 self.add_special_character("<")
6353 self.add_special_character("!")
6354 self.add_special_character("\\")
```

Add method `reader->initialize_named_group` that defines named groups with a default capture value.

```
6355 self.initialize_named_group = function(name, value)
6356   syntax.InitializeState = syntax.InitializeState
6357     * Cg(Ct("") / value, name)
6358 end
```

Apply syntax extensions.

```
6359 for _, extension in ipairs(extensions) do
6360   current_extension_name = extension.name
6361   extension.extend_writer(writer)
```



```

6362     extension.extend_reader(self)
6363     end
6364     current_extension_name = nil

```

If the `debugExtensions` option is enabled, serialize `walkable_syntax` to a JSON for debugging purposes.

```

6365     if options.debugExtensions then
6366         local sorted_lhs = {}
6367         for lhs, _ in pairs(walkable_syntax) do
6368             table.insert(sorted_lhs, lhs)
6369         end
6370         table.sort(sorted_lhs)
6371
6372         local output_lines = {"{"}
6373         for lhs_index, lhs in ipairs(sorted_lhs) do
6374             local encoded_lhs = util.encode_json_string(lhs)
6375             table.insert(output_lines, [{" "] .. encoded_lhs .. [{" ": [{" ]}]})
6376             local rule = walkable_syntax[lhs]
6377             for rhs_index, rhs in ipairs(rule) do
6378                 local human_readable_rhs
6379                 if type(rhs) == "string" then
6380                     human_readable_rhs = rhs
6381                 else
6382                     local pattern_name
6383                     if rhs[3] then
6384                         pattern_name = rhs[3]
6385                     else
6386                         pattern_name = "Anonymous Pattern"
6387                     end
6388                     local extension_name = rhs[2]
6389                     human_readable_rhs = pattern_name .. [{" (["] .. extension_name .. [{" (["]})]
6390                 end
6391                 local encoded_rhs = util.encode_json_string(human_readable_rhs)
6392                 local output_line = [{" " "] .. encoded_rhs
6393                 if rhs_index < #rule then
6394                     output_line = output_line .. ", "
6395                 end
6396                 table.insert(output_lines, output_line)
6397             end
6398             local output_line = [{" " "]
6399             if lhs_index < #sorted_lhs then
6400                 output_line = output_line .. ", "
6401             end
6402             table.insert(output_lines, output_line)
6403         end
6404         table.insert(output_lines, "}")
6405     end

```

```

6406     local output = table.concat(output_lines, "\n")
6407     local output_filename = options.debugExtensionsFileName
6408     local output_file = assert(io.open(output_filename, "w"),
6409         [[Could not open file ]] .. output_filename .. [{" for writing}])
6410     assert(output_file:write(output))
6411     assert(output_file:close())
6412 end

```

Duplicate the `Inline` rule as `IndentedInline` with the right-hand-side terminal symbol `Space` replaced with `OptionalIndent`.

```

6413     walkable_syntax["IndentedInline"] = util.table_copy(
6414         walkable_syntax["Inline"])
6415     self.insert_pattern(
6416         "IndentedInline instead of Space",
6417         "OptionalIndent")

```

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`.

```

6418     for lhs, rule in pairs(walkable_syntax) do
6419         syntax[lhs] = parsers.fail
6420         for _, rhs in ipairs(rule) do
6421             local pattern

```

Although the interface of the `reader->insert_pattern` method does 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.

```

6422         if type(rhs) == "string" then
6423             pattern = V(rhs)
6424         else
6425             pattern = rhs[1]
6426             if type(pattern) == "string" then
6427                 pattern = V(pattern)
6428             end
6429         end
6430         syntax[lhs] = syntax[lhs] + pattern
6431     end
6432 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.

```

6433     if options.underscores then
6434         self.add_special_character("_")
6435     end
6436
6437     if not options.codeSpans then

```

```

6438     syntax.Code = parsers.fail
6439 else
6440     self.add_special_character("`")
6441 end
6442
6443 if not options.html then
6444     syntax.DisplayHtml = parsers.fail
6445     syntax.InlineHtml = parsers.fail
6446     syntax.HtmlEntity = parsers.fail
6447 else
6448     self.add_special_character("&")
6449 end
6450
6451 if options.preserveTabs then
6452     options.stripIndent = false
6453 end
6454
6455 if not options.smartEllipses then
6456     syntax.Smart = parsers.fail
6457 else
6458     self.add_special_character(".")
6459 end
6460
6461 if not options.relativeReferences then
6462     syntax.AutoLinkRelativeReference = parsers.fail
6463 end
6464
6465 local blocks_nested_t = util.table_copy(syntax)
6466 blocks_nested_t.ExpectedJekyllData = parsers.fail
6467 parsers.blocks_nested = Ct(blocks_nested_t)
6468
6469 parsers.blocks = Ct(syntax)
6470
6471 local inlines_t = util.table_copy(syntax)
6472 inlines_t[1] = "Inlines"
6473 inlines_t.Inlines = V("InitializeState")
6474     * parsers.Inline^0
6475     * ( parsers.spacing^0
6476     * parsers.eof / "")
6477 parsers.inlines = Ct(inlines_t)
6478
6479 local inlines_no_link_t = util.table_copy(inlines_t)
6480 inlines_no_link_t.Link = parsers.fail
6481 parsers.inlines_no_link = Ct(inlines_no_link_t)
6482
6483 local inlines_no_inline_note_t = util.table_copy(inlines_t)
6484 inlines_no_inline_note_t.InlineNote = parsers.fail

```

```

6485     parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
6486
6487     local inlines_no_html_t = util.table_copy(inlines_t)
6488     inlines_no_html_t.DisplayHtml = parsers.fail
6489     inlines_no_html_t.InlineHtml = parsers.fail
6490     inlines_no_html_t.HtmlEntity = parsers.fail
6491     parsers.inlines_no_html = Ct(inlines_no_html_t)
6492
6493     local inlines_nbsp_t = util.table_copy(inlines_t)
6494     inlines_nbsp_t.Endline = parsers.NonbreakingEndline
6495     inlines_nbsp_t.Space = parsers.NonbreakingSpace
6496     parsers.inlines_nbsp = Ct(inlines_nbsp_t)

```

Return a function that converts markdown string `input` into a plain \TeX output and returns it..

```

6497     return function(input)

```

Since the Lua converter expects UNIX line endings, normalize the input. Also add a line ending at the end of the file in case the input file has none.

```

6498     input = input:gsub("\r\n?", "\n")
6499     if input:sub(-1) ~= "\n" then
6500         input = input .. "\n"
6501     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.

```

6502     references = {}
6503     local opt_string = {}
6504     for k, _ in pairs(defaultOptions) do
6505         local v = options[k]
6506         if type(v) == "table" then
6507             for _, i in ipairs(v) do
6508                 opt_string[#opt_string+1] = k .. "=" .. tostring(i)
6509             end
6510         elseif k ~= "cacheDir" then
6511             opt_string[#opt_string+1] = k .. "=" .. tostring(v)
6512         end
6513     end
6514     table.sort(opt_string)
6515     local salt = table.concat(opt_string, ",") .. "," .. metadata.version
6516     local output

```

If we cache markdown documents, produce the cache file and transform its filename to plain \TeX output via the `writer->pack` method.

```

6517     local function convert(input)
6518         local document = self.parser_functions.parse_blocks(input)
6519         return util.rope_to_string(writer.document(document))

```

```

6520     end
6521     if options.eagerCache or options.finalizeCache then
6522         local name = util.cache(options.cacheDir, input, salt, convert,
6523                                ".md" .. writer.suffix)
6524         output = writer.pack(name)

```

Otherwise, return the result of the conversion directly.

```

6525     else
6526         output = convert(input)
6527     end

```

If the `finalizeCache` option is enabled, populate the frozen cache in the file `frozenCacheFileName` with an entry for markdown document number `frozenCacheCounter`.

```

6528     if options.finalizeCache then
6529         local file, mode
6530         if options.frozenCacheCounter > 0 then
6531             mode = "a"
6532         else
6533             mode = "w"
6534         end
6535         file = assert(io.open(options.frozenCacheFileName, mode),
6536                        [[Could not open file ]] .. options.frozenCacheFileName
6537                        .. [{" for writing"}])
6538         assert(file:write([[\\expandafter\\global\\expandafter\\def\\csname ]]
6539                            .. [{"markdownFrozenCache"}] .. options.frozenCacheCounter
6540                            .. [{"\\endcsname{}}] .. output .. [{"}]"] .. "\\n"))
6541         assert(file:close())
6542     end
6543     return output
6544 end
6545 end
6546 return self
6547 end

```

3.1.6 Built-In Syntax Extensions

Create `extensions` hash table that contains built-in syntax extensions. Syntax extensions are functions that produce objects with two methods: `extend_writer` and `extend_reader`. The `extend_writer` object takes a `writer` object as the only parameter and mutates it. Similarly, `extend_reader` takes a `reader` object as the only parameter and mutates it.

```

6548 M.extensions = {}

```

3.1.6.1 Bracketed Spans The `extensions.bracketed_spans` function implements the Pandoc bracketed spans syntax extension.

```

6549 M.extensions.bracketed_spans = function()
6550   return {
6551     name = "built-in bracketed_spans syntax extension",
6552     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.

```

6553     function self.span(s, attr)
6554       return {"\\markdownRendererBracketedSpanAttributeContextBegin",
6555             self.attributes(attr),
6556             s,
6557             "\\markdownRendererBracketedSpanAttributeContextEnd{}}"}
6558     end
6559   end, extend_reader = function(self)
6560     local parsers = self.parsers
6561     local writer = self.writer
6562
6563     local Span = parsers.between(parsers.Inline,
6564                                parsers.lbracket,
6565                                parsers.rbracket)
6566                               * Ct(parsers.attributes)
6567                               / writer.span
6568
6569     self.insert_pattern("Inline after Emph",
6570                       Span, "Span")
6571   end
6572 }
6573 end

```

3.1.6.2 Citations The `extensions.citations` function implements the Pandoc citation syntax extension. When the `citation_nbsps` parameter is enabled, the syntax extension will replace regular spaces with non-breaking spaces inside the prenotes and postnotes of citations.

```

6574 M.extensions.citations = function(citation_nbsps)
6575   return {
6576     name = "built-in citations syntax extension",
6577     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.

```

6578     function self.citations(text_cites, cites)
6579         local buffer = {"\markdownRenderer", text_cites and "TextCite" or "Cite",
6580             "{", #cites, "}"}
6581         for _,cite in ipairs(cites) do
6582             buffer[#buffer+1] = {cite.suppress_author and "-" or "+", "{",
6583                 cite.prenote or "", "}{" , cite.postnote or "", "}{" , cite.name, "}"}
6584         end
6585         return buffer
6586     end
6587 end, extend_reader = function(self)
6588     local parsers = self.parsers
6589     local writer = self.writer
6590
6591     local citation_chars
6592         = parsers.alphanumeric
6593         + S("#$%&-+<>~/_")
6594
6595     local citation_name
6596         = Cs(parsers.dash^-1) * parsers.at
6597         * Cs(citation_chars
6598             * (((citation_chars + parsers.internal_punctuation
6599                 - parsers.comma - parsers.semicolon)
6600                 * -#((parsers.internal_punctuation - parsers.comma
6601                     - parsers.semicolon)^0
6602                     * -(citation_chars + parsers.internal_punctuation
6603                         - parsers.comma - parsers.semicolon)))^0
6604                 * citation_chars)^-1)
6605
6606     local citation_body_prenote
6607         = Cs((parsers.alphanumeric^1
6608             + parsers.bracketed
6609             + parsers.inticks
6610             + (parsers.anyescaped
6611                 - (parsers.rbracket + parsers.blankline^2))
6612             - (parsers.spnl * parsers.dash^-1 * parsers.at))^0)
6613
6614     local citation_body_postnote
6615         = Cs((parsers.alphanumeric^1
6616             + parsers.bracketed
6617             + parsers.inticks

```

```

6618         + (parsers.anyescaped
6619         - (parsers.rbracket + parsers.semicolon
6620         + parsers.blankline^2))
6621         - (parsers.spnl * parsers.rbracket))^0)
6622
6623     local citation_body_chunk
6624         = citation_body_prenote
6625         * parsers.spnl * citation_name
6626         * (parsers.internal_punctuation - parsers.semicolon)^-
1
6627         * parsers.spnl * citation_body_postnote
6628
6629     local citation_body
6630         = citation_body_chunk
6631         * (parsers.semicolon * parsers.spnl
6632         * citation_body_chunk)^0
6633
6634     local citation_headless_body_postnote
6635         = Cs((parsers.alphanumeric^1
6636         + parsers.bracketed
6637         + parsers.inticks
6638         + (parsers.anyescaped
6639         - (parsers.rbracket + parsers.at
6640         + parsers.semicolon + parsers.blankline^2))
6641         - (parsers.spnl * parsers.rbracket))^0)
6642
6643     local citation_headless_body
6644         = citation_headless_body_postnote
6645         * (parsers.sp * parsers.semicolon * parsers.spnl
6646         * citation_body_chunk)^0
6647
6648     local citations
6649         = function(text_cites, raw_cites)
6650         local function normalize(str)
6651             if str == "" then
6652                 str = nil
6653             else
6654                 str = (citation_nbsps and
6655                 self.parser_functions.parse_inlines_nbsp or
6656                 self.parser_functions.parse_inlines)(str)
6657             end
6658             return str
6659         end
6660
6661         local cites = {}
6662         for i = 1,#raw_cites,4 do
6663             cites[#cites+1] = {

```



```

6664         prenote = normalize(raw_cites[i]),
6665         suppress_author = raw_cites[i+1] == "-",
6666         name = writer.identifier(raw_cites[i+2]),
6667         postnote = normalize(raw_cites[i+3]),
6668     }
6669     end
6670     return writer.citations(text_cites, cites)
6671 end
6672
6673 local TextCitations
6674     = Ct((parsers.spnl
6675     * Cc("")
6676     * citation_name
6677     * ((parsers.spnl
6678     * parsers.lbracket
6679     * citation_headless_body
6680     * parsers.rbracket) + Cc("")))^1)
6681 / function(raw_cites)
6682     return citations(true, raw_cites)
6683 end
6684
6685 local ParenthesizedCitations
6686     = Ct((parsers.spnl
6687     * parsers.lbracket
6688     * citation_body
6689     * parsers.rbracket)^1)
6690 / function(raw_cites)
6691     return citations(false, raw_cites)
6692 end
6693
6694 local Citations = TextCitations + ParenthesizedCitations
6695
6696 self.insert_pattern("Inline after Emph",
6697     Citations, "Citations")
6698
6699 self.add_special_character("@")
6700 self.add_special_character("-")
6701 end
6702 }
6703 end

```

3.1.6.3 Content Blocks The `extensions.content_blocks` function implements the iA,Writer content blocks syntax extension. The `language_map` parameter specifies the filename of the JSON file that maps filename extensions to programming language names.

```

6704 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.

```

6705 local languages_json = (function()
6706     local base, prev, curr
6707     for _, pathname in ipairs{util.lookup_files(language_map, { all=true })} do
6708         local file = io.open(pathname, "r")
6709         if not file then goto continue end
6710         local input = assert(file:read("*a"))
6711         assert(file:close())
6712         local json = input:gsub('[^\n]-:', '[%1]=')
6713         curr = load("_ENV = {}; return "..json")()
6714         if type(curr) == "table" then
6715             if base == nil then
6716                 base = curr
6717             else
6718                 setmetatable(prev, { __index = curr })
6719             end
6720             prev = curr
6721         end
6722         ::continue::
6723     end
6724     return base or {}
6725 end)()
6726
6727 return {
6728     name = "built-in content_blocks syntax extension",
6729     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.

```

6730     function self.contentblock(src,suf,type,tit)
6731         if not self.is_writing then return "" end
6732         src = src..".."..suf
6733         suf = suf:lower()
6734         if type == "onlineimage" then
6735             return {"\\markdownRendererContentBlockOnlineImage{"..suf.."}",
6736                 {"",self.string(src),"}",
6737                 {"",self.uri(src),"}",
6738                 {"",self.string(tit or ""),"}"}
6739         elseif languages_json[suf] then
6740             return {"\\markdownRendererContentBlockCode{"..suf.."}",
6741                 {"",self.string(languages_json[suf]),"}",
6742                 {"",self.string(src),"}",

```

```

6743             {"",self.uri(src),""},
6744             {"",self.string(tit or ""),""}
6745     else
6746         return {"\\markdownRendererContentBlock{"",suf,""},
6747             {"",self.string(src),""},
6748             {"",self.uri(src),""},
6749             {"",self.string(tit or ""),""}
6750     end
6751 end
6752 end, extend_reader = function(self)
6753     local parsers = self.parsers
6754     local writer = self.writer
6755
6756     local contentblock_tail
6757         = parsers.optionaltitle
6758         * (parsers.newline + parsers.eof)
6759
6760     -- case insensitive online image suffix:
6761     local onlineimagesuffix
6762         = (function(...)
6763             local parser = nil
6764             for _, suffix in ipairs({...}) do
6765                 local pattern=nil
6766                 for i=1,#suffix do
6767                     local char=suffix:sub(i,i)
6768                     char = S(char:lower()..char:upper())
6769                     if pattern == nil then
6770                         pattern = char
6771                     else
6772                         pattern = pattern * char
6773                     end
6774                 end
6775                 if parser == nil then
6776                     parser = pattern
6777                 else
6778                     parser = parser + pattern
6779                 end
6780             end
6781             return parser
6782         end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
6783
6784     -- online image url for iA Writer content blocks with mandatory suffix,
6785     -- allowing nested brackets:
6786     local onlineimageurl
6787         = (parsers.less
6788             * Cs((parsers.anyescaped
6789                 - parsers.more

```

```

6790         - #(parsers.period
6791           * onlineimagesuffix
6792           * parsers.more
6793           * contentblock_tail))^0)
6794     * parsers.period
6795     * Cs(onlineimagesuffix)
6796     * parsers.more
6797     + (Cs((parsers.inparens
6798         + (parsers.anyescaped
6799           - parsers.spacing
6800           - parsers.rparent
6801           - #(parsers.period
6802             * onlineimagesuffix
6803             * contentblock_tail))))^0)
6804     * parsers.period
6805     * Cs(onlineimagesuffix))
6806   ) * Cc("onlineimage")
6807
6808   -- filename for iA Writer content blocks with mandatory suffix:
6809   local localfilepath
6810     = parsers.slash
6811     * Cs((parsers.anyescaped
6812         - parsers.tab
6813         - parsers.newline
6814         - #(parsers.period
6815           * parsers.alphanumeric^1
6816           * contentblock_tail))^1)
6817     * parsers.period
6818     * Cs(parsers.alphanumeric^1)
6819     * Cc("localfile")
6820
6821   local ContentBlock
6822     = parsers.leader
6823     * (localfilepath + onlineimageurl)
6824     * contentblock_tail
6825     / writer.contentblock
6826
6827   self.insert_pattern("Block before Blockquote",
6828                       ContentBlock, "ContentBlock")
6829 end
6830 }
6831 end

```

3.1.6.4 Definition Lists The `extensions.definition_lists` function implements the Pandoc definition list syntax extension. If the `tight_lists` parameter is `true`, tight lists will produce special right item renderers.

```

6832 M.extensions.definition_lists = function(tight_lists)
6833   return {
6834     name = "built-in definition_lists syntax extension",
6835     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.

```

6836     local function dlitem(term, defs)
6837       local retVal = {"\\markdownRendererDlItem{",term,"}"}
6838       for _, def in ipairs(defs) do
6839         retVal[#retVal+1] = {"\\markdownRendererDlDefinitionBegin ",def,
6840           "\\markdownRendererDlDefinitionEnd "}
6841       end
6842       retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
6843       return retVal
6844     end
6845
6846     function self.definitionlist(items,tight)
6847       if not self.is_writing then return "" end
6848       local buffer = {}
6849       for _,item in ipairs(items) do
6850         buffer[#buffer + 1] = dlitem(item.term, item.definitions)
6851       end
6852       if tight and tight_lists then
6853         return {"\\markdownRendererDlBeginTight\\n", buffer,
6854           "\\n\\markdownRendererDlEndTight"}
6855       else
6856         return {"\\markdownRendererDlBegin\\n", buffer,
6857           "\\n\\markdownRendererDlEnd"}
6858       end
6859     end
6860     end, extend_reader = function(self)
6861       local parsers = self.parsers
6862       local writer = self.writer
6863
6864       local defstartchar = S("~:")
6865
6866       local defstart = ( defstartchar * #parsers.spacing
6867         * (parsers.tab + parsers.space^-
3)
6868         + parsers.space * defstartchar * #parsers.spacing
6869         * (parsers.tab + parsers.space^-
2)
6870         + parsers.space * parsers.space * defstartchar
6871         * #parsers.spacing

```

```

6872                                     * (parsers.tab + parsers.space^-
1)
6873                                     + parsers.space * parsers.space * parsers.space
6874                                     * defstartchar * #parsers.spacing
6875                                     )
6876
6877     local dlchunk = Cs(parsers.line * (parsers.indentedline - parsers.blankline)^0)
6878
6879     local function definition_list_item(term, defs, _)
6880         return { term = self.parser_functions.parse_inlines(term),
6881                 definitions = defs }
6882     end
6883
6884     local DefinitionListItemLoose
6885         = C(parsers.line) * parsers.skipblanklines
6886         * Ct((defstart
6887             * parsers.indented_blocks(dlchunk)
6888             / self.parser_functions.parse_blocks_nested)^1)
6889         * Cc(false) / definition_list_item
6890
6891     local DefinitionListItemTight
6892         = C(parsers.line)
6893         * Ct((defstart * dlchunk
6894             / self.parser_functions.parse_blocks_nested)^1)
6895         * Cc(true) / definition_list_item
6896
6897     local DefinitionList
6898         = ( Ct(DefinitionListItemLoose^1) * Cc(false)
6899           + Ct(DefinitionListItemTight^1)
6900           * (parsers.skipblanklines
6901             * -DefinitionListItemLoose * Cc(true))
6902           ) / writer.definitionlist
6903
6904     self.insert_pattern("Block after Heading",
6905                       DefinitionList, "DefinitionList")
6906 end
6907 }
6908 end

```

3.1.6.5 Fancy Lists The `extensions.fancy_lists` function implements the Pandoc fancy list syntax extension.

```

6909 M.extensions.fancy_lists = function()
6910     return {
6911         name = "built-in fancy_lists syntax extension",
6912         extend_writer = function(self)
6913             local options = self.options

```

6914

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.

```
6915     function self.fancylist(items,tight,startnum,numstyle,numdelim)
6916         if not self.is_writing then return "" end
6917         local buffer = {}
6918         local num = startnum
6919         for _,item in ipairs(items) do
6920             buffer[#buffer + 1] = self.fancyitem(item,num)
6921             if num ~= nil then
6922                 num = num + 1
6923             end
6924         end
6925         local contents = util.intersperse(buffer,"\n")
6926         if tight and options.tightLists then
6927             return {"\markdownRendererFancy01BeginTight{",
6928                 numstyle,"}{",numdelim,"}",contents,
6929                 "\n\markdownRendererFancy01EndTight "}
6930         else
6931             return {"\markdownRendererFancy01Begin{",
6932                 numstyle,"}{",numdelim,"}",contents,
6933                 "\n\markdownRendererFancy01End "}
6934         end
6935     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.

```
6936     function self.fancyitem(s,num)
6937         if num ~= nil then
```

```

6938         return {"\\markdownRendererFancyOListItemWithNumber{" ,num,"} ",s,
6939                 "\\markdownRendererFancyOListItemEnd "}
6940     else
6941         return {"\\markdownRendererFancyOListItem " ,s,"\\markdownRendererFancyOListItemEnd "}
6942     end
6943 end
6944 end, extend_reader = function(self)
6945     local parsers = self.parsers
6946     local options = self.options
6947     local writer = self.writer
6948
6949     local label = parsers.dig + parsers.letter
6950     local numdelim = parsers.period + parsers.rparent
6951     local enumerator = C(label^3 * numdelim) * #parsers.spacing
6952                     + C(label^2 * numdelim) * #parsers.spacing
6953                       * (parsers.tab + parsers.space^1)
6954                     + C(label * numdelim) * #parsers.spacing
6955                       * (parsers.tab + parsers.space^-
2)
6956                     + parsers.space * C(label^2 * numdelim)
6957                       * #parsers.spacing
6958                     + parsers.space * C(label * numdelim)
6959                       * #parsers.spacing
6960                       * (parsers.tab + parsers.space^-
1)
6961                     + parsers.space * parsers.space * C(label^1
6962                       * numdelim) * #parsers.spacing
6963     local starter = parsers.bullet + enumerator
6964
6965     local NestedList = Cs((parsers.optionallyindentedline
6966                          - starter)^1)
6967                       / function(a) return "\\001"..a end
6968
6969     local ListBlockLine = parsers.optionallyindentedline
6970                          - parsers.blankline - (parsers.indent^-1
6971                                                  * starter)
6972
6973     local ListBlock = parsers.line * ListBlockLine^0
6974
6975     local ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
6976                               * ListBlock
6977
6978     local TightListItem = function(starter)
6979         return -parsers.ThematicBreak
6980                * (Cs(starter / "" * parsers.tickbox^-1 * ListBlock * NestedList^-
1)
6981                  / self.parser_functions.parse_blocks_nested)

```



```

6982         * -(parsers.blanklines * parsers.indent)
6983     end
6984
6985     local LooseListItem = function(starter)
6986         return -parsers.ThematicBreak
6987             * Cs( starter / "" * parsers.tickbox^-1 * ListBlock * Cc("\n")
6988                 * (NestedList + ListContinuationBlock^0)
6989                 * (parsers.blanklines / "\n\n")
6990                 ) / self.parser_functions.parse_blocks_nested
6991     end
6992
6993     local function roman2number(roman)
6994         local romans = { ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
6995         local numeral = 0
6996
6997         local i = 1
6998         local len = string.len(roman)
6999         while i < len do
7000             local z1, z2 = romans[ string.sub(roman, i, i) ], romans[ string.sub(roman,
7001                 i+1, i+1) ]
7002             if z1 < z2 then
7003                 numeral = numeral + (z2 - z1)
7004                 i = i + 2
7005             else
7006                 numeral = numeral + z1
7007                 i = i + 1
7008             end
7009         end
7010         if i <= len then numeral = numeral + romans[ string.sub(roman,i,i) ] end
7011         return numeral
7012     end
7013
7014     local function sniffstyle(itemprefix)
7015         local numstr, delimend = itemprefix:match("^([A-Za-z0-9]*)([.])")
7016         local numdelim
7017         if delimend == ")" then
7018             numdelim = "OneParen"
7019         elseif delimend == "." then
7020             numdelim = "Period"
7021         else
7022             numdelim = "Default"
7023         end
7024         numstr = numstr or itemprefix
7025
7026         local num
7027         num = numstr:match("^([IVXL]+)")
7028         if num then
7029             return roman2number(num), "UpperRoman", numdelim

```

```

7029     end
7030     num = numstr:match("^([ivxl]+)")
7031     if num then
7032         return roman2number(string.upper(num)), "LowerRoman", numdelim
7033     end
7034     num = numstr:match("^([A-Z])")
7035     if num then
7036         return string.byte(num) - string.byte("A") + 1, "UpperAlpha", numdelim
7037     end
7038     num = numstr:match("^([a-z])")
7039     if num then
7040         return string.byte(num) - string.byte("a") + 1, "LowerAlpha", numdelim
7041     end
7042     return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
7043 end
7044
7045 local function fancylist(items,tight,start)
7046     local startnum, numstyle, numdelim = sniffstyle(start)
7047     return writer.fancylist(items,tight,
7048                             options.startNumber and startnum,
7049                             numstyle or "Decimal",
7050                             numdelim or "Default")
7051 end
7052
7053 local FancyList = Cg(enumerator, "listtype") *
7054     ( Ct(TightListItem(Cb("listtype")))
7055       * TightListItem(enumerator)^0
7056       * Cc(true) * parsers.skipblanklines * -enumerator
7057       + Ct(LooseListItem(Cb("listtype")))
7058         * LooseListItem(enumerator)^0
7059       * Cc(false) * parsers.skipblanklines
7060       ) * Cb("listtype") / fancylist
7061
7062     self.update_rule("OrderedList", function() return FancyList end)
7063 end
7064 }
7065 end

```

3.1.6.6 Fenced Code The `extensions.fenced_code` function implements the commonmark fenced code block syntax extension. When the `blank_before_code_fence` parameter is `true`, the syntax extension requires a blank line between a paragraph and the following fenced code block.

When the `allow_attributes` option is `true`, the syntax extension permits attributes following the infostring. When the `allow_raw_blocks` option is `true`, the

syntax extension permits the specification of raw blocks using Pandoc's raw attribute syntax extension.

```
7066 M.extensions.fenced_code = function(blank_before_code_fence,  
7067                                     allow_attributes,  
7068                                     allow_raw_blocks)  
7069   return {  
7070     name = "built-in fenced_code syntax extension",  
7071     extend_writer = function(self)  
7072       local options = self.options  
7073
```

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.

```
7074     function self.fencedCode(s, i, attr)  
7075       if not self.is_writing then return "" end  
7076       s = s:gsub("\n$", "")  
7077       local buf = {}  
7078       if attr ~= nil then  
7079         table.insert(buf, {"\\markdownRendererFencedCodeAttributeContextBegin",  
7080                             self.attributes(attr)})  
7081       end  
7082       local name = util.cache_verbatim(options.cacheDir, s)  
7083       table.insert(buf, {"\\markdownRendererInputFencedCode{",  
7084                             name,"}{",self.string(i),"}")  
7085       if attr ~= nil then  
7086         table.insert(buf, "\\markdownRendererFencedCodeAttributeContextEnd")  
7087       end  
7088       return buf  
7089     end  
7090
```

Define `writer->rawBlock` as a function that will transform an input raw block `s` with the raw attribute `attr` to the output format.

```
7091     if allow_raw_blocks then  
7092       function self.rawBlock(s, attr)  
7093         if not self.is_writing then return "" end  
7094         s = s:gsub("\n$", "")  
7095         local name = util.cache_verbatim(options.cacheDir, s)  
7096         return {"\\markdownRendererInputRawBlock{",  
7097                             name,"}{", self.string(attr),"}")  
7098       end  
7099     end  
7100   end, extend_reader = function(self)  
7101     local parsers = self.parsers  
7102     local writer = self.writer  
7103  
7104     local function captures_geq_length(_,i,a,b)
```

```

7105     return #a >= #b and i
7106 end
7107
7108 local tilde_infostring
7109     = C((parsers.linechar
7110         - (parsers.spacechar^1 * parsers.newline))^0)
7111
7112 local backtick_infostring
7113     = C((parsers.linechar
7114         - (parsers.backtick
7115           + parsers.spacechar^1 * parsers.newline))^0)
7116
7117 local fenceindent
7118 local fencehead = function(char, infostring)
7119     return C(parsers.nonindentospace) / function(s) fenceindent = #s
7120     * Cg(char^3, "fencelength")
7121     * parsers.optionalspace
7122     * infostring
7123     * (parsers.newline + parsers.eof)
7124 end
7125
7126 local fencetail = function(char)
7127     return parsers.nonindentospace
7128     * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
7129     * parsers.optionalspace * (parsers.newline + parsers.eof)
7130     + parsers.eof
7131 end
7132
7133 local fencedline = function(char)
7134     return C(parsers.line - fencetail(char))
7135     / function(s)
7136         local i = 1
7137         local remaining = fenceindent
7138         while true do
7139             local c = s:sub(i, i)
7140             if c == " " and remaining > 0 then
7141                 remaining = remaining - 1
7142                 i = i + 1
7143             elseif c == "\t" and remaining > 3 then
7144                 remaining = remaining - 4
7145                 i = i + 1
7146             else
7147                 break
7148             end
7149         end
7150         return s:sub(i)
7151     end

```

```

7152     end
7153
7154     local TildeFencedCode
7155         = fencehead(parsers.tilde, tilde_infostring)
7156         * Cs(fencedline(parsers.tilde)^0)
7157         * fencetail(parsers.tilde)
7158
7159     local BacktickFencedCode
7160         = fencehead(parsers.backtick, backtick_infostring)
7161         * Cs(fencedline(parsers.backtick)^0)
7162         * fencetail(parsers.backtick)
7163
7164     local infostring_with_attributes
7165         = Ct(C((parsers.linechar
7166             - ( parsers.optionalspace
7167               * parsers.attributes))^0)
7168             * parsers.optionalspace
7169             * Ct(parsers.attributes))
7170
7171     local FencedCode
7172         = (TildeFencedCode + BacktickFencedCode)
7173         / function(infostring, code)
7174             local expanded_code = self.expandtabs(code)
7175
7176             if allow_raw_blocks then
7177                 local raw_attr = lpeg.match(parsers.raw_attribute,
7178                                         infostring)
7179
7180                 if raw_attr then
7181                     return writer.rawBlock(expanded_code, raw_attr)
7182                 end
7183             end
7184
7185             local attr = nil
7186             if allow_attributes then
7187                 local match = lpeg.match(infostring_with_attributes,
7188                                         infostring)
7189
7190                 if match then
7191                     infostring, attr = table.unpack(match)
7192                 end
7193             end
7194
7195             return writer.fencedCode(expanded_code, infostring, attr)
7196         end
7197
7198     self.insert_pattern("Block after Verbatim",
7199                       FencedCode, "FencedCode")
7200
7201     local fencestart

```

```

7199     if blank_before_code_fence then
7200         fencestart = parsers.fail
7201     else
7202         fencestart = fencehead(parsers.backtick, backtick_infostring)
7203             + fencehead(parsers.tilde, tilde_infostring)
7204     end
7205
7206     self.update_rule("EndlineExceptions", function(previous_pattern)
7207         if previous_pattern == nil then
7208             previous_pattern = parsers.EndlineExceptions
7209         end
7210         return previous_pattern + fencestart
7211     end)
7212
7213     self.add_special_character("`")
7214     self.add_special_character("~")
7215 end
7216 }
7217 end

```

3.1.6.7 Fenced Divs The `extensions.fenced_divs` function implements the Pandoc fenced divs 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.

```

7218 M.extensions.fenced_divs = function(blank_before_div_fence)
7219     return {
7220         name = "built-in fenced_divs syntax extension",
7221         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.

```

7222         function self.div_begin(attributes)
7223             local start_output = {"\markdownRendererFencedDivAttributeContextBegin\n",
7224                 self.attributes(attributes)}
7225             local end_output = {"\n\markdownRendererFencedDivAttributeContextEnd "}
7226             return self.push_attributes("div", attributes, start_output, end_output)
7227         end

```

Define `writer->div_end` as a function that will produce the end of a fenced div in the output format.

```

7228         function self.div_end()
7229             return self.pop_attributes("div")
7230         end
7231     end, extend_reader = function(self)
7232         local parsers = self.parsers
7233         local writer = self.writer

```

Define basic patterns for matching the opening and the closing tag of a div.

```
7234     local fenced_div_infostring
7235         = C((parsers.linechar
7236             - ( parsers.spacechar^1
7237               * parsers.colon^1))^1)
7238
7239     local fenced_div_begin = parsers.nonindentspace
7240         * parsers.colon^3
7241         * parsers.optionalspace
7242         * fenced_div_infostring
7243         * ( parsers.spacechar^1
7244           * parsers.colon^1)^0
7245         * parsers.optionalspace
7246         * (parsers.newline + parsers.eof)
7247
7248     local fenced_div_end = parsers.nonindentspace
7249         * parsers.colon^3
7250         * parsers.optionalspace
7251         * (parsers.newline + parsers.eof)
```

Initialize a named group named `div_level` for tracking how deep we are nested in divs.

```
7252     self.initialize_named_group("div_level", "0")
7253
7254     local function increment_div_level(increment)
7255         local function update_div_level(s, i, current_level) -- luacheck: ignore s i
7256             current_level = tonumber(current_level)
7257             local next_level = tostring(current_level + increment)
7258             return true, next_level
7259         end
7260
7261         return Cg( Cmt(Cb("div_level"), update_div_level)
7262                 , "div_level")
7263     end
7264
7265     local FencedDiv = fenced_div_begin
7266         / function (infostring)
7267             local attr = lpeg.match(Ct(parsers.attributes), infostring)
7268             if attr == nil then
7269                 attr = {".." .. infostring}
7270             end
7271             return attr
7272         end
7273         / writer.div_begin
7274         * increment_div_level(1)
7275         * parsers.skipblanklines
7276         * Ct( (V("Block") - fenced_div_end)^-1
```

```

7277         * ( parsers.blanklines
7278           / function()
7279             return writer.interblocksep
7280           end
7281         * (V("Block") - fenced_div_end)^0)
7282     * parsers.skipblanklines
7283     * fenced_div_end * increment_div_level(-1)
7284     * (Cc("") / writer.div_end)
7285
7286     self.insert_pattern("Block after Verbatim",
7287                       FencedDiv, "FencedDiv")
7288
7289     self.add_special_character(":")
7290

```

Patch blockquotes, so that they allow the end of a fenced div immediately afterwards.

```

7291     local function check_div_level(s, i, current_level) -- luacheck: ignore s i
7292       current_level = tonumber(current_level)
7293       return current_level > 0
7294     end
7295
7296     local is_inside_div = Cmt(Cb("div_level"), check_div_level)
7297     local fencestart = is_inside_div * fenced_div_end
7298
7299     self.update_rule("BlockquoteExceptions", function(previous_pattern)
7300       if previous_pattern == nil then
7301         previous_pattern = parsers.BlockquoteExceptions
7302       end
7303       return previous_pattern + fencestart
7304     end)
7305

```

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.

```

7306     if not blank_before_div_fence then
7307       self.update_rule("EndlineExceptions", function(previous_pattern)
7308         if previous_pattern == nil then
7309           previous_pattern = parsers.EndlineExceptions
7310         end
7311         return previous_pattern + fencestart
7312       end)
7313     end
7314   end
7315 }
7316 end

```


3.1.6.8 Header Attributes The `extensions.header_attributes` function implements the Pandoc header attributes syntax extension.

```

7317 M.extensions.header_attributes = function()
7318   return {
7319     name = "built-in header_attributes syntax extension",
7320     extend_writer = function()
7321     end, extend_reader = function(self)
7322       local parsers = self.parsers
7323       local writer = self.writer
7324
7325       local AtxHeading = Cg(parsers.heading_start, "level")
7326         * parsers.optionalspace
7327         * (C(((parsers.linechar
7328           - ((parsers.hash^1
7329             * parsers.optionalspace
7330             * parsers.attributes^-1
7331             + parsers.attributes)
7332             * parsers.optionalspace
7333             * parsers.newline)))
7334           * (parsers.linechar
7335             - parsers.hash
7336             - parsers.lbrace)^0)^1)
7337         / self.parser_functions.parse_inlines)
7338     * Cg(Ct(parsers.newline
7339       + (parsers.hash^1
7340         * parsers.optionalspace
7341         * parsers.attributes^-1
7342         + parsers.attributes)
7343         * parsers.optionalspace
7344         * parsers.newline), "attributes")
7345     * Cb("level")
7346     * Cb("attributes")
7347     / writer.heading
7348
7349     local SetextHeading = #(parsers.line * S("-"))
7350       * (C(((parsers.linechar
7351         - (parsers.attributes
7352           * parsers.optionalspace
7353           * parsers.newline)))
7354         * (parsers.linechar
7355           - parsers.lbrace)^0)^1)
7356       / self.parser_functions.parse_inlines)
7357     * Cg(Ct(parsers.newline
7358       + (parsers.attributes
7359         * parsers.optionalspace
7360         * parsers.newline)), "attributes")
7361     * parsers.heading_level

```

```

7362             * Cb("attributes")
7363             * parsers.optionalspace
7364             * parsers.newline
7365             / writer.heading
7366
7367         local Heading = AtxHeading + SetextHeading
7368         self.update_rule("Heading", function() return Heading end)
7369     end
7370 }
7371 end

```

3.1.6.9 Line Blocks The `extensions.line_blocks` function implements the Pandoc line blocks syntax extension.

```

7372 M.extensions.line_blocks = function()
7373     return {
7374         name = "built-in line_blocks syntax extension",
7375         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.

```

7376         function self.lineblock(lines)
7377             if not self.is_writing then return "" end
7378             local buffer = {}
7379             for i = 1, #lines - 1 do
7380                 buffer[#buffer + 1] = { lines[i], self.hard_line_break }
7381             end
7382             buffer[#buffer + 1] = lines[#lines]
7383
7384             return {"\\markdownRendererLineBlockBegin\n"
7385                 ,buffer,
7386                 "\\n\\markdownRendererLineBlockEnd "}
7387         end
7388     end, extend_reader = function(self)
7389         local parsers = self.parsers
7390         local writer = self.writer
7391
7392         local LineBlock = Ct(
7393             (Cs(
7394                 ( (parsers.pipe * parsers.space)/""
7395                 * ((parsers.space)/entities.char_entity("nbsp"))^0
7396                 * parsers.linechar^0 * (parsers.newline/"")
7397                 * (-parsers.pipe
7398                 * (parsers.space^1/" ")
7399                 * parsers.linechar^1
7400                 * (parsers.newline/"")
7401                 )^0
7402                 * (parsers.blankline/"")^0

```

```

7403         ) / self.parser_functions.parse_inlines)^1) / writer.lineblock
7404
7405     self.insert_pattern("Block after Blockquote",
7406                       LineBlock, "LineBlock")
7407 end
7408 }
7409 end

```

3.1.6.10 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.

```

7410 M.extensions.notes = function(notes, inline_notes)
7411   assert(notes or inline_notes)
7412   return {
7413     name = "built-in notes syntax extension",
7414     extend_writer = function(self)

```

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

```

7415     function self.note(s)
7416       return {"\\markdownRendererNote{",s,"}"}
7417     end
7418   end, extend_reader = function(self)
7419     local parsers = self.parsers
7420     local writer = self.writer
7421
7422     if inline_notes then
7423       local InlineNote
7424         = parsers.circumflex
7425         * (parsers.tag / self.parser_functions.parse_inlines_no_inline_no
7426           / writer.note
7427
7428       self.insert_pattern("Inline after Emph",
7429                         InlineNote, "InlineNote")
7430     end
7431     if notes then
7432       local function strip_first_char(s)
7433         return s:sub(2)
7434       end
7435
7436       local RawNoteRef
7437         = #(parsers.lbracket * parsers.circumflex)
7438         * parsers.tag / strip_first_char
7439
7440     local rawnotes = {}

```

```

7441
7442     -- like indirect_link
7443     local function lookup_note(ref)
7444         return writer.defer_call(function()
7445             local found = rawnotes[self.normalize_tag(ref)]
7446             if found then
7447                 return writer.note(
7448                     self.parser_functions.parse_blocks_nested(found))
7449             else
7450                 return {"[",
7451                     self.parser_functions.parse_inlines("^" .. ref), "]" }
7452             end
7453         end)
7454     end
7455
7456     local function register_note(ref,rawnote)
7457         rawnotes[self.normalize_tag(ref)] = rawnote
7458         return ""
7459     end
7460
7461     local NoteRef = RawNoteRef / lookup_note
7462
7463     local NoteBlock
7464         = parsers.leader * RawNoteRef * parsers.colon
7465         * parsers.spnl * parsers.indented_blocks(parsers.chunk)
7466         / register_note
7467
7468     local Blank = NoteBlock + parsers.Blank
7469     self.update_rule("Blank", function() return Blank end)
7470
7471     self.insert_pattern("Inline after Emph",
7472                         NoteRef, "NoteRef")
7473     end
7474
7475     self.add_special_character("^")
7476 end
7477 }
7478 end

```

3.1.6.11 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_captions` syntax extension for table captions.

```

7479 M.extensions.pipe_tables = function(table_captions)
7480
7481     local function make_pipe_table_rectangular(rows)

```

```

7482     local num_columns = #rows[2]
7483     local rectangular_rows = {}
7484     for i = 1, #rows do
7485         local row = rows[i]
7486         local rectangular_row = {}
7487         for j = 1, num_columns do
7488             rectangular_row[j] = row[j] or ""
7489         end
7490         table.insert(rectangular_rows, rectangular_row)
7491     end
7492     return rectangular_rows
7493 end
7494
7495 local function pipe_table_row(allow_empty_first_column
7496                             , nonempty_column
7497                             , column_separator
7498                             , column)
7499     local row_beginning
7500     if allow_empty_first_column then
7501         row_beginning = -- empty first column
7502             #(parsers.spacechar^4
7503             * column_separator)
7504             * parsers.optionalspace
7505             * column
7506             * parsers.optionalspace
7507             -- non-empty first column
7508             + parsers.nonindentspace
7509             * nonempty_column^-1
7510             * parsers.optionalspace
7511     else
7512         row_beginning = parsers.nonindentspace
7513             * nonempty_column^-1
7514             * parsers.optionalspace
7515     end
7516
7517     return Ct(row_beginning
7518             * (-- single column with no leading pipes
7519             #(column_separator
7520             * parsers.optionalspace
7521             * parsers.newline)
7522             * column_separator
7523             * parsers.optionalspace
7524             -- single column with leading pipes or
7525             -- more than a single column
7526             + (column_separator
7527             * parsers.optionalspace
7528             * column

```

```

7529         * parsers.optionalspace)^1
7530     * (column_separator
7531         * parsers.optionalspace)^-1))
7532 end
7533
7534 return {
7535     name = "built-in pipe_tables syntax extension",
7536     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.

```

7537     function self.table(rows, caption)
7538         if not self.is_writing then return "" end
7539         local buffer = {"\\markdownRendererTable{",
7540             caption or "", "}{" , #rows - 1, "}{" , #rows[1], "}"}
7541         local temp = rows[2] -- put alignments on the first row
7542         rows[2] = rows[1]
7543         rows[1] = temp
7544         for i, row in ipairs(rows) do
7545             table.insert(buffer, "{")
7546             for _, column in ipairs(row) do
7547                 if i > 1 then -- do not use braces for alignments
7548                     table.insert(buffer, "{")
7549                 end
7550                 table.insert(buffer, column)
7551                 if i > 1 then
7552                     table.insert(buffer, "}")
7553                 end
7554             end
7555             table.insert(buffer, "}")
7556         end
7557         return buffer
7558     end
7559 end, extend_reader = function(self)
7560     local parsers = self.parsers
7561     local writer = self.writer
7562
7563     local table_hline_separator = parsers.pipe + parsers.plus
7564
7565     local table_hline_column = (parsers.dash
7566         - #(parsers.dash
7567             * (parsers.spacechar
7568                 + table_hline_separator
7569                 + parsers.newline)))^1
7570     * (parsers.colon * Cc("r")
7571         + parsers.dash * Cc("d"))

```

```

7572         + parsers.colon
7573         * (parsers.dash
7574           - #(parsers.dash
7575             * (parsers.spacechar
7576               + table_hline_separator
7577               + parsers.newline)))^1
7578         * (parsers.colon * Cc("c")
7579           + parsers.dash * Cc("l"))
7580
7581     local table_hline = pipe_table_row(false
7582                                       , table_hline_column
7583                                       , table_hline_separator
7584                                       , table_hline_column)
7585
7586     local table_caption_beginning = parsers.skipblanklines
7587                                   * parsers.nonindentSPACE
7588                                   * (P("Table")^-1 * parsers.colon)
7589                                   * parsers.optionalspace
7590
7591     local table_row = pipe_table_row(true
7592                                     , ((parsers.linechar - parsers.pipe)^1)
7593                                       / self.parser_functions.parse_inlines)
7594                                   , parsers.pipe
7595                                   , ((parsers.linechar - parsers.pipe)^0)
7596                                       / self.parser_functions.parse_inlines))
7597
7598     local table_caption
7599     if table_captions then
7600         table_caption = #table_caption_beginning
7601                       * table_caption_beginning
7602                       * Ct(parsers.IndentedInline^1)
7603                       * parsers.newline
7604     else
7605         table_caption = parsers.fail
7606     end
7607
7608     local PipeTable = Ct(table_row * parsers.newline
7609                       * table_hline
7610                       * (parsers.newline * table_row)^0)
7611                       / make_pipe_table_rectangular
7612                       * table_caption^-1
7613                       / writer.table
7614
7615     self.insert_pattern("Block after Blockquote",
7616                       PipeTable, "PipeTable")
7617 end
7618 }

```

```
7619 end
```

3.1.6.12 Raw Attributes The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```
7620 M.extensions.raw_inline = function()
7621   return {
7622     name = "built-in raw_inline syntax extension",
7623     extend_writer = function(self)
7624       local options = self.options
7625
```

Define `writer->rawInline` as a function that will transform an input inline raw span `s` with the raw attribute `attr` to the output format.

```
7626     function self.rawInline(s, attr)
7627       if not self.is_writing then return "" end
7628       local name = util.cache_verbatim(options.cacheDir, s)
7629       return {"\\markdownRendererInputRawInline{" ,
7630             name,"}{" , self.string(attr),"}"}
7631     end
7632   end, extend_reader = function(self)
7633     local writer = self.writer
7634
7635     local RawInline = parsers.inticks
7636                       * parsers.raw_attribute
7637                       / writer.rawInline
7638
7639     self.insert_pattern("Inline before Code",
7640                       RawInline, "RawInline")
7641   end
7642 }
7643 end
```

3.1.6.13 Strike-Through The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```
7644 M.extensions.strike_through = function()
7645   return {
7646     name = "built-in strike_through syntax extension",
7647     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.

```
7648     function self.strike_through(s)
7649       return {"\\markdownRendererStrikeThrough{" ,s,"}"}
7650     end
7651   end, extend_reader = function(self)
7652     local parsers = self.parsers
```



```

7653     local writer = self.writer
7654
7655     local StrikeThrough = (
7656         parsers.between(parsers.Inline, parsers.doubletildes,
7657             parsers.doubletildes)
7658     ) / writer.strike_through
7659
7660     self.insert_pattern("Inline after Emph",
7661         StrikeThrough, "StrikeThrough")
7662
7663     self.add_special_character("~")
7664 end
7665 }
7666 end

```

3.1.6.14 Subscripts The `extensions.subscripts` function implements the Pandoc subscript syntax extension.

```

7667 M.extensions.subscripts = function()
7668     return {
7669         name = "built-in subscripts syntax extension",
7670         extend_writer = function(self)

```

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

```

7671         function self.subscript(s)
7672             return {"\\markdownRendererSubscript{" ,s,""}"}
7673         end
7674     end, extend_reader = function(self)
7675         local parsers = self.parsers
7676         local writer = self.writer
7677
7678         local Subscript = (
7679             parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
7680         ) / writer.subscript
7681
7682         self.insert_pattern("Inline after Emph",
7683             Subscript, "Subscript")
7684
7685         self.add_special_character("~")
7686     end
7687 }
7688 end

```

3.1.6.15 Superscripts The `extensions.superscripts` function implements the Pandoc superscript syntax extension.

```

7689 M.extensions.superscripts = function()

```

```

7690 return {
7691   name = "built-in superscripts syntax extension",
7692   extend_writer = function(self)

```

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

```

7693     function self.superscript(s)
7694       return {"\\markdownRendererSuperscript{" ,s,"}"}
7695     end
7696   end, extend_reader = function(self)
7697     local parsers = self.parsers
7698     local writer = self.writer
7699
7700     local Superscript = (
7701       parsers.between(parsers.Str, parsers.circumflex, parsers.circumflex)
7702     ) / writer.superscript
7703
7704     self.insert_pattern("Inline after Emph",
7705                       Superscript, "Superscript")
7706
7707     self.add_special_character("^")
7708   end
7709 }
7710 end

```

3.1.6.16 Tex Math Dollars The `extensions.tex_math_dollars` function implements the Pandoc `tex_math_dollars` syntax extension.

```

7711 M.extensions.tex_math_dollars = function()
7712 return {
7713   name = "built-in tex_math_dollars syntax extension",
7714   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.

```

7715     function self.display_math(s)
7716       if not self.is_writing then return "" end
7717       return {"\\markdownRendererDisplayMath{" ,self.math(s),"}"}
7718     end

```

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

```

7719     function self.inline_math(s)
7720       if not self.is_writing then return "" end
7721       return {"\\markdownRendererInlineMath{" ,self.math(s),"}"}
7722     end
7723   end, extend_reader = function(self)
7724     local parsers = self.parsers

```

```

7725     local writer = self.writer
7726
7727     local function between(p, starter, ender)
7728         return (starter * C(p * (p - ender)^0) * ender)
7729     end
7730
7731     local inlinemathtail = B( parsers.any * parsers.nonspacechar
7732         + parsers.backslash * parsers.any)
7733         * parsers.dollar
7734         * -#(parsers.digit)
7735
7736     local inlinemath = between(C( parsers.backslash^-1
7737         * parsers.any
7738         - parsers.blankline^2
7739         - parsers.dollar),
7740         parsers.dollar * #(parsers.nonspacechar),
7741         inlinemathtail)
7742
7743     local displaymathdelim = parsers.dollar
7744         * parsers.dollar
7745
7746     local displaymath = between(C( parsers.backslash^-1
7747         * parsers.any
7748         - parsers.blankline^2
7749         - parsers.dollar),
7750         displaymathdelim,
7751         displaymathdelim)
7752
7753     local TexMathDollars = displaymath / writer.display_math
7754         + inlinemath / writer.inline_math
7755
7756     self.insert_pattern("Inline after Emph",
7757         TexMathDollars, "TexMathDollars")
7758
7759     self.add_special_character("$")
7760 end
7761 }
7762 end

```

3.1.6.17 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.

```

7763 M.extensions.jekyll_data = function(expect_jekyll_data)
7764     return {
7765         name = "built-in jekyll_data syntax extension",

```

```
7766     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`.

```
7767     function self.jekyllData(d, t, p)
7768         if not self.is_writing then return "" end
7769
7770         local buf = {}
7771
7772         local keys = {}
7773         for k, _ in pairs(d) do
7774             table.insert(keys, k)
7775         end
7776         table.sort(keys)
7777
7778         if not p then
7779             table.insert(buf, "\\markdownRendererJekyllDataBegin")
7780         end
7781
7782         if #d > 0 then
7783             table.insert(buf, "\\markdownRendererJekyllDataSequenceBegin{")
7784             table.insert(buf, self.identifier(p or "null"))
7785             table.insert(buf, "}{")
7786             table.insert(buf, #keys)
7787             table.insert(buf, "}")
7788         else
7789             table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
7790             table.insert(buf, self.identifier(p or "null"))
7791             table.insert(buf, "}{")
7792             table.insert(buf, #keys)
7793             table.insert(buf, "}")
7794         end
7795
7796         for _, k in ipairs(keys) do
7797             local v = d[k]
7798             local typ = type(v)
7799             k = tostring(k or "null")
7800             if typ == "table" and next(v) ~= nil then
7801                 table.insert(
7802                     buf,
7803                     self.jekyllData(v, t, k)
7804                 )
7805             else
7806                 k = self.identifier(k)
```

```

7807         v = tostring(v)
7808         if typ == "boolean" then
7809             table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
7810             table.insert(buf, k)
7811             table.insert(buf, "{")
7812             table.insert(buf, v)
7813             table.insert(buf, "}")
7814         elseif typ == "number" then
7815             table.insert(buf, "\\markdownRendererJekyllDataNumber{")
7816             table.insert(buf, k)
7817             table.insert(buf, "{")
7818             table.insert(buf, v)
7819             table.insert(buf, "}")
7820         elseif typ == "string" then
7821             table.insert(buf, "\\markdownRendererJekyllDataString{")
7822             table.insert(buf, k)
7823             table.insert(buf, "{")
7824             table.insert(buf, t(v))
7825             table.insert(buf, "}")
7826         elseif typ == "table" then
7827             table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
7828             table.insert(buf, k)
7829             table.insert(buf, "}")
7830         else
7831             error(format("Unexpected type %s for value of " ..
7832                 "YAML key %s", typ, k))
7833         end
7834     end
7835 end
7836
7837 if #d > 0 then
7838     table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
7839 else
7840     table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
7841 end
7842
7843 if not p then
7844     table.insert(buf, "\\markdownRendererJekyllDataEnd")
7845 end
7846
7847 return buf
7848 end
7849 end, extend_reader = function(self)
7850     local parsers = self.parsers
7851     local writer = self.writer
7852
7853     local JekyllData

```

```

7854         = Cmt( C((parsers.line - P("----") - P("..."))^0)
7855           , function(s, i, text) -- luacheck: ignore s i
7856             local data
7857             local ran_ok, _ = pcall(function()
7858               local tinyyaml = require("markdown-tinyyaml")
7859               data = tinyyaml.parse(text, {timestamps=false})
7860             end)
7861             if ran_ok and data ~= nil then
7862               return true, writer.jekyllData(data, function(s)
7863                 return self.parser_functions.parse_blocks_nested(s)
7864               end, nil)
7865             else
7866               return false
7867             end
7868           end
7869         )
7870
7871     local UnexpectedJekyllData
7872     = P("----")
7873     * parsers.blankline / 0
7874     * #(-parsers.blankline) -- if followed by blank, it's thematic b
7875     * JekyllData
7876     * (P("----") + P("..."))
7877
7878     local ExpectedJekyllData
7879     = ( P("----")
7880       * parsers.blankline / 0
7881       * #(-parsers.blankline) -- if followed by blank, it's thematic
7882       )^-1
7883     * JekyllData
7884     * (P("----") + P("..."))^-1
7885
7886     self.insert_pattern("Block before Blockquote",
7887       UnexpectedJekyllData, "UnexpectedJekyllData")
7888     if expect_jekyll_data then
7889       self.update_rule("ExpectedJekyllData", function() return ExpectedJekyllData e
7890     end
7891   end
7892 }
7893 end

```

3.1.7 Conversion from Markdown to Plain T_EX

The `new` function returns a conversion function that takes a markdown string and turns it into a plain T_EX output. See Section 2.1.1.

```

7894 function M.new(options)

```

Make the `options` table inherit from the `defaultOptions` table.

```
7895 options = options or {}
7896 setmetatable(options, { __index = function (_, key)
7897     return defaultOptions[key] end })
```

Apply built-in syntax extensions based on `options`.

```
7898 local extensions = {}
7899
7900 if options.bracketedSpans then
7901     local bracketed_spans_extension = M.extensions.bracketed_spans()
7902     table.insert(extensions, bracketed_spans_extension)
7903 end
7904
7905 if options.contentBlocks then
7906     local content_blocks_extension = M.extensions.content_blocks(
7907         options.contentBlocksLanguageMap)
7908     table.insert(extensions, content_blocks_extension)
7909 end
7910
7911 if options.definitionLists then
7912     local definition_lists_extension = M.extensions.definition_lists(
7913         options.tightLists)
7914     table.insert(extensions, definition_lists_extension)
7915 end
7916
7917 if options.fencedCode then
7918     local fenced_code_extension = M.extensions.fenced_code(
7919         options.blankBeforeCodeFence,
7920         options.fencedCodeAttributes,
7921         options.rawAttribute)
7922     table.insert(extensions, fenced_code_extension)
7923 end
7924
7925 if options.fencedDivs then
7926     local fenced_div_extension = M.extensions.fenced_divs(
7927         options.blankBeforeDivFence)
7928     table.insert(extensions, fenced_div_extension)
7929 end
7930
7931 if options.headerAttributes then
7932     local header_attributes_extension = M.extensions.header_attributes()
7933     table.insert(extensions, header_attributes_extension)
7934 end
7935
7936 if options.jekyllData then
7937     local jekyll_data_extension = M.extensions.jekyll_data(
7938         options.expectJekyllData)
```

```

7939     table.insert(extensions, jekyll_data_extension)
7940 end
7941
7942 if options.lineBlocks then
7943     local line_block_extension = M.extensions.line_blocks()
7944     table.insert(extensions, line_block_extension)
7945 end
7946
7947 if options.pipeTables then
7948     local pipe_tables_extension = M.extensions.pipe_tables(
7949         options.tableCaptions)
7950     table.insert(extensions, pipe_tables_extension)
7951 end
7952
7953 if options.rawAttribute then
7954     local raw_inline_extension = M.extensions.raw_inline()
7955     table.insert(extensions, raw_inline_extension)
7956 end
7957
7958 if options.strikeThrough then
7959     local strike_through_extension = M.extensions.strike_through()
7960     table.insert(extensions, strike_through_extension)
7961 end
7962
7963 if options.subscripts then
7964     local subscript_extension = M.extensions.subscripts()
7965     table.insert(extensions, subscript_extension)
7966 end
7967
7968 if options.superscripts then
7969     local superscript_extension = M.extensions.superscripts()
7970     table.insert(extensions, superscript_extension)
7971 end
7972
7973 if options.texMathDollars then
7974     local tex_math_dollars_extension = M.extensions.tex_math_dollars()
7975     table.insert(extensions, tex_math_dollars_extension)
7976 end
7977

```

The footnotes and inlineFootnotes option has been deprecated and will be removed in Markdown 3.0.0.

```

7978 if options.footnotes or options.inlineFootnotes or
7979     options.notes or options.inlineNotes then
7980     local notes_extension = M.extensions.notes(
7981         options.footnotes or options.notes,
7982         options.inlineFootnotes or options.inlineNotes)

```



```

7983     table.insert(extensions, notes_extension)
7984 end
7985
7986 if options.citations then
7987     local citations_extension = M.extensions.citations(options.citationNbsps)
7988     table.insert(extensions, citations_extension)
7989 end
7990
7991 if options.fancyLists then
7992     local fancy_lists_extension = M.extensions.fancy_lists()
7993     table.insert(extensions, fancy_lists_extension)
7994 end

```

Apply user-defined syntax extensions based on `options.extensions`.

```

7995 for _, user_extension_filename in ipairs(options.extensions) do
7996     local user_extension = (function(filename)

```

First, load and compile the contents of the user-defined syntax extension.

```

7997     local pathname = util.lookup_files(filename)
7998     local input_file = assert(io.open(pathname, "r"),
7999         [[Could not open user-defined syntax extension ]]
8000         .. pathname .. [{" for reading"}])
8001     local input = assert(input_file:read("*a"))
8002     assert(input_file:close())
8003     local user_extension, err = load([[
8004         local sandbox = {}
8005         setmetatable(sandbox, {__index = _G})
8006         _ENV = sandbox
8007     ]] .. input)()
8008     assert(user_extension,
8009         [[Failed to compile user-defined syntax extension ]]
8010         .. pathname .. [{": "} .. (err or [{"})])

```

Then, validate the user-defined syntax extension.

```

8011     assert(user_extension.api_version ~= nil,
8012         [[User-defined syntax extension ]] .. pathname
8013         .. [{" does not specify mandatory field "api_version"}])
8014     assert(type(user_extension.api_version) == "number",
8015         [[User-defined syntax extension ]] .. pathname
8016         .. [{" specifies field "api_version" of type "}]
8017         .. type(user_extension.api_version)
8018         .. [{" but "number" was expected}])
8019     assert(user_extension.api_version > 0
8020         and user_extension.api_version <= metadata.user_extension_api_version,
8021         [[User-defined syntax extension ]] .. pathname
8022         .. [{" uses syntax extension API version "}]
8023         .. user_extension.api_version .. [{" but markdown.lua "}]
8024         .. metadata.version .. [{" uses API version "}]

```

```

8025     .. metadata.user_extension_api_version
8026     .. [[, which is incompatible]])
8027
8028     assert(user_extension.grammar_version ~= nil,
8029           [[User-defined syntax extension "]] .. pathname
8030           .. [[ " does not specify mandatory field "grammar_version"]])
8031     assert(type(user_extension.grammar_version) == "number",
8032           [[User-defined syntax extension "]] .. pathname
8033           .. [[ " specifies field "grammar_version" of type "]]
8034           .. type(user_extension.grammar_version)
8035           .. [[ " but "number" was expected]])
8036     assert(user_extension.grammar_version == metadata.grammar_version,
8037           [[User-defined syntax extension "]] .. pathname
8038           .. [[ " uses grammar version "]] .. user_extension.grammar_version
8039           .. [[ but markdown.lua ]] .. metadata.version
8040           .. [[ uses grammar version ]] .. metadata.grammar_version
8041           .. [[, which is incompatible]])
8042
8043     assert(user_extension.finalize_grammar ~= nil,
8044           [[User-defined syntax extension "]] .. pathname
8045           .. [[ " does not specify mandatory "finalize_grammar" field]])
8046     assert(type(user_extension.finalize_grammar) == "function",
8047           [[User-defined syntax extension "]] .. pathname
8048           .. [[ " specifies field "finalize_grammar" of type "]]
8049           .. type(user_extension.finalize_grammar)
8050           .. [[ " but "function" was expected]])

```

Finally, cast the user-defined syntax extension to the internal format of user extensions used by the Markdown package (see Section 3.1.6.)

```

8051     local extension = {
8052       name = [[user-defined "]] .. pathname .. [[ " syntax extension]],
8053       extend_reader = user_extension.finalize_grammar,
8054       extend_writer = function() end,
8055     }
8056     return extension
8057   end)(user_extension_filename)
8058   table.insert(extensions, user_extension)
8059 end

```

Produce and return a conversion function from markdown to plain TeX.

```

8060   local writer = M.writer.new(options)
8061   local reader = M.reader.new(writer, options)
8062   local convert = reader.finalize_grammar(extensions)
8063
8064   return convert
8065 end
8066
8067 return M

```

3.1.8 Command-Line Implementation

The command-line implementation provides the actual conversion routine for the command-line interface described in Section 2.1.6.

```
8068
8069 local input
8070 if input_filename then
8071   local input_file = assert(io.open(input_filename, "r"),
8072     [[Could not open file ]] .. input_filename .. [[ for reading]])
8073   input = assert(input_file:read("*a"))
8074   assert(input_file:close())
8075 else
8076   input = assert(io.read("*a"))
8077 end
8078
```

First, ensure that the `options.cacheDir` directory exists.

```
8079 local lfs = require("lfs")
8080 if options.cacheDir and not lfs.isdir(options.cacheDir) then
8081   assert(lfs.mkdir(options["cacheDir"]))
8082 end
8083
8084 local ran_ok, kpse = pcall(require, "kpse")
8085 if ran_ok then kpse.set_program_name("luatex") end
8086 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.

```
8087 if metadata.version ~= md.metadata.version then
8088   warn("markdown-cli.lua " .. metadata.version .. " used with " ..
8089     "markdown.lua " .. md.metadata.version .. ".")
8090 end
8091 local convert = md.new(options)
8092 local output = convert(input)
8093
8094 if output_filename then
8095   local output_file = assert(io.open(output_filename, "w"),
8096     [[Could not open file ]] .. output_filename .. [[ for writing]])
8097   assert(output_file:write(output))
8098   assert(output_file:close())
8099 else
8100   assert(io.write(output))
8101 end
```

3.2 Plain T_EX Implementation

The plain T_EX implementation provides macros for the interfacing between T_EX and Lua and for the buffering of input text. These macros are then used to implement

the macros for the conversion from markdown to plain T_EX exposed by the plain T_EX interface (see Section 2.2).

3.2.1 Logging Facilities

```

8102 \ifx\markdownInfo\undefined
8103   \def\markdownInfo#1{%
8104     \immediate\write-1{(1.\the\inputlineno) markdown.tex info: #1.}}%
8105 \fi
8106 \ifx\markdownWarning\undefined
8107   \def\markdownWarning#1{%
8108     \immediate\write16{(1.\the\inputlineno) markdown.tex warning: #1}}%
8109 \fi
8110 \ifx\markdownError\undefined
8111   \def\markdownError#1#2{%
8112     \errhelp{#2.}%
8113     \errmessage{(1.\the\inputlineno) markdown.tex error: #1}}%
8114 \fi

```

3.2.2 Token Renderer Prototypes

The following definitions should be considered placeholder.

```

8115 \def\markdownRendererInterblockSeparatorPrototype{\par}%
8116 \def\markdownRendererHardLineBreakPrototype{\hfil\break}%
8117 \let\markdownRendererEllipsisPrototype\dots
8118 \def\markdownRendererNbspPrototype{~}%
8119 \def\markdownRendererLeftBracePrototype{\char`\{}%
8120 \def\markdownRendererRightBracePrototype{\char`\}%
8121 \def\markdownRendererDollarSignPrototype{\char`\$}%
8122 \def\markdownRendererPercentSignPrototype{\char`\}%
8123 \def\markdownRendererAmpersandPrototype{\&}%
8124 \def\markdownRendererUnderscorePrototype{\char`\_%
8125 \def\markdownRendererHashPrototype{\char`\#}%
8126 \def\markdownRendererCircumflexPrototype{\char`\^}%
8127 \def\markdownRendererBackslashPrototype{\char`\}%
8128 \def\markdownRendererTildePrototype{\char`\~}%
8129 \def\markdownRendererPipePrototype{|}%
8130 \def\markdownRendererCodeSpanPrototype#1{\tt#1}%
8131 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
8132 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
8133   \markdownInput{#3}}%
8134 \def\markdownRendererContentBlockOnlineImagePrototype{%
8135   \markdownRendererImage}%
8136 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
8137   \markdownRendererInputFencedCode{#3}{#2}}%
8138 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
8139 \def\markdownRendererUlBeginPrototype{}%

```

```

8140 \def\markdownRendererUlBeginTightPrototype{}%
8141 \def\markdownRendererUlItemPrototype{}%
8142 \def\markdownRendererUlItemEndPrototype{}%
8143 \def\markdownRendererUlEndPrototype{}%
8144 \def\markdownRendererUlEndTightPrototype{}%
8145 \def\markdownRendererOlBeginPrototype{}%
8146 \def\markdownRendererOlBeginTightPrototype{}%
8147 \def\markdownRendererFancyOlBeginPrototype#1#2{\markdownRendererOlBegin}%
8148 \def\markdownRendererFancyOlBeginTightPrototype#1#2{\markdownRendererOlBeginTight}%
8149 \def\markdownRendererOlItemPrototype{}%
8150 \def\markdownRendererOlItemWithNumberPrototype#1{}%
8151 \def\markdownRendererOlItemEndPrototype{}%
8152 \def\markdownRendererFancyOlItemPrototype{\markdownRendererOlItem}%
8153 \def\markdownRendererFancyOlItemWithNumberPrototype{\markdownRendererOlItemWithNumber}
8154 \def\markdownRendererFancyOlItemEndPrototype{}%
8155 \def\markdownRendererOlEndPrototype{}%
8156 \def\markdownRendererOlEndTightPrototype{}%
8157 \def\markdownRendererFancyOlEndPrototype{\markdownRendererOlEnd}%
8158 \def\markdownRendererFancyOlEndTightPrototype{\markdownRendererOlEndTight}%
8159 \def\markdownRendererDlBeginPrototype{}%
8160 \def\markdownRendererDlBeginTightPrototype{}%
8161 \def\markdownRendererDlItemPrototype#1{#1}%
8162 \def\markdownRendererDlItemEndPrototype{}%
8163 \def\markdownRendererDlDefinitionBeginPrototype{}%
8164 \def\markdownRendererDlDefinitionEndPrototype{\par}%
8165 \def\markdownRendererDlEndPrototype{}%
8166 \def\markdownRendererDlEndTightPrototype{}%
8167 \def\markdownRendererEmphasisPrototype#1{\it#1}%
8168 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
8169 \def\markdownRendererBlockQuoteBeginPrototype{\begingroup\it}%
8170 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
8171 \def\markdownRendererLineBlockBeginPrototype{\begingroup\parindent=0pt}%
8172 \def\markdownRendererLineBlockEndPrototype{\endgroup}%
8173 \def\markdownRendererInputVerbatimPrototype#1{%
8174   \par{\tt\input#1\relax{}}\par}%
8175 \def\markdownRendererInputFencedCodePrototype#1#2{%
8176   \markdownRendererInputVerbatim{#1}}%
8177 \def\markdownRendererHeadingOnePrototype#1{#1}%
8178 \def\markdownRendererHeadingTwoPrototype#1{#1}%
8179 \def\markdownRendererHeadingThreePrototype#1{#1}%
8180 \def\markdownRendererHeadingFourPrototype#1{#1}%
8181 \def\markdownRendererHeadingFivePrototype#1{#1}%
8182 \def\markdownRendererHeadingSixPrototype#1{#1}%
8183 \def\markdownRendererThematicBreakPrototype{}%
8184 \def\markdownRendererNotePrototype#1{#1}%
8185 \def\markdownRendererCitePrototype#1{}%
8186 \def\markdownRendererTextCitePrototype#1{}%

```

```

8187 \def\markdownRendererTickedBoxPrototype{[X]}%
8188 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
8189 \def\markdownRendererUntickedBoxPrototype{[ ]}%
8190 \def\markdownRendererStrikeThroughPrototype#1{#1}%
8191 \def\markdownRendererSuperscriptPrototype#1{#1}%
8192 \def\markdownRendererSubscriptPrototype#1{#1}%
8193 \def\markdownRendererDisplayMathPrototype#1{$$#1$$}%
8194 \def\markdownRendererInlineMathPrototype#1{$#1$}%
8195 \ExplSyntaxOn
8196 \cs_gset:Npn
8197   \markdownRendererHeaderAttributeContextBeginPrototype
8198   {
8199     \group_begin:
8200     \color_group_begin:
8201   }
8202 \cs_gset:Npn
8203   \markdownRendererHeaderAttributeContextEndPrototype
8204   {
8205     \color_group_end:
8206     \group_end:
8207   }
8208 \cs_gset_eq:NN
8209   \markdownRendererBracketedSpanAttributeContextBeginPrototype
8210   \markdownRendererHeaderAttributeContextBeginPrototype
8211 \cs_gset_eq:NN
8212   \markdownRendererBracketedSpanAttributeContextEndPrototype
8213   \markdownRendererHeaderAttributeContextEndPrototype
8214 \cs_gset_eq:NN
8215   \markdownRendererFencedDivAttributeContextBeginPrototype
8216   \markdownRendererHeaderAttributeContextBeginPrototype
8217 \cs_gset_eq:NN
8218   \markdownRendererFencedDivAttributeContextEndPrototype
8219   \markdownRendererHeaderAttributeContextEndPrototype
8220 \cs_gset_eq:NN
8221   \markdownRendererFencedCodeAttributeContextBeginPrototype
8222   \markdownRendererHeaderAttributeContextBeginPrototype
8223 \cs_gset_eq:NN
8224   \markdownRendererFencedCodeAttributeContextEndPrototype
8225   \markdownRendererHeaderAttributeContextEndPrototype
8226 \cs_gset:Npn
8227   \markdownRendererReplacementCharacterPrototype
8228   {
8229     % TODO: Replace with \codepoint_generate:nn in TeX Live 2023
8230     \sys_if_engine_pdftex:TF
8231     { ^^ef^^bf^^bd }
8232     { ^^fffd }
8233   }

```

```

8234 \ExplSyntaxOff
8235 \def\markdownRendererSectionBeginPrototype{}%
8236 \def\markdownRendererSectionEndPrototype{}%

```

3.2.2.1 Raw Attribute Renderer Prototypes 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.

```

8237 \ExplSyntaxOn
8238 \cs_new:Nn
8239   \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
8240   {
8241     \str_case:nn
8242       { #2 }
8243       {
8244         { md } { \markdownInput{#1} }
8245         { tex } { \markdownEscape{#1} \unskip }
8246       }
8247   }
8248 \cs_new:Nn
8249   \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
8250   {
8251     \str_case:nn
8252       { #2 }
8253       {
8254         { md } { \markdownInput{#1} }
8255         { tex } { \markdownEscape{#1} }
8256       }
8257   }
8258 \cs_gset:Npn
8259   \markdownRendererInputRawInlinePrototype#1#2
8260   {
8261     \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
8262       { #1 }
8263       { #2 }
8264   }
8265 \cs_gset:Npn
8266   \markdownRendererInputRawBlockPrototype#1#2
8267   {
8268     \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
8269       { #1 }
8270       { #2 }
8271   }
8272 \ExplSyntaxOff

```

3.2.2.2 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 .

```
8273 \ExplSyntaxOn
8274 \seq_new:N \g_@@_jekyll_data_datatypes_seq
8275 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
8276 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
8277 \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.

```
8278 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
8279 \cs_new:Nn \markdown_jekyll_data_push_address_segment:n
8280 {
8281   \seq_if_empty:NF
8282     \g_@@_jekyll_data_datatypes_seq
8283     {
8284       \seq_get_right:NN
8285       \g_@@_jekyll_data_datatypes_seq
8286       \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.

```
8287   \str_if_eq:NNTF
8288     \l_tmpa_tl
8289     \c_@@_jekyll_data_sequence_tl
8290     {
8291       \seq_put_right:Nn
8292       \g_@@_jekyll_data_wildcard_absolute_address_seq
8293       { * }
8294     }
```



```

8295     {
8296         \seq_put_right:Nn
8297         \g_@@_jekyll_data_wildcard_absolute_address_seq
8298         { #1 }
8299     }
8300 }
8301 }

```

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.

```

8302 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
8303 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
8304 \cs_new:Nn \markdown_jekyll_data_concatenate_address:NN
8305   {
8306     \seq_pop_left:NN #1 \l_tmpa_tl
8307     \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
8308     \seq_put_left:NV #1 \l_tmpa_tl
8309   }
8310 \cs_new:Nn \markdown_jekyll_data_update_address_tls:
8311   {
8312     \markdown_jekyll_data_concatenate_address:NN
8313     \g_@@_jekyll_data_wildcard_absolute_address_seq

```

```

8314     \g_@@_jekyll_data_wildcard_absolute_address_tl
8315     \seq_get_right:NN
8316     \g_@@_jekyll_data_wildcard_absolute_address_seq
8317     \g_@@_jekyll_data_wildcard_relative_address_tl
8318 }

```

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.

```

8319 \cs_new:Nn \markdown_jekyll_data_push:nN
8320 {
8321     \markdown_jekyll_data_push_address_segment:n
8322     { #1 }
8323     \seq_put_right:NV
8324     \g_@@_jekyll_data_datatypes_seq
8325     #2
8326     \markdown_jekyll_data_update_address_tls:
8327 }
8328 \cs_new:Nn \markdown_jekyll_data_pop:
8329 {
8330     \seq_pop_right:NN
8331     \g_@@_jekyll_data_wildcard_absolute_address_seq
8332     \l_tmpa_tl
8333     \seq_pop_right:NN
8334     \g_@@_jekyll_data_datatypes_seq
8335     \l_tmpa_tl
8336     \markdown_jekyll_data_update_address_tls:
8337 }

```

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.

```

8338 \cs_new:Nn \markdown_jekyll_data_set_keyval:nn
8339 {
8340     \keys_set_known:nn
8341     { markdown/jekyllData }
8342     { { #1 } = { #2 } }
8343 }
8344 \cs_generate_variant:Nn
8345     \markdown_jekyll_data_set_keyval:nn
8346     { Vn }
8347 \cs_new:Nn \markdown_jekyll_data_set_keyvals:nn
8348 {
8349     \markdown_jekyll_data_push:nN
8350     { #1 }
8351     \c_@@_jekyll_data_scalar_tl
8352     \markdown_jekyll_data_set_keyval:Vn
8353     \g_@@_jekyll_data_wildcard_absolute_address_tl
8354     { #2 }

```

```

8355     \markdown_jekyll_data_set_keyval:Vn
8356     \g_@@_jekyll_data_wildcard_relative_address_tl
8357     { #2 }
8358     \markdown_jekyll_data_pop:
8359   }

```

Finally, we will register our macros as token renderer prototypes to be able to react to the traversal of a YAML document.

```

8360 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
8361   \markdown_jekyll_data_push:nN
8362   { #1 }
8363   \c_@@_jekyll_data_sequence_tl
8364 }
8365 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
8366   \markdown_jekyll_data_push:nN
8367   { #1 }
8368   \c_@@_jekyll_data_mapping_tl
8369 }
8370 \def\markdownRendererJekyllDataSequenceEndPrototype{
8371   \markdown_jekyll_data_pop:
8372 }
8373 \def\markdownRendererJekyllDataMappingEndPrototype{
8374   \markdown_jekyll_data_pop:
8375 }
8376 \def\markdownRendererJekyllDataBooleanPrototype#1#2{
8377   \markdown_jekyll_data_set_keyvals:nn
8378   { #1 }
8379   { #2 }
8380 }
8381 \def\markdownRendererJekyllDataEmptyPrototype#1{}
8382 \def\markdownRendererJekyllDataNumberPrototype#1#2{
8383   \markdown_jekyll_data_set_keyvals:nn
8384   { #1 }
8385   { #2 }
8386 }
8387 \def\markdownRendererJekyllDataStringPrototype#1#2{
8388   \markdown_jekyll_data_set_keyvals:nn
8389   { #1 }
8390   { #2 }
8391 }
8392 \ExplSyntaxOff

```

3.2.3 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).

```

8393 \ExplSyntaxOn
8394 \tl_new:N \g_@@_formatted_lua_options_tl
8395 \cs_new:Nn \@@_format_lua_options:
8396 {
8397   \tl_gclear:N
8398   \g_@@_formatted_lua_options_tl
8399   \seq_map_function:NN
8400   \g_@@_lua_options_seq
8401   \@@_format_lua_option:n
8402 }
8403 \cs_new:Nn \@@_format_lua_option:n
8404 {
8405   \@@_typecheck_option:n
8406   { #1 }
8407   \@@_get_option_type:nN
8408   { #1 }
8409   \l_tmpa_tl
8410   \bool_case_true:nF
8411   {
8412     {
8413       \str_if_eq_p:VV
8414       \l_tmpa_tl
8415       \c_@@_option_type_boolean_tl ||
8416       \str_if_eq_p:VV
8417       \l_tmpa_tl
8418       \c_@@_option_type_number_tl ||
8419       \str_if_eq_p:VV
8420       \l_tmpa_tl
8421       \c_@@_option_type_counter_tl
8422     }
8423     {
8424       \@@_get_option_value:nN
8425       { #1 }
8426       \l_tmpa_tl
8427       \tl_gput_right:Nx
8428       \g_@@_formatted_lua_options_tl
8429       { #1~::~ \l_tmpa_tl ,~ }
8430     }
8431   }
8432   \str_if_eq_p:VV
8433   \l_tmpa_tl
8434   \c_@@_option_type_clist_tl
8435 }
8436 {
8437   \@@_get_option_value:nN
8438   { #1 }
8439   \l_tmpa_tl

```

```

8440         \tl_gput_right:Nx
8441         \g_@@_formatted_lua_options_tl
8442         { #1~::~\c_left_brace_str }
8443     \clist_map_inline:Vn
8444         \l_tmpa_tl
8445         {
8446             \tl_gput_right:Nx
8447             \g_@@_formatted_lua_options_tl
8448             { "##1" ,~ }
8449         }
8450     \tl_gput_right:Nx
8451     \g_@@_formatted_lua_options_tl
8452     { \c_right_brace_str ,~ }
8453 }
8454 }
8455 {
8456     \@@_get_option_value:nN
8457     { #1 }
8458     \l_tmpa_tl
8459     \tl_gput_right:Nx
8460     \g_@@_formatted_lua_options_tl
8461     { #1~::~ " \l_tmpa_tl " ,~ }
8462 }
8463 }
8464 \cs_generate_variant:Nn
8465   \clist_map_inline:nn
8466   { Vn }
8467 \let\markdownPrepareLuaOptions=\@@_format_lua_options:
8468 \def\markdownLuaOptions{{ \g_@@_formatted_lua_options_tl }}
8469 \ExplSyntaxOff

```

The `\markdownPrepare` macro contains the Lua code that is executed prior to any conversion from markdown to plain T_EX. It exposes the `convert` function for the use by any further Lua code.

```
8470 \def\markdownPrepare{%
```

First, ensure that the `cacheDir` directory exists.

```

8471   local lfs = require("lfs")
8472   local cacheDir = "\markdownOptionCacheDir"
8473   if not lfs.isdir(cacheDir) then
8474       assert(lfs.mkdir(cacheDir))
8475   end

```

Next, load the `markdown` module and create a converter function using the plain T_EX options, which were serialized to a Lua table via the `\markdownLuaOptions` macro.

```

8476   local md = require("markdown")
8477   local convert = md.new(\markdownLuaOptions)
8478 }%

```

3.2.4 Buffering Markdown Input

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

```
8479 \ExplSyntaxOn
8480 \cs_new:Nn
8481   \@@_if_option:nTF
8482   {
8483     \@@_get_option_type:nN
8484     { #1 }
8485     \l_tmpa_tl
8486     \str_if_eq:NNF
8487     \l_tmpa_tl
8488     \c_@@_option_type_boolean_tl
8489     {
8490       \msg_error:nxxx
8491       { @@ }
8492       { expected-boolean-option }
8493       { #1 }
8494       { \l_tmpa_tl }
8495     }
8496     \@@_get_option_value:nN
8497     { #1 }
8498     \l_tmpa_tl
8499     \str_if_eq:NNTF
8500     \l_tmpa_tl
8501     \c_@@_option_value_true_tl
8502     { #2 }
8503     { #3 }
8504   }
8505 \msg_new:nnn
8506   { @@ }
8507   { expected-boolean-option }
8508   {
8509     Option~#1~has~type~#2,~
8510     but~a~boolean~was~expected.
8511   }
8512 \let\markdownIfOption=\@@_if_option:nTF
8513 \ExplSyntaxOff
```

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.

```
8514 \csname newread\endcsname\markdownInputFileStream
8515 \csname newwrite\endcsname\markdownOutputFileStream
```

The `\markdownReadAndConvertTab` macro contains the tab character literal.

```

8516 \begingroup
8517   \catcode\^^I=12%
8518   \gdef\markdownReadAndConvertTab{^^I}%
8519 \endgroup

```

The `\markdownReadAndConvert` macro is largely a rewrite of the $\text{\LaTeX} 2_{\epsilon}$ `\filecontents` macro to plain \TeX .

```

8520 \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.

```

8521   \catcode\^^M=13%
8522   \catcode\^^I=13%
8523   \catcode|=0%
8524   \catcode\=12%
8525   |catcode@=14%
8526   |catcode|=12@
8527   |gdef|markdownReadAndConvert#1#2{@
8528     |begingroup@

```

If we are not reading markdown documents from the frozen cache, open the `inputTempFileName` file for writing.

```

8529     |markdownIfOption{frozenCache}{-}{@
8530       |immediate|openout|markdownOutputFileStream@
8531       |markdownOptionInputTempFileName|relax@
8532       |markdownInfo{Buffering markdown input into the temporary @
8533         input file "|markdownOptionInputTempFileName" and scanning @
8534         for the closing token sequence "#1"}@
8535     }@

```

Locally change the category of the special plain \TeX characters to *other* in order to prevent unwanted interpretation of the input. Change also the category of the space character, so that we can retrieve it unaltered.

```

8536     |def|do##1{|catcode`##1=12}|dospecials@
8537     |catcode`|=12@
8538     |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.

```

8539     |def|markdownReadAndConvertStripPercentSign##1{@
8540       |markdownIfOption{stripPercentSigns}{-}{@
8541         |if##1%@
8542         |expandafter|expandafter|expandafter@

```

```

8543         |markdownReadAndConvertProcessLine@
8544     |else@
8545         |expandafter|expandafter|expandafter@
8546         |markdownReadAndConvertProcessLine@
8547         |expandafter|expandafter|expandafter##1@
8548     |fi@
8549 }{@
8550     |expandafter@
8551     |markdownReadAndConvertProcessLine@
8552     |expandafter##1@
8553 }@
8554 }@

```

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.

```

8555     |def|markdownReadAndConvertProcessLine##1##2##3|relax{@

```

If we are not reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, store the line in the `inputTempFileName` file. If we are reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, gobble the line.

```

8556     |ifx|relax##3|relax@
8557         |markdownIfOption{frozenCache}{-}{@
8558         |immediate|write|markdownOutputFileStream{##1}@
8559     }@
8560     |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_EX, `\input` the result of the conversion, and expand the ending control sequence.

```

8561     |def^^M{@
8562         |markdownInfo{The ending token sequence was found}@
8563         |markdownIfOption{frozenCache}{-}{@
8564         |immediate|closeout|markdownOutputFileStream@
8565     }@
8566     |endgroup@
8567     |markdownInput{@
8568         |markdownOptionOutputDir@
8569         /|markdownOptionInputTempFileName@
8570     }@
8571     #2}@
8572     |fi@

```

Repeat with the next line.

```

8573     ^^M}@

```


Make the tab character active at expansion time and make it expand to a literal tab character.

```
8574 |catcode`\^^I=13@
8575 |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.

```
8576 |catcode`\^^M=13@
8577 |def^^M##1^^M{@
8578 |def^^M###1^^M{@
8579 |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
8580 ^^M}@
8581 ^^M}@
```

Reset the character categories back to the former state.

```
8582 |endgroup
```

The following two sections of the implementation have been deprecated and will be removed in Markdown 3.0.0. The code that corresponds to `\markdownMode` value of `3` will be the only implementation.

```
8583 \ExplSyntaxOn
8584 \int_compare:nT
8585 { \markdownMode = 3 }
8586 {
8587 \markdownInfo{Using~mode~3:~The~lt3luabridge~package}
8588 \file_input:n { lt3luabridge.tex }
8589 \cs_new:Npn
8590 \markdownLuaExecute
8591 { \luabridgeExecute }
8592 }
8593 \ExplSyntaxOff
```

3.2.5 Lua Shell Escape Bridge

The following \TeX code is intended for \TeX engines that do not provide direct access to Lua, but expose the shell of the operating system. This corresponds to the `\markdownMode` values of `0` and `1`.

The `\markdownLuaExecute` macro defined here and in Section 3.2.6 are meant to be indistinguishable to the remaining code.

The package assumes that although the user is not using the Lua \TeX engine, their \TeX distribution contains it, and uses shell access to produce and execute Lua scripts using the \TeX Lua interpreter [1, Section 4.1.1].

```
8594 \ifnum\markdownMode<2\relax
8595 \ifnum\markdownMode=0\relax
8596 \markdownWarning{Using mode 0: Shell escape via write18
```

```

8597             (deprecated, to be removed in Markdown 3.0.0)}%
8598 \else
8599   \markdownWarning{Using mode 1: Shell escape via os.execute
8600             (deprecated, to be removed in Markdown 3.0.0)}%
8601 \fi

```

The `\markdownExecuteShellEscape` macro contains the numeric value indicating whether the shell access is enabled (1), disabled (0), or restricted (2).

Inherit the value of the `\pdfshellescape` (LuaTeX, PdfTeX) or the `\shellescape` (XeTeX) commands. If neither of these commands is defined and Lua is available, attempt to access the `status.shell_escape` configuration item.

If you cannot detect, whether the shell access is enabled, act as if it were.

```

8602 \ifx\pdfshellescape\undefined
8603   \ifx\shellescape\undefined
8604     \ifnum\markdownMode=0\relax
8605       \def\markdownExecuteShellEscape{1}%
8606     \else
8607       \def\markdownExecuteShellEscape{%
8608         \directlua{tex.sprint(status.shell_escape or "1")}}%
8609     \fi
8610   \else
8611     \let\markdownExecuteShellEscape\shellescape
8612   \fi
8613 \else
8614   \let\markdownExecuteShellEscape\pdfshellescape
8615 \fi

```

The `\markdownExecuteDirect` macro executes the code it has received as its first argument by writing it to the output file stream 18, if Lua is unavailable, or by using the Lua `os.execute` method otherwise.

```

8616 \ifnum\markdownMode=0\relax
8617   \def\markdownExecuteDirect#1{\immediate\write18{#1}}%
8618 \else
8619   \def\markdownExecuteDirect#1{%
8620     \directlua{os.execute("\luaescapestring{#1}")}}%
8621 \fi

```

The `\markdownExecute` macro is a wrapper on top of `\markdownExecuteDirect` that checks the value of `\markdownExecuteShellEscape` and prints an error message if the shell is inaccessible.

```

8622 \def\markdownExecute#1{%
8623   \ifnum\markdownExecuteShellEscape=1\relax
8624     \markdownExecuteDirect{#1}%
8625   \else
8626     \markdownError{I can not access the shell}{Either run the TeX
8627       compiler with the --shell-escape or the --enable-write18 flag,
8628       or set shell_escape=t in the texmf.cnf file}%

```

```
8629 \fi}%
```

The `\markdownLuaExecute` macro executes the Lua code it has received as its first argument. The Lua code may not directly interact with the TeX engine, but it can use the `print` function in the same manner it would use the `tex.print` method.

```
8630 \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.

```
8631 \catcode`\|=0%
8632 \catcode`\|=12%
8633 |gdef|markdownLuaExecute#1{%
```

Create the file `helperScriptFileName` and fill it with the input Lua code prepended with `kpathsea` initialization, so that Lua modules from the TeX distribution are available.

```
8634 |immediate|openout|markdownOutputFileStream=%
8635 |markdownOptionHelperScriptFileName
8636 |markdownInfo{Writing a helper Lua script to the file
8637 "|markdownOptionHelperScriptFileName"%
8638 |immediate|write|markdownOutputFileStream{%
8639 local ran_ok, error = pcall(function()
8640 local ran_ok, kpse = pcall(require, "kpse")
8641 if ran_ok then kpse.set_program_name("luatex") end
8642 #1
8643 end)
```

If there was an error, use the file `errorTempFileName` to store the error message.

```
8644 if not ran_ok then
8645 local file = io.open("%
8646 |markdownOptionOutputDir
8647 /|markdownOptionErrorTempFileName", "w")
8648 if file then
8649 file:write(error .. "\n")
8650 file:close()
8651 end
8652 print('\|markdownError{An error was encountered while executing
8653 Lua code}{For further clues, examine the file
8654 "|markdownOptionOutputDir
8655 /|markdownOptionErrorTempFileName}')
8656 end}%
8657 |immediate|closeout|markdownOutputFileStream
```

Execute the generated `helperScriptFileName` Lua script using the TeX Lua binary and store the output in the `outputTempFileName` file.

```
8658 |markdownInfo{Executing a helper Lua script from the file
8659 "|markdownOptionHelperScriptFileName" and storing the result in the
8660 file "|markdownOptionOutputTempFileName"%
```

```

8661     |markdownExecute{texlua "|markdownOptionOutputDir
8662     /|markdownOptionHelperScriptFileName" > %
8663     "|markdownOptionOutputDir
8664     /|markdownOptionOutputTempFileName"}%

```

`\input` the generated `outputTempFileName` file.

```

8665     |input|markdownOptionOutputTempFileName|relax}%
8666 |endgroup

```

3.2.6 Direct Lua Access

The following `TEX` code is intended for `TEX` engines that provide direct access to Lua (Lua`TEX`). The macro `\markdownLuaExecute` defined here and in Section 3.2.5 are meant to be indistinguishable to the remaining code. This corresponds to the `\markdownMode` value of 2.

```

8667 \fi
8668 \ifnum\markdownMode=2\relax
8669   \markdownWarning{Using mode 2: Direct Lua access
8670                   (deprecated, to be removed in Markdown 3.0.0)}%

```

The direct Lua access version of the `\markdownLuaExecute` macro is defined in terms of the `\directlua` primitive. The `print` function is set as an alias to the `tex.print` method in order to mimic the behaviour of the `\markdownLuaExecute` definition from Section 3.2.5,

```

8671 \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.

```

8672   \catcode`|=0%
8673   \catcode`\|=12%
8674   |gdef|markdownLuaExecute#1{%
8675     |directlua{%
8676       local function print(input)
8677         local output = {}
8678         for line in input:gmatch("[^\r\n]+") do
8679           table.insert(output, line)
8680         end
8681         tex.print(output)
8682       end
8683       #1
8684     }%
8685   }%
8686 |endgroup
8687 \fi

```

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 TeX.

```
8688 \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.

```
8689 \catcode\|=0%
8690 \catcode\|=12%
8691 \catcode\&=6%
8692 |gdef|markdownInput#1{%
```

Change the category code of the percent sign (%) to other, so that a user of the `hybrid` Lua option or a malevolent actor can't produce TeX comments in the plain TeX output of the Markdown package.

```
8693 |begingroup
8694 |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.

```
8695 |catcode\#=12
```

If we are reading from the frozen cache, input it, expand the corresponding `\markdownFrozenCache<number>` macro, and increment `frozenCacheCounter`.

```
8696 |markdownIfOption{frozenCache}{%
8697 |ifnum|markdownOptionFrozenCacheCounter=0|relax
8698 |markdownInfo{Reading frozen cache from
8699 |"|markdownOptionFrozenCacheFileName"|}%
8700 |input|markdownOptionFrozenCacheFileName|relax
8701 |fi
8702 |markdownInfo{Including markdown document number
8703 |"|the|markdownOptionFrozenCacheCounter" from frozen cache}%
8704 |csname markdownFrozenCache|the|markdownOptionFrozenCacheCounter|endcsname
8705 |global|advance|markdownOptionFrozenCacheCounter by 1|relax
8706 }{%
8707 |markdownInfo{Including markdown document "&1"}%
```

Attempt to open the markdown document to record it in the `.log` and `.fls` files. This allows external programs such as L^AT_EX Mk to track changes to the markdown document.

```
8708 |openin|markdownInputFileStream&1
8709 |closein|markdownInputFileStream
8710 |markdownPrepareLuaOptions
8711 |markdownLuaExecute{%
```

```

8712     |markdownPrepare
8713     local file = assert(io.open("&1", "r"),
8714         [[Could not open file "&1" for reading]])
8715     local input = assert(file:read("*a"))
8716     assert(file:close())

```

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.

```

8717     print(convert(input))}%

```

In case we were finalizing the frozen cache, increment `frozenCacheCounter`.

```

8718     |global|advance|markdownOptionFrozenCacheCounter by 1|relax
8719     }%
8720     |endgroup
8721     }%
8722 |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.

```

8723 \gdef\markdownEscape#1{%
8724   \catcode`\%=14\relax
8725   \catcode`\#=6\relax
8726   \input #1\relax
8727   \catcode`\%=12\relax
8728   \catcode`\#=12\relax
8729 }%

```

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.

```

8730 \def\markdownVersionSpace{ }%
8731 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
8732   \markdownVersion\markdownVersionSpace markdown renderer]%

```

Use reflection to define the `renderers` and `rendererPrototypes` keys of `\markdownSetup` as well as the keys that correspond to Lua options.

```

8733 \ExplSyntaxOn
8734 \@@_latex_define_renderers:
8735 \@@_latex_define_renderer_prototypes:
8736 \ExplSyntaxOff

```

3.3.1 Logging Facilities

The \LaTeX implementation redefines the plain \TeX logging macros (see Section 3.2.1) to use the \LaTeX `\PackageInfo`, `\PackageWarning`, and `\PackageError` macros.

3.3.2 Typesetting Markdown

The `\markdownInputPlainTeX` macro is used to store the original plain \TeX implementation of the `\markdownInput` macro. The `\markdownInput` is then redefined to accept an optional argument with options recognized by the \LaTeX interface (see Section 2.3.2).

```
8737 \let\markdownInputPlainTeX\markdownInput
8738 \renewcommand\markdownInput[2][{}]{%
8739   \begingroup
8740     \markdownSetup{#1}%
8741     \markdownInputPlainTeX{#2}%
8742   \endgroup}%
```

The `markdown`, and `markdown*` \LaTeX environments are implemented using the `\markdownReadAndConvert` macro.

```
8743 \renewenvironment{markdown}{%
8744   \markdownReadAndConvert@markdown{}}{%
8745   \markdownEnd}%
8746 \renewenvironment{markdown*}[1]{%
8747   \markdownSetup{#1}%
8748   \markdownReadAndConvert@markdown*}{%
8749   \markdownEnd}%
8750 \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*.

```
8751 \catcode`\|=0\catcode`\<=1\catcode`\>=2%
8752 \catcode`\|=12\catcode`\{=12\catcode`\}=12%
8753 |gdef|markdownReadAndConvert@markdown#1<%
8754   |markdownReadAndConvert<\end{markdown#1}>%
8755   <|end<markdown#1>>>%
8756 |endgroup
```

3.3.2.1 \LaTeX Themes This section implements the theme-loading mechanism and the example themes provided with the Markdown package.

```
8757 \ExplSyntaxOn
```

To keep track of our current place when packages themes have been nested, we will maintain the `\g_@@_latex_themes_seq` stack of theme names.

```

8758 \newcommand\markdownLaTeXThemeName{}
8759 \seq_new:N \g_@@_latex_themes_seq
8760 \seq_gput_right:NV
8761   \g_@@_latex_themes_seq
8762   \markdownLaTeXThemeName
8763 \newcommand\markdownLaTeXThemeLoad[2]{
8764   \def\@tempa{%
8765     \def\markdownLaTeXThemeName{#2}
8766     \seq_gput_right:NV
8767       \g_@@_latex_themes_seq
8768       \markdownLaTeXThemeName
8769     \RequirePackage{#1}
8770     \seq_pop_right:NN
8771       \g_@@_latex_themes_seq
8772       \l_tmpa_tl
8773     \seq_get_right:NN
8774       \g_@@_latex_themes_seq
8775       \l_tmpa_tl
8776     \exp_args:NNV
8777       \def
8778         \markdownLaTeXThemeName
8779         \l_tmpa_tl}
8780   \ifmarkdownLaTeXLoaded
8781     \@tempa
8782   \else
8783     \exp_args:No
8784       \AtEndOfPackage
8785       { \@tempa }
8786   \fi}
8787 \ExplSyntaxOff

```

The `witiko/dot` theme enables the `fencedCode` Lua option:

```
8788 \markdownSetup{fencedCode}%
```

We load the `ifthen` and `grffile` packages, see also Section 1.1.3:

```
8789 \RequirePackage{ifthen,grffile}
```

We store the previous definition of the fenced code token renderer prototype:

```
8790 \let\markdown@witiko@dot@oldRendererInputFencedCodePrototype
8791   \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_EX option is disabled and the code block has not been previously typeset:

```

8792 \renewcommand\markdownRendererInputFencedCodePrototype[2]{%
8793   \def\next##1 ##2\relax{%
8794     \ifthenelse{equal{##1}{dot}}{%

```



```

8795     \markdownIfOption{frozenCache}{-}{%
8796         \immediate\write18{%
8797             if ! test -e #1.pdf.source || ! diff #1 #1.pdf.source;
8798             then
8799                 dot -Tpdf -o #1.pdf #1;
8800                 cp #1 #1.pdf.source;
8801             fi}}%

```

We include the typeset image using the image token renderer:

```

8802     \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:

```

8803     }{%
8804         \markdown@witiko@dot@oldRendererInputFencedCodePrototype{#1}{#2}%
8805     }%
8806 }%
8807 \next#2 \relax}%

```

The `witiko/graphicx/http` theme stores the previous definition of the image token renderer prototype:

```

8808 \let\markdown@witiko@graphicx@http@oldRendererImagePrototype
8809 \markdownRendererImagePrototype

```

We load the catchfile and grffile packages, see also Section 1.1.3:

```

8810 \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.

```

8811 \newcount\markdown@witiko@graphicx@http@counter
8812 \markdown@witiko@graphicx@http@counter=0
8813 \newcommand\markdown@witiko@graphicx@http@filename{%
8814     \markdownOptionCacheDir/witiko_graphicx_http%
8815     .\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.

```

8816 \newcommand\markdown@witiko@graphicx@http@download[2]{%
8817     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:

```

8818 \begingroup
8819 \catcode`\%=12
8820 \catcode`\^^A=14

```

We redefine the image token renderer prototype, so that it tries to download an online image.

```
8821 \global\def\markdownRendererImagePrototype#1#2#3#4{^^A
8822   \begingroup
8823     \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:

```
8824     \markdownIfOption{frozenCache}{^^A
8825       \immediate\write18{^^A
8826         mkdir -p "\markdownOptionCacheDir";
8827         if printf '%s' "#3" | grep -q -E '^https?:';
8828         then
```

The image will be downloaded to the pathname `cacheDir/⟨the MD5 digest of the image URL⟩.⟨the suffix of the image URL⟩`:

```
8829         OUTPUT_PREFIX="\markdownOptionCacheDir";
8830         OUTPUT_BODY="$(printf '%s' '#3' | md5sum | cut -d' ' -f1)";
8831         OUTPUT_SUFFIX="$(printf '%s' '#3' | sed 's/.*[.]//')";
8832         OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";
```

The image will be downloaded only if it has not already been downloaded:

```
8833         if ! [ -e "$OUTPUT" ];
8834         then
8835           \markdown@witiko@graphicx@http@download{'#3'}{"$OUTPUT"};
8836           printf '%s' "$OUTPUT" > "\filename";
8837         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:

```
8838         else
8839           printf '%s' '#3' > "\filename";
8840         fi}}^^A
```

We load the pathname of the downloaded image and we typeset the image using the previous definition of the image renderer prototype:

```
8841     \CatchFileDef{\filename}{\filename}{\endlinechar=-1}^^A
8842     \markdown@witiko@graphicx@http@oldRendererImagePrototype^^A
8843     {#1}{#2}{\filename}{#4}^^A
8844   \endgroup
8845   \global\advance\markdown@witiko@graphicx@http@counter by 1\relax}^^A
8846 \endgroup
```

The `witiko/tilde` theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```
8847 \renewcommand\markdownRendererTildePrototype{~}%
```

3.3.3 Options

The supplied package options are processed using the `\markdownSetup` macro.

```
8848 \DeclareOption*{%
8849   \expandafter\markdownSetup\expandafter{\CurrentOption}}%
8850 \ProcessOptions\relax
```

After processing the options, activate the `jekyllDataRenderes`, `renderers`, `rendererPrototypes`, and `code` keys.

```
8851 \ExplSyntaxOn
8852 \keys_define:nn
8853   { markdown/latex-options }
8854   {
8855     renderers .code:n = {
8856       \keys_set:nn
8857         { markdown/latex-options/renderers }
8858         { #1 }
8859     },
8860   }
8861 \@@_with_various_cases:nn
8862   { rendererPrototypes }
8863   {
8864     \keys_define:nn
8865       { markdown/latex-options }
8866       {
8867         #1 .code:n = {
8868           \keys_set:nn
8869             { markdown/latex-options/renderer-prototypes }
8870             { ##1 }
8871         },
8872       }
8873   }
```

The `code` key is used to immediately expand and execute code, which can be especially useful in \LaTeX setup snippets.

```
8874 \keys_define:nn
8875   { markdown/latex-options }
8876   {
8877     code .code:n = { #1 },
8878   }
```

The `jekyllDataRenderers` key can be used as a syntactic sugar for setting the `markdown/jekyllData` key-values (see Section 2.2.4.1) without using the `expl3` language.

```
8879 \@@_with_various_cases:nn
8880   { jekyllDataRenderers }
8881   {
8882     \keys_define:nn
```

```

8883     { markdown/latex-options }
8884     {
8885         #1 .code:n = {
8886             \tl_set:Nn
8887             \l_tmpa_tl
8888             { ##1 }

```

To ensure that keys containing forward slashes get passed correctly, we replace all forward slashes in the nput 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.

```

8889         \tl_replace_all:NnV
8890         \l_tmpa_tl
8891         { / }
8892         \c_backslash_str
8893         \keys_set:nV
8894         { markdown/latex-options/jekyll-data-renderers }
8895         \l_tmpa_tl
8896     },
8897 }
8898 }
8899 \keys_define:nn
8900 { markdown/latex-options/jekyll-data-renderers }
8901 {
8902     unknown .code:n = {
8903         \tl_set_eq:NN
8904         \l_tmpa_tl
8905         \l_keys_key_str
8906         \tl_replace_all:NVn
8907         \l_tmpa_tl
8908         \c_backslash_str
8909         { / }
8910         \tl_put_right:Nn
8911         \l_tmpa_tl
8912         {
8913             .code:n = { #1 }
8914         }
8915         \keys_define:nV
8916         { markdown/jekyllData }
8917         \l_tmpa_tl
8918     }
8919 }
8920 \cs_generate_variant:Nn
8921 \keys_define:nn
8922 { nV }
8923 \cs_generate_variant:Nn

```

```

8924 \tl_replace_all:Nnn
8925 { NVn }
8926 \cs_generate_variant:Nn
8927 \tl_replace_all:Nnn
8928 { NnV }
8929 \ExplSyntaxOff

```

3.3.4 Token Renderer Prototypes

The following configuration should be considered placeholder. If the `plain` package option has been enabled (see Section 2.3.2.1), none of it will take effect.

```

8930 \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.

```

8931 \@ifclassloaded{beamer}{}{%
8932 \markdownIfOption{tightLists}{\RequirePackage{paralist}}{}%
8933 \markdownIfOption{fancyLists}{\RequirePackage{paralist}}{}%
8934 }

```

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.

```

8935 \ExplSyntaxOn
8936 \@ifpackageloaded{paralist}{
8937 \tl_new:N
8938 \l_@@_latex_fancy_list_item_label_number_style_tl
8939 \tl_new:N
8940 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
8941 \cs_new:Nn
8942 \@@_latex_fancy_list_item_label_number:nn
8943 {
8944 \str_case:nn
8945 { #1 }
8946 {
8947 { Decimal } { #2 }
8948 { LowerRoman } { \int_to_roman:n { #2 } }
8949 { UpperRoman } { \int_to_Roman:n { #2 } }
8950 { LowerAlpha } { \int_to_alph:n { #2 } }
8951 { UpperAlpha } { \int_to_alph:n { #2 } }
8952 }
8953 }
8954 \cs_new:Nn
8955 \@@_latex_fancy_list_item_label_delimiter:n
8956 {
8957 \str_case:nn
8958 { #1 }

```

```

8959     {
8960     { Default } { . }
8961     { OneParen } { ) }
8962     { Period } { . }
8963     }
8964   }
8965   \cs_new:Nn
8966     \@@_latex_fancy_list_item_label:nnn
8967     {
8968     \@@_latex_fancy_list_item_label_number:nn
8969     { #1 }
8970     { #3 }
8971     \@@_latex_fancy_list_item_label_delimiter:n
8972     { #2 }
8973   }
8974   \cs_new:Nn
8975     \@@_latex_paralist_style:nn
8976     {
8977     \str_case:nn
8978       { #1 }
8979       {
8980         { Decimal } { 1 }
8981         { LowerRoman } { i }
8982         { UpperRoman } { I }
8983         { LowerAlpha } { a }
8984         { UpperAlpha } { A }
8985       }
8986     \@@_latex_fancy_list_item_label_delimiter:n
8987     { #2 }
8988   }
8989   \markdownSetup{rendererPrototypes={
8990     ulBeginTight = {\begin{compactitem}},
8991     ulEndTight = {\end{compactitem}},
8992     fancyOlBegin = {
8993       \group_begin:
8994       \tl_set:Nn
8995         \l_@@_latex_fancy_list_item_label_number_style_tl
8996         { #1 }
8997       \tl_set:Nn
8998         \l_@@_latex_fancy_list_item_label_delimiter_style_tl
8999         { #2 }
9000       \tl_set:Nn
9001         \l_tmpa_tl
9002         { \begin{enumerate}[ ] }
9003       \tl_put_right:Nx
9004         \l_tmpa_tl
9005         { \@@_latex_paralist_style:nn { #1 } { #2 } }

```

```

9006     \tl_put_right:Nn
9007         \l_tmpa_tl
9008         { ] }
9009     \l_tmpa_tl
9010 },
9011 fancyOlEnd = {
9012     \end{enumerate}
9013     \group_end:
9014 },
9015 olBeginTight = {\begin{compactenum}},
9016 olEndTight = {\end{compactenum}},
9017 fancyOlBeginTight = {
9018     \group_begin:
9019     \tl_set:Nn
9020         \l_@@_latex_fancy_list_item_label_number_style_tl
9021         { #1 }
9022     \tl_set:Nn
9023         \l_@@_latex_fancy_list_item_label_delimiter_style_tl
9024         { #2 }
9025     \tl_set:Nn
9026         \l_tmpa_tl
9027         { \begin{compactenum}[ ] }
9028     \tl_put_right:Nx
9029         \l_tmpa_tl
9030         { \@@_latex_paralist_style:nn { #1 } { #2 } }
9031     \tl_put_right:Nn
9032         \l_tmpa_tl
9033         { ] }
9034     \l_tmpa_tl
9035 },
9036 fancyOlEndTight = {
9037     \end{compactenum}
9038     \group_end:
9039 },
9040 fancyOlItemWithNumber = {
9041     \item
9042     [
9043         \@@_latex_fancy_list_item_label:VVn
9044         \l_@@_latex_fancy_list_item_label_number_style_tl
9045         \l_@@_latex_fancy_list_item_label_delimiter_style_tl
9046         { #1 }
9047     ]
9048 },
9049 dlBeginTight = {\begin{compactdesc}},
9050 dlEndTight = {\end{compactdesc}}}}
9051 \cs_generate_variant:Nn
9052     \@@_latex_fancy_list_item_label:nnn

```

```

9053     { VVn }
9054 }{
9055   \markdownSetup{rendererPrototypes={
9056     ulBeginTight = {\markdownRendererUlBegin},
9057     ulEndTight = {\markdownRendererUlEnd},
9058     fancyOlBegin = {\markdownRendererOlBegin},
9059     fancyOlEnd = {\markdownRendererOlEnd},
9060     olBeginTight = {\markdownRendererOlBegin},
9061     olEndTight = {\markdownRendererOlEnd},
9062     fancyOlBeginTight = {\markdownRendererOlBegin},
9063     fancyOlEndTight = {\markdownRendererOlEnd},
9064     dlBeginTight = {\markdownRendererDlBegin},
9065     dlEndTight = {\markdownRendererDlEnd}}
9066 }
9067 \ExplSyntaxOff
9068 \RequirePackage{amsmath}

```

Unless the unicode-math package has been loaded, load the amssymb package with symbols to be used for tickboxes.

```

9069 \@ifpackageloaded{unicode-math}{
9070   \markdownSetup{rendererPrototypes={
9071     untickedBox = {\$mdlgwhtsquare$},
9072   }}
9073 }{
9074   \RequirePackage{amssymb}
9075   \markdownSetup{rendererPrototypes={
9076     untickedBox = {\$square$},
9077   }}
9078 }
9079 \RequirePackage{csvsimple}
9080 \RequirePackage{fancyvrb}
9081 \RequirePackage{graphicx}
9082 \markdownSetup{rendererPrototypes={
9083   hardLineBreak = {\},
9084   leftBrace = {\textbraceleft},
9085   rightBrace = {\textbraceright},
9086   dollarSign = {\textdollar},
9087   underscore = {\textunderscore},
9088   circumflex = {\textasciicircum},
9089   backslash = {\textbackslash},
9090   tilde = {\textasciitilde},
9091   pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by \TeX during the typesetting. Therefore, even if we don't know whether a span of text is

part of math formula or not when we are parsing markdown,⁸ 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.

```

9092 codeSpan = {%
9093   \ifmmode
9094     \text{#1}%
9095   \else
9096     \texttt{#1}%
9097   \fi
9098   }}
9099 \ExplSyntaxOn
9100 \markdownSetup{
9101   rendererPrototypes = {
9102     contentBlock = {
9103       \str_case:nnF
9104         { #1 }
9105         {
9106           { csv }
9107           {
9108             \begin{table}
9109               \begin{center}
9110                 \csvautotabular{#3}
9111               \end{center}
9112             \tl_if_empty:nF
9113               { #4 }
9114               { \caption{#4} }
9115             \end{table}
9116           }
9117           { tex } { \markdownEscape{#3} }
9118         }
9119         { \markdownInput{#3} }
9120       },
9121     },
9122   }
9123 \ExplSyntaxOff
9124 \markdownSetup{rendererPrototypes={
9125   image = {%
9126     \begin{figure}%
9127       \begin{center}%
9128         \includegraphics{#3}%
9129       \end{center}%
9130     \ifx\empty#4\empty\else

```

⁸This 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.

```

9131     \caption{#4}%
9132     \fi
9133   \end{figure}},
9134   ulBegin = {\begin{itemize}},
9135   ulEnd = {\end{itemize}},
9136   olBegin = {\begin{enumerate}},
9137   olItem = {\item{}},
9138   olItemWithNumber = {\item[#1.]},
9139   olEnd = {\end{enumerate}},
9140   dlBegin = {\begin{description}},
9141   dlItem = {\item[#1]},
9142   dlEnd = {\end{description}},
9143   emphasis = {\emph{#1}},
9144   tickedBox = {\$\boxtimes$},
9145   halfTickedBox = {\$\boxdot$},

```

If identifier attributes appear at the beginning of a section, we make the next heading produce the `\label` macro.

```

9146   headerAttributeContextBegin = {%
9147     \markdownSetup{
9148       rendererPrototypes = {
9149         attributeIdentifier = {%
9150           \begingroup
9151           \def\next###1{%
9152             \def####1#####1{%
9153               \endgroup
9154               ###1{#####1}%
9155               \label{##1}%
9156             }%
9157           }%
9158           \next\markdownRendererHeadingOne
9159           \next\markdownRendererHeadingTwo
9160           \next\markdownRendererHeadingThree
9161           \next\markdownRendererHeadingFour
9162           \next\markdownRendererHeadingFive
9163           \next\markdownRendererHeadingSix
9164         },
9165       },
9166     }%
9167   },
9168   headerAttributeContextEnd = {},
9169   superscript = {\textsuperscript{#1}},
9170   subscript = {\textsubscript{#1}},
9171   displayMath = {\begin{displaymath}#1\end{displaymath}},
9172   inlineMath = {\begin{math}#1\end{math}},
9173   blockQuoteBegin = {\begin{quotation}},
9174   blockQuoteEnd = {\end{quotation}},

```

```

9175 inputVerbatim = {\VerbatimInput{#1}},
9176 thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
9177 note = {\footnote{#1}}}}

```

3.3.4.1 Fenced Code When no infostring has been specified, default to the indented code block renderer.

```

9178 \RequirePackage{ltxcmds}
9179 \ExplSyntaxOn
9180 \cs_gset:Npn
9181   \markdownRendererInputFencedCodePrototype#1#2
9182   {
9183     \tl_if_empty:nTF
9184       { #2 }
9185     { \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.

```

9186     {
9187       \regex_extract_once:nnN
9188         { \w* }
9189         { #2 }
9190         \l_tmpa_seq
9191       \seq_pop_left:NN
9192         \l_tmpa_seq
9193         \l_tmpa_tl

```

When the minted package is loaded, use it for syntax highlighting.

```

9194     \ltx@ifpackageloaded
9195       { minted }
9196     {
9197       \catcode`\#=6\relax
9198       \exp_args:NV
9199         \inputminted
9200         \l_tmpa_tl
9201         { #1 }
9202       \catcode`\#=12\relax
9203     }
9204     {

```

When the listings package is loaded, use it for syntax highlighting.

```

9205       \ltx@ifpackageloaded
9206         { listings }
9207         { \lstinputlisting[language=\l_tmpa_tl]{#1} }

```

When neither the listings package nor the minted package is loaded, act as though no infostring were given.

```

9208         { \markdownRendererInputFencedCode{#1}{ } }
9209     }

```

```

9210     }
9211   }
9212 \ExplSyntaxOff

Support the nesting of strong emphasis.

9213 \ExplSyntaxOn
9214 \def\markdownLATEXStrongEmphasis#1{%
9215   \str_if_in:NnTF
9216     \f@series
9217     { b }
9218     { \textnormal{#1} }
9219     { \textbf{#1} }
9220 }
9221 \ExplSyntaxOff
9222 \markdownSetup{rendererPrototypes={strongEmphasis={%
9223   \protect\markdownLATEXStrongEmphasis{#1}}}}

```

Support L^AT_EX document classes that do not provide chapters.

```

9224 \@ifundefined{chapter}{%
9225   \markdownSetup{rendererPrototypes = {
9226     headingOne = {\section{#1}},
9227     headingTwo = {\subsection{#1}},
9228     headingThree = {\subsubsection{#1}},
9229     headingFour = {\paragraph{#1}\leavevmode},
9230     headingFive = {\subparagraph{#1}\leavevmode}}}
9231 }{%
9232   \markdownSetup{rendererPrototypes = {
9233     headingOne = {\chapter{#1}},
9234     headingTwo = {\section{#1}},
9235     headingThree = {\subsection{#1}},
9236     headingFour = {\subsubsection{#1}},
9237     headingFive = {\paragraph{#1}\leavevmode},
9238     headingSix = {\subparagraph{#1}\leavevmode}}}
9239 }%

```

3.3.4.2 Tickboxes If the `taskLists` option is enabled, we will hide bullets in unordered list items with tickboxes.

```

9240 \markdownSetup{
9241   rendererPrototypes = {
9242     ulItem = {%
9243       \futurelet\markdownLaTeXCheckbox\markdownLaTeXULItem
9244     },
9245   },
9246 }
9247 \def\markdownLaTeXULItem{%
9248   \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
9249     \item[\markdownLaTeXCheckbox]%

```

```

9250     \expandafter\@gobble
9251   \else
9252     \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
9253       \item[\markdownLaTeXCheckbox]%
9254       \expandafter\expandafter\expandafter\@gobble
9255     \else
9256       \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
9257         \item[\markdownLaTeXCheckbox]%
9258         \expandafter\expandafter\expandafter\expandafter
9259         \expandafter\expandafter\expandafter\@gobble
9260       \else
9261         \item{}%
9262       \fi
9263     \fi
9264   \fi
9265 }

```

3.3.4.3 HTML elements If the `html` option is enabled and we are using `TeX4ht`⁹, we will pass HTML elements to the output HTML document unchanged.

```

9266 \@ifundefined{HCode}{}{
9267   \markdownSetup{
9268     rendererPrototypes = {
9269       inlineHtmlTag = {%
9270         \ifvmode
9271           \IgnorePar
9272         \EndP
9273         \fi
9274         \HCode{#1}%
9275       },
9276       inputBlockHtmlElement = {%
9277         \ifvmode
9278           \IgnorePar
9279         \fi
9280         \EndP
9281         \special{t4ht*<#1}%
9282         \par
9283         \ShowPar
9284       },
9285     },
9286   }
9287 }

```

3.3.4.4 Citations Here is a basic implementation for citations that uses the `LATEX` `\cite` macro. There are also implementations that use the `natbib` `\citep`, and

⁹See <https://tug.org/tex4ht/>.

`\citet` macros, and the Bib_{La}TeX `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```

9288 \newcount\markdownLaTeXCitationsCounter
9289
9290 % Basic implementation
9291 \RequirePackage{gobble}
9292 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
9293   \advance\markdownLaTeXCitationsCounter by 1\relax
9294   \ifx\relax#4\relax
9295     \ifx\relax#5\relax
9296       \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
9297         \cite{#1#2#6}% Without prenotes and postnotes, just accumulate cites
9298         \expandafter\expandafter\expandafter
9299         \expandafter\expandafter\expandafter\expandafter
9300         \@gobblethree
9301       \fi
9302     \else% Before a postnote (#5), dump the accumulator
9303       \ifx\relax#1\relax\else
9304         \cite{#1}%
9305       \fi
9306       \cite[#5]{#6}%
9307       \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
9308     \else
9309       \expandafter\expandafter\expandafter
9310       \expandafter\expandafter\expandafter\expandafter
9311       \expandafter\expandafter\expandafter
9312       \expandafter\expandafter\expandafter\expandafter
9313       \markdownLaTeXBasicCitations
9314     \fi
9315     \expandafter\expandafter\expandafter
9316     \expandafter\expandafter\expandafter\expandafter{%
9317     \expandafter\expandafter\expandafter
9318     \expandafter\expandafter\expandafter\expandafter}%
9319     \expandafter\expandafter\expandafter
9320     \expandafter\expandafter\expandafter\expandafter{%
9321     \expandafter\expandafter\expandafter
9322     \expandafter\expandafter\expandafter\expandafter}%
9323     \expandafter\expandafter\expandafter
9324     \@gobblethree
9325   \fi
9326 \else% Before a prenote (#4), dump the accumulator
9327   \ifx\relax#1\relax\else
9328     \cite{#1}%
9329   \fi
9330   \ifnum\markdownLaTeXCitationsCounter>1\relax
9331     \space % Insert a space before the prenote in later citations
9332   \fi

```

```

9333 #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
9334 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
9335 \else
9336   \expandafter\expandafter\expandafter
9337   \expandafter\expandafter\expandafter\expandafter
9338   \markdownLaTeXBasicCitations
9339 \fi
9340 \expandafter\expandafter\expandafter{%
9341 \expandafter\expandafter\expandafter}%
9342 \expandafter\expandafter\expandafter{%
9343 \expandafter\expandafter\expandafter}%
9344 \expandafter
9345 \@gobblethree
9346 \fi\markdownLaTeXBasicCitations{#1#2#6},}
9347 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
9348
9349 % Natbib implementation
9350 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
9351 \advance\markdownLaTeXCitationsCounter by 1\relax
9352 \ifx\relax#3\relax
9353 \ifx\relax#4\relax
9354 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
9355 \citep{#1,#5}% Without prenotes and postnotes, just accumulate cites
9356 \expandafter\expandafter\expandafter
9357 \expandafter\expandafter\expandafter\expandafter
9358 \@gobbletwo
9359 \fi
9360 \else% Before a postnote (#4), dump the accumulator
9361 \ifx\relax#1\relax\else
9362 \citep{#1}%
9363 \fi
9364 \citep[] [#4]{#5}%
9365 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
9366 \else
9367 \expandafter\expandafter\expandafter
9368 \expandafter\expandafter\expandafter\expandafter
9369 \expandafter\expandafter\expandafter
9370 \expandafter\expandafter\expandafter\expandafter
9371 \markdownLaTeXNatbibCitations
9372 \fi
9373 \expandafter\expandafter\expandafter
9374 \expandafter\expandafter\expandafter\expandafter{%
9375 \expandafter\expandafter\expandafter
9376 \expandafter\expandafter\expandafter\expandafter}%
9377 \expandafter\expandafter\expandafter
9378 \@gobbletwo
9379 \fi

```

```

9380 \else% Before a prenote (#3), dump the accumulator
9381 \ifx\relax#1\relax\relax\else
9382 \citep{#1}%
9383 \fi
9384 \citep[#3][#4]{#5}%
9385 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
9386 \else
9387 \expandafter\expandafter\expandafter
9388 \expandafter\expandafter\expandafter\expandafter
9389 \markdownLaTeXNatbibCitations
9390 \fi
9391 \expandafter\expandafter\expandafter{%
9392 \expandafter\expandafter\expandafter}%
9393 \expandafter
9394 \@gobbletwo
9395 \fi\markdownLaTeXNatbibCitations{#1,#5}}
9396 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
9397 \advance\markdownLaTeXCitationsCounter by 1\relax
9398 \ifx\relax#3\relax
9399 \ifx\relax#4\relax
9400 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
9401 \citet{#1,#5}% Without prenotes and postnotes, just accumulate cites
9402 \expandafter\expandafter\expandafter
9403 \expandafter\expandafter\expandafter\expandafter
9404 \@gobbletwo
9405 \fi
9406 \else% After a prenote or a postnote, dump the accumulator
9407 \ifx\relax#1\relax\else
9408 \citet{#1}%
9409 \fi
9410 , \citet[#3][#4]{#5}%
9411 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
9412 ,
9413 \else
9414 \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
9415 ,
9416 \fi
9417 \fi
9418 \expandafter\expandafter\expandafter
9419 \expandafter\expandafter\expandafter\expandafter
9420 \markdownLaTeXNatbibTextCitations
9421 \expandafter\expandafter\expandafter
9422 \expandafter\expandafter\expandafter\expandafter{%
9423 \expandafter\expandafter\expandafter
9424 \expandafter\expandafter\expandafter\expandafter}%
9425 \expandafter\expandafter\expandafter
9426 \@gobbletwo

```



```

9427   \fi
9428 \else% After a prenote or a postnote, dump the accumulator
9429   \ifx\relax#1\relax\relax\else
9430     \citet{#1}%
9431   \fi
9432   , \citet[#3][#4]{#5}%
9433   \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
9434     ,
9435   \else
9436     \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
9437     ,
9438   \fi
9439 \fi
9440 \expandafter\expandafter\expandafter
9441 \markdownLaTeXNatbibTextCitations
9442 \expandafter\expandafter\expandafter{%
9443 \expandafter\expandafter\expandafter}%
9444 \expandafter
9445 \@gobbletwo
9446 \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
9447
9448 % BibLaTeX implementation
9449 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
9450 \advance\markdownLaTeXCitationsCounter by 1\relax
9451 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
9452 \autocites#1[#3][#4]{#5}%
9453 \expandafter\@gobbletwo
9454 \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
9455 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
9456 \advance\markdownLaTeXCitationsCounter by 1\relax
9457 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
9458 \textcites#1[#3][#4]{#5}%
9459 \expandafter\@gobbletwo
9460 \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}}
9461
9462 \markdownSetup{rendererPrototypes = {
9463 cite = {%
9464 \markdownLaTeXCitationsCounter=1%
9465 \def\markdownLaTeXCitationsTotal{#1}%
9466 \@ifundefined{autocites}{%
9467 \@ifundefined{citep}{%
9468 \expandafter\expandafter\expandafter
9469 \markdownLaTeXBasicCitations
9470 \expandafter\expandafter\expandafter{%
9471 \expandafter\expandafter\expandafter}%
9472 \expandafter\expandafter\expandafter{%
9473 \expandafter\expandafter\expandafter}%

```

```

9474     }{%
9475     \expandafter\expandafter\expandafter
9476     \markdownLaTeXNatbibCitations
9477     \expandafter\expandafter\expandafter{%
9478     \expandafter\expandafter\expandafter}%
9479     }%
9480   }{%
9481     \expandafter\expandafter\expandafter
9482     \markdownLaTeXBibLaTeXCitations
9483     \expandafter{\expandafter}%
9484   }},
9485   textCite = {%
9486     \markdownLaTeXCitationsCounter=1%
9487     \def\markdownLaTeXCitationsTotal{#1}%
9488     \@ifundefined{autocites}{%
9489       \@ifundefined{citep}{%
9490         \expandafter\expandafter\expandafter
9491         \markdownLaTeXBasicTextCitations
9492         \expandafter\expandafter\expandafter{%
9493         \expandafter\expandafter\expandafter}%
9494         \expandafter\expandafter\expandafter{%
9495         \expandafter\expandafter\expandafter}%
9496       }{%
9497         \expandafter\expandafter\expandafter
9498         \markdownLaTeXNatbibTextCitations
9499         \expandafter\expandafter\expandafter{%
9500         \expandafter\expandafter\expandafter}%
9501       }%
9502     }{%
9503       \expandafter\expandafter\expandafter
9504       \markdownLaTeXBibLaTeXTextCitations
9505       \expandafter{\expandafter}%
9506     }}}

```

3.3.4.5 Links Before consuming the parameters for the hyperlink renderer, we change the category code of the hash sign (#) to other, so that it cannot be mistaken for a parameter character.

```

9507 \RequirePackage{url}
9508 \RequirePackage{expl3}
9509 \ExplSyntaxOn
9510 \def\markdownRendererLinkPrototype#1#2#3#4{
9511   \tl_set:Nn \l_tmpa_tl { #1 }
9512   \tl_set:Nn \l_tmpb_tl { #2 }
9513   \bool_set:Nn
9514     \l_tmpa_bool
9515     {

```

```

9516     \tl_if_eq_p:NN
9517     \l_tmpa_tl
9518     \l_tmpb_tl
9519   }
9520 \tl_set:Nn \l_tmpa_tl { #4 }
9521 \bool_set:Nn
9522   \l_tmpb_bool
9523   {
9524     \tl_if_empty_p:N
9525     \l_tmpa_tl
9526   }

```

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.

```

9527 \bool_if:nTF
9528   {
9529     \l_tmpa_bool && \l_tmpb_bool
9530   }
9531   {
9532     \markdownLaTeXRendererAutolink { #2 } { #3 }
9533   }{
9534     \markdownLaTeXRendererDirectOrIndirectLink { #1 } { #2 } { #3 } { #4 }
9535   }
9536 }
9537 \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.

```

9538 \tl_set:Nn
9539   \l_tmpa_tl
9540   { #2 }
9541 \tl_trim_spaces:N
9542   \l_tmpa_tl
9543 \tl_set:Nx
9544   \l_tmpb_tl
9545   {
9546     \tl_range:Nnn
9547     \l_tmpa_tl
9548     { 1 }
9549     { 1 }
9550   }
9551 \str_if_eq:NNTF
9552   \l_tmpb_tl
9553   \c_hash_str
9554   {
9555     \tl_set:Nx
9556     \l_tmpb_tl

```

```

9557     {
9558         \tl_range:Nnn
9559         \l_tmpa_tl
9560         { 2 }
9561         { -1 }
9562     }
9563     \exp_args:NV
9564     \ref
9565     \l_tmpb_tl
9566 }{
9567     \url { #2 }
9568 }
9569 }
9570 \ExplSyntaxOff
9571 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
9572     #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.

```

9573 \newcount\markdownLaTeXRowCount
9574 \newcount\markdownLaTeXRowTotal
9575 \newcount\markdownLaTeXColumnCounter
9576 \newcount\markdownLaTeXColumnTotal
9577 \newtoks\markdownLaTeXTable
9578 \newtoks\markdownLaTeXTableAlignment
9579 \newtoks\markdownLaTeXTableEnd
9580 \AtBeginDocument{%
9581     \@ifpackageloaded{booktabs}{%
9582         \def\markdownLaTeXTopRule{\toprule}%
9583         \def\markdownLaTeXMidRule{\midrule}%
9584         \def\markdownLaTeXBottomRule{\bottomrule}%
9585     }{%
9586         \def\markdownLaTeXTopRule{\hline}%
9587         \def\markdownLaTeXMidRule{\hline}%
9588         \def\markdownLaTeXBottomRule{\hline}%
9589     }%
9590 }
9591 \markdownSetup{rendererPrototypes={
9592     table = {%
9593         \markdownLaTeXTable={}%
9594         \markdownLaTeXTableAlignment={}%
9595         \markdownLaTeXTableEnd={%
9596             \markdownLaTeXBottomRule
9597             \end{tabular}}%
9598         \ifx\empty#1\empty\else
9599             \addto@hook\markdownLaTeXTable{%

```

```

9600     \begin{table}
9601     \centering}%
9602     \addto@hook\markdownLaTeXTableEnd{%
9603     \caption{#1}
9604     \end{table}}%
9605     \fi
9606     \addto@hook\markdownLaTeXTable{\begin{tabular}}}%
9607     \markdownLaTeXRowCounter=0%
9608     \markdownLaTeXRowTotal=#2%
9609     \markdownLaTeXColumnTotal=#3%
9610     \markdownLaTeXRenderTableRow
9611   }
9612 }}
9613 \def\markdownLaTeXRenderTableRow#1{%
9614   \markdownLaTeXColumnCounter=0%
9615   \ifnum\markdownLaTeXRowCounter=0\relax
9616     \markdownLaTeXReadAlignments#1%
9617     \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
9618       \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
9619         \the\markdownLaTeXTableAlignment}}}%
9620     \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
9621   \else
9622     \markdownLaTeXRenderTableCell#1%
9623   \fi
9624   \ifnum\markdownLaTeXRowCounter=1\relax
9625     \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
9626   \fi
9627   \advance\markdownLaTeXRowCounter by 1\relax
9628   \ifnum\markdownLaTeXRowCounter>\markdownLaTeXRowTotal\relax
9629     \the\markdownLaTeXTable
9630     \the\markdownLaTeXTableEnd
9631     \expandafter\@gobble
9632   \fi\markdownLaTeXRenderTableRow}
9633 \def\markdownLaTeXReadAlignments#1{%
9634   \advance\markdownLaTeXColumnCounter by 1\relax
9635   \if#1d%
9636     \addto@hook\markdownLaTeXTableAlignment{1}%
9637   \else
9638     \addto@hook\markdownLaTeXTableAlignment{#1}%
9639   \fi
9640   \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
9641     \expandafter\@gobble
9642   \fi\markdownLaTeXReadAlignments}
9643 \def\markdownLaTeXRenderTableCell#1{%
9644   \advance\markdownLaTeXColumnCounter by 1\relax
9645   \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
9646     \addto@hook\markdownLaTeXTable{#1&}%

```

```

9647 \else
9648   \addto@hook\markdownLaTeXTable{#1\}%
9649   \expandafter@gobble
9650 \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.

```

9651
9652 \markdownIfOption{lineBlocks}{%
9653   \RequirePackage{verse}
9654   \markdownSetup{rendererPrototypes={
9655     lineBlockBegin = {%
9656       \begingroup
9657       \def\markdownRendererHardLineBreak{\}%
9658       \begin{verse}%
9659     },
9660     lineBlockEnd = {%
9661       \end{verse}%
9662     \endgroup
9663   },
9664 }}
9665 }{}
9666

```

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.

```

9667 \ExplSyntaxOn
9668 \keys_define:nn
9669 { markdown/jekyllData }
9670 {
9671   author .code:n = { \author{#1} },
9672   date   .code:n = { \date{#1}   },
9673   title  .code:n = { \title{#1}  },
9674 }

```

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.

```

9675 % TODO: Remove the command definition in TeX Live 2021.
9676 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
9677 \markdownSetup{
9678   rendererPrototypes = {
9679     jekyllDataEnd = {

```

```

9680 %      TODO: Remove the else branch in TeX Live 2021.
9681 \IfFormatAtLeastTF
9682   { 2020-10-01 }
9683   { \AddToHook{begindocument/end}{\maketitle} }
9684   {
9685     \ifx\@onlypreamble\@notprerr
9686       % We are in the document
9687       \maketitle
9688     \else
9689       % We are in the preamble
9690       \RequirePackage{etoolbox}
9691       \AfterEndPreamble{\maketitle}
9692     \fi
9693   }
9694 },
9695 },
9696 }
9697 \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.

```

9698 \markdownIfOption{strikeThrough}{%
9699   \RequirePackage{soulutf8}%
9700   \markdownSetup{
9701     rendererPrototypes = {
9702       strikeThrough = {%
9703         \st{#1}%
9704       },
9705     }
9706   }
9707 }{}

```

3.3.4.10 Raw Attribute Renderer Prototypes In the raw block and inline raw span renderer prototypes, default to the plain TeX renderer prototypes, translating raw attribute `latex` to `tex`.

```

9708 \ExplSyntaxOn
9709 \cs_gset:Npn
9710   \markdownRendererInputRawInlinePrototype#1#2
9711   {
9712     \str_case:nnF
9713       { #2 }
9714       {
9715         { latex }
9716         {
9717           \@_plain_tex_default_input_raw_inline_renderer_prototype:nn

```

```

9718         { #1 }
9719         { tex }
9720     }
9721 }
9722 {
9723     \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
9724     { #1 }
9725     { #2 }
9726 }
9727 }
9728 \cs_gset:Npn
9729 \markdownRendererInputRawBlockPrototype#1#2
9730 {
9731     \str_case:nnF
9732     { #2 }
9733     {
9734         { latex }
9735         {
9736             \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
9737             { #1 }
9738             { tex }
9739         }
9740     }
9741     {
9742         \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
9743         { #1 }
9744         { #2 }
9745     }
9746 }
9747 \ExplSyntaxOff
9748 \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.

```

9749 \newcommand\markdownMakeOther{%
9750     \count0=128\relax
9751     \loop
9752         \catcode\count0=11\relax
9753         \advance\count0 by 1\relax
9754     \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^AT_EX package.

```
9755 \def\markdownMakeOther{%
9756   \count0=128\relax
9757   \loop
9758     \catcode\count0=11\relax
9759     \advance\count0 by 1\relax
9760   \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.

```
9761   \catcode`|=12}%
```

3.4.1 Typesetting Markdown

The `\inputmarkdown` is defined to accept an optional argument with options recognized by the ConTeXt interface (see Section 2.4.2).

```
9762 \long\def\inputmarkdown{%
9763   \dosingleempty
9764   \doinputmarkdown}%
9765 \long\def\doinputmarkdown[#1]#2{%
9766   \begingroup
9767     \iffirstargument
9768     \setupmarkdown{#1}%
9769   \fi
9770   \markdownInput{#2}%
9771   \endgroup}%
```

The `\startmarkdown` and `\stopmarkdown` macros are implemented using the `\markdownReadAndConvert` macro.

In Knuth's TeX, 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)TeX, but ConTeXt MkIV funnels all input through its own input handler. This makes it possible to suppress the removal of trailing spaces in ConTeXt MkIV and therefore to insert hard line breaks into markdown text.

```

9772 \ifx\startluacode\undefined % MkII
9773 \beginingroup
9774 \catcode`\|=0%
9775 \catcode`\|=12%
9776 |gdef|startmarkdown{%
9777 |markdownReadAndConvert{\stopmarkdown}%
9778 |stopmarkdown}}%
9779 |gdef|stopmarkdown{%
9780 |markdownEnd}%
9781 |endgroup
9782 \else % MkIV
9783 \startluacode
9784 document.markdown_buffering = false
9785 local function preserve_trailing_spaces(line)
9786 if document.markdown_buffering then
9787 line = line:gsub("[ \t][ \t]$", "\t\t")
9788 end
9789 return line
9790 end
9791 resolvers.installinputlinehandler(preserve_trailing_spaces)
9792 \stoptluacode
9793 \beginingroup
9794 \catcode`\|=0%
9795 \catcode`\|=12%
9796 |gdef|startmarkdown{%
9797 |ctxlua{document.markdown_buffering = true}%
9798 |markdownReadAndConvert{\stopmarkdown}%
9799 |stopmarkdown}}%
9800 |gdef|stopmarkdown{%
9801 |ctxlua{document.markdown_buffering = false}%
9802 |markdownEnd}%
9803 |endgroup
9804 \fi

```

3.4.2 Token Renderer Prototypes

The following configuration should be considered placeholder.

```

9805 \def\markdownRendererHardLineBreakPrototype{\blank}%
9806 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
9807 \def\markdownRendererRightBracePrototype{\textbraceright}%
9808 \def\markdownRendererDollarSignPrototype{\textdollar}%
9809 \def\markdownRendererPercentSignPrototype{\percent}%
9810 \def\markdownRendererUnderscorePrototype{\textunderscore}%
9811 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
9812 \def\markdownRendererBackslashPrototype{\textbackslash}%
9813 \def\markdownRendererTildePrototype{\textasciitilde}%
9814 \def\markdownRendererPipePrototype{\char`|}%

```

```

9815 \def\markdownRendererLinkPrototype#1#2#3#4{%
9816   \useURL[#1][#3][[#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
9817   \fi\texttt<\hyphenatedurl{#3}>}}%
9818 \usemodule[database]
9819 \defineseparatedlist
9820   [MarkdownConTeXtCSV]
9821   [separator={,},
9822   before=\bTABLE,after=\eTABLE,
9823   first=\bTR,last=\eTR,
9824   left=\bTD,right=\eTD]
9825 \def\markdownConTeXtCSV{csv}
9826 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
9827   \def\markdownConTeXtCSV@arg{#1}%
9828   \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
9829     \placetable[] [tab:#1]{#4}{%
9830       \processseparatedfile [MarkdownConTeXtCSV] [#3]}%
9831   \else
9832     \markdownInput{#3}%
9833   \fi}%
9834 \def\markdownRendererImagePrototype#1#2#3#4{%
9835   \placefigure[] []{#4}{\externalfigure[#3]}%
9836 \def\markdownRendererUlBeginPrototype{\startitemize}%
9837 \def\markdownRendererUlBeginTightPrototype{\startitemize[packed]}%
9838 \def\markdownRendererUlItemPrototype{\item}%
9839 \def\markdownRendererUlEndPrototype{\stopitemize}%
9840 \def\markdownRendererUlEndTightPrototype{\stopitemize}%
9841 \def\markdownRendererOlBeginPrototype{\startitemize[n]}%
9842 \def\markdownRendererOlBeginTightPrototype{\startitemize[packed,n]}%
9843 \def\markdownRendererOlItemPrototype{\item}%
9844 \def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
9845 \def\markdownRendererOlEndPrototype{\stopitemize}%
9846 \def\markdownRendererOlEndTightPrototype{\stopitemize}%
9847 \definedescription
9848   [MarkdownConTeXtDlItemPrototype]
9849   [location=hanging,
9850   margin=standard,
9851   headstyle=bold]%
9852 \definestartstop
9853   [MarkdownConTeXtDlPrototype]
9854   [before=\blank,
9855   after=\blank]%
9856 \definestartstop
9857   [MarkdownConTeXtDlTightPrototype]
9858   [before=\blank\startpacked,
9859   after=\stoppacked\blank]%
9860 \def\markdownRendererDlBeginPrototype{%
9861   \startMarkdownConTeXtDlPrototype}%

```

```

9862 \def\markdownRendererDlBeginTightPrototype{%
9863   \startMarkdownConTeXtDlTightPrototype}%
9864 \def\markdownRendererDlItemPrototype#1{%
9865   \startMarkdownConTeXtDlItemPrototype{#1}}%
9866 \def\markdownRendererDlItemEndPrototype{%
9867   \stopMarkdownConTeXtDlItemPrototype}%
9868 \def\markdownRendererDlEndPrototype{%
9869   \stopMarkdownConTeXtDlPrototype}%
9870 \def\markdownRendererDlEndTightPrototype{%
9871   \stopMarkdownConTeXtDlTightPrototype}%
9872 \def\markdownRendererEmphasisPrototype#1{\em#1}%
9873 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
9874 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
9875 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
9876 \def\markdownRendererLineBlockBeginPrototype{%
9877   \begingroup
9878     \def\markdownRendererHardLineBreak{
9879       }%
9880     \startlines
9881   }%
9882 \def\markdownRendererLineBlockEndPrototype{%
9883   \stoplines
9884   \endgroup
9885 }%
9886 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%

```

3.4.2.1 Fenced Code When no infostring has been specified, default to the indented code block renderer.

```

9887 \ExplSyntaxOn
9888 \cs_gset:Npn
9889   \markdownRendererInputFencedCodePrototype#1#2
9890   {
9891     \tl_if_empty:nTF
9892       { #2 }
9893       { \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

```

```

9894     {
9895         \regex_extract_once:nnN
9896         { \w* }
9897         { #2 }
9898         \l_tmpa_seq
9899         \seq_pop_left:NN
9900         \l_tmpa_seq
9901         \l_tmpa_tl
9902         \typefile[\l_tmpa_tl] []{#1}
9903     }
9904 }
9905 \ExplSyntaxOff
9906 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
9907 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
9908 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
9909 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
9910 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
9911 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
9912 \def\markdownRendererThematicBreakPrototype{%
9913   \blackrule[height=1pt, width=\hsize]}%
9914 \def\markdownRendererFootnotePrototype#1{\footnote{#1}}%
9915 \def\markdownRendererTickedBoxPrototype{\$ \boxtimes \$}
9916 \def\markdownRendererHalfTickedBoxPrototype{\$ \boxdot \$}
9917 \def\markdownRendererUntickedBoxPrototype{\$ \square \$}
9918 \def\markdownRendererStrikeThroughPrototype#1{\overstrides{#1}}
9919 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
9920 \def\markdownRendererSubscriptPrototype#1{\low{#1}}
9921 \def\markdownRendererDisplayMathPrototype#1{\startformula#1\stopformula}%
9922 \def\markdownRendererInlineMathPrototype#1{\$#1\$}%

```

3.4.2.2 Tables There is a basic implementation of tables.

```

9923 \newcount\markdownConTeXtRowCounter
9924 \newcount\markdownConTeXtRowTotal
9925 \newcount\markdownConTeXtColumnCounter
9926 \newcount\markdownConTeXtColumnTotal
9927 \newtoks\markdownConTeXtTable

```

```

9928 \newtoks\markdownConTeXtTableFloat
9929 \def\markdownRendererTablePrototype#1#2#3{%
9930   \markdownConTeXtTable={}%
9931   \ifx\empty#1\empty
9932     \markdownConTeXtTableFloat={%
9933       \the\markdownConTeXtTable}%
9934   \else
9935     \markdownConTeXtTableFloat={%
9936       \placetable{#1}{\the\markdownConTeXtTable}}%
9937   \fi
9938   \begingroup
9939   \setupTABLE[r][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
9940   \setupTABLE[c][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
9941   \setupTABLE[r][1][topframe=on, bottomframe=on]
9942   \setupTABLE[r][#1][bottomframe=on]
9943   \markdownConTeXtRowCount=0%
9944   \markdownConTeXtRowTotal=#2%
9945   \markdownConTeXtColumnTotal=#3%
9946   \markdownConTeXtRenderTableRow}
9947 \def\markdownConTeXtRenderTableRow#1{%
9948   \markdownConTeXtColumnCounter=0%
9949   \ifnum\markdownConTeXtRowCount=0\relax
9950     \markdownConTeXtReadAlignments#1%
9951     \markdownConTeXtTable={\bTABLE}%
9952   \else
9953     \markdownConTeXtTable=\expandafter{%
9954       \the\markdownConTeXtTable\bTR}%
9955     \markdownConTeXtRenderTableCell#1%
9956     \markdownConTeXtTable=\expandafter{%
9957       \the\markdownConTeXtTable\eTR}%
9958   \fi
9959   \advance\markdownConTeXtRowCount by 1\relax
9960   \ifnum\markdownConTeXtRowCount>\markdownConTeXtRowTotal\relax
9961     \markdownConTeXtTable=\expandafter{%
9962       \the\markdownConTeXtTable\eTABLE}%
9963     \the\markdownConTeXtTableFloat
9964   \endgroup
9965   \expandafter\gobbleoneargument
9966   \fi\markdownConTeXtRenderTableRow}
9967 \def\markdownConTeXtReadAlignments#1{%
9968   \advance\markdownConTeXtColumnCounter by 1\relax
9969   \if#1d%
9970     \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
9971   \fi\if#1l%
9972     \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
9973   \fi\if#1c%
9974     \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]

```

```

9975 \fi\if#1r%
9976 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
9977 \fi
9978 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
9979 \expandafter\gobbleoneargument
9980 \fi\markdownConTeXtReadAlignments}
9981 \def\markdownConTeXtRenderTableCell#1{%
9982 \advance\markdownConTeXtColumnCounter by 1\relax
9983 \markdownConTeXtTable=\expandafter{%
9984 \the\markdownConTeXtTable\bTD#1\eTD}%
9985 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
9986 \expandafter\gobbleoneargument
9987 \fi\markdownConTeXtRenderTableCell}

```

3.4.2.3 Raw Attribute Renderer Prototypes In the raw block and inline raw span renderer prototypes, default to the plain TeX renderer prototypes, translating raw attribute `context` to `tex`.

```

9988 \ExplSyntaxOn
9989 \cs_gset:Npn
9990 \markdownRendererInputRawInlinePrototype#1#2
9991 {
9992 \str_case:nnF
9993 { #2 }
9994 {
9995 { latex }
9996 {
9997 \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
9998 { #1 }
9999 { context }
10000 }
10001 }
10002 {
10003 \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
10004 { #1 }
10005 { #2 }
10006 }
10007 }
10008 \cs_gset:Npn
10009 \markdownRendererInputRawBlockPrototype#1#2
10010 {
10011 \str_case:nnF
10012 { #2 }
10013 {
10014 { context }
10015 {
10016 \@@_plain_tex_default_input_raw_block_renderer_prototype:nn

```

```

10017         { #1 }
10018         { tex }
10019     }
10020 }
10021 {
10022     \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
10023     { #1 }
10024     { #2 }
10025 }
10026 }
10027 \cs_gset_eq:NN
10028   \markdownRendererInputRawBlockPrototype
10029   \markdownRendererInputRawInlinePrototype
10030 \ExplSyntaxOff
10031 \stopmodule\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 Novotný. *L^AT_EX 2_ε no longer keys packages by pathnames*. Feb. 20, 2021. URL: <https://github.com/latex3/latex2e/issues/510> (visited on 02/21/2021).

- [10] Geoffrey M. Poore. *The minted Package. Highlighted source code in L^AT_EX*. 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^AT_EX 2_ε Sources*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/source2e.pdf> (visited on 01/08/2018).
- [13] Donald Ervin Knuth. *T_EX: The Program*. Vol. B. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. xvi, 594. ISBN: 0-201-13437-7.
- [14] Victor Eijkhout. *T_EX by Topic. A T_EXnician's Reference*. Wokingham, England: Addison-Wesley, Feb. 1, 1992. 307 pp. ISBN: 0-201-56882-0.

Index

| | |
|---------------------------------------|---------------------------------------|
| <code>blankBeforeBlockquote</code> | 17 |
| <code>blankBeforeCodeFence</code> | 17 |
| <code>blankBeforeDivFence</code> | 17 |
| <code>blankBeforeHeading</code> | 18 |
| <code>bracketedSpans</code> | 18, 53 |
| <code>breakableBlockquotes</code> | 18 |
| <code>cacheDir</code> | 4, 15, 22, 46, 47, 106, 204, 253, 266 |
| <code>citationNbsps</code> | 19 |
| <code>citations</code> | 19, 77, 84 |
| <code>codeSpans</code> | 20 |
| <code>contentBlocks</code> | 16, 20 |
| <code>contentBlocksLanguageMap</code> | 16 |
| <code>debugExtensions</code> | 9, 16, 21, 201 |
| <code>debugExtensionsFileName</code> | 16, 21 |
| <code>defaultOptions</code> | 9, 42, 239 |
| <code>definitionLists</code> | 21, 58 |
| <code>eagerCache</code> | 22, 22 |
| <code>\endmarkdown</code> | 94 |
| <code>entities.char_entity</code> | 162 |
| <code>entities.dec_entity</code> | 162 |
| <code>entities.hex_entity</code> | 162 |
| <code>errorTempFileName</code> | 48, 259 |
| <code>expandtabs</code> | 186 |

| | |
|------------------------------|---|
| expectJekyllData | 22 |
| extensions | 23, 113, 205 |
| extensions.bracketed_spans | 205 |
| extensions.citations | 206 |
| extensions.content_blocks | 209 |
| extensions.definition_lists | 212 |
| extensions.fancy_lists | 214 |
| extensions.fenced_code | 218 |
| extensions.fenced_divs | 222 |
| extensions.header_attributes | 225 |
| extensions.jekyll_data | 235 |
| extensions.line_blocks | 226 |
| extensions.notes | 227 |
| extensions.pipe_table | 228 |
| extensions.raw_inline | 232 |
| extensions.strike_through | 232 |
| extensions.subscripts | 233 |
| extensions.superscripts | 233 |
| extensions.tex_math_dollars | 234 |
|
 | |
| fancyLists | 25, 72–76, 269 |
| fencedCode | 25, 33, 56, 61, 77, 264 |
| fencedCodeAttributes | 26 |
| fencedDiv | 62 |
| fencedDivs | 26, 34 |
| finalizeCache | 16, 22, 27, 27, 46, 47, 105, 205 |
| frozenCache | 16, 27, 47, 100, 105, 264, 266 |
| frozenCacheCounter | 27, 205, 261, 262 |
| frozenCacheFileName | 16, 27, 46, 205 |
|
 | |
| hardLineBreaks | 28 |
| hashEnumerators | 28 |
| headerAttributes | 28, 34, 51, 63 |
| helperScriptFileName | 47–49, 259 |
| html | 29, 65, 66, 277 |
| hybrid | 29, 33, 38, 39, 48, 79, 101, 106, 166, 187, 261 |
|
 | |
| inlineNotes | 30 |
| \inputmarkdown | 110, 110, 111, 289 |
| inputTempFileName | 255, 256 |
| iterlines | 186 |
|
 | |
| jekyllData | 3, 22, 23, 30, 86–89 |

| | |
|--|-------------------------------|
| languages_json | 210, 210 |
| lineBlocks | 31, 67 |
|
 | |
| \markdown | 94 |
| markdown | 94, 94, 263 |
| markdown* | 94, 94, 95, 263 |
| \markdown_jekyll_data_concatenate_address:NN | 249 |
| \markdown_jekyll_data_pop: | 250 |
| \markdown_jekyll_data_push:nN | 250 |
| \markdown_jekyll_data_push_address_segment:n | 248 |
| \markdown_jekyll_data_set_keyval:Nn | 250 |
| \markdown_jekyll_data_set_keyvals:nn | 250 |
| \markdown_jekyll_data_update_address_tls: | 249 |
| \markdownBegin | 44, 44, 45, 91, 94, 110 |
| \markdownEnd | 44, 44, 45, 91, 94, 110 |
| \markdownError | 91, 91 |
| \markdownEscape | 44, 45, 46, 262 |
| \markdownExecute | 258 |
| \markdownExecuteDirect | 258, 258 |
| \markdownExecuteShellEscape | 258, 258 |
| \markdownIfOption | 254 |
| \markdownIfSnippetExists | 95 |
| \markdownInfo | 91 |
| \markdownInput | 44, 45, 94, 95, 110, 261, 263 |
| \markdownInputFileStream | 254 |
| \markdownInputPlainTeX | 263 |
| \markdownLuaExecute | 257, 259, 260, 261 |
| \markdownLuaOptions | 251, 253 |
| \markdownLuaRegisterIBCcallback | 93 |
| \markdownLuaUnregisterIBCcallback | 93 |
| \markdownMakeOther | 91, 288, 289 |
| \markdownMode | 4, 47, 48, 92, 92, 257, 260 |
| \markdownOptionErrorTempFileName | 48 |
| \markdownOptionFinalizeCache | 46 |
| \markdownOptionFrozenCache | 46 |
| \markdownOptionHelperScriptFileName | 47 |
| \markdownOptionHybrid | 48 |
| \markdownOptionInputTempFileName | 47 |
| \markdownOptionOutputDir | 48 |
| \markdownOptionOutputTempFileName | 47 |
| \markdownOptionStripPercentSigns | 50 |
| \markdownOutputFileStream | 254 |

| | |
|--|-------------------|
| <code>\markdownPrepare</code> | 253 |
| <code>\markdownPrepareLuaOptions</code> | 251 |
| <code>\markdownReadAndConvert</code> | 91, 255, 263, 289 |
| <code>\markdownReadAndConvertProcessLine</code> | 256, 257 |
| <code>\markdownReadAndConvertStripPercentSigns</code> | 255 |
| <code>\markdownReadAndConvertTab</code> | 254 |
| <code>\markdownRendererAttributeClassName</code> | 51 |
| <code>\markdownRendererAttributeIdentifier</code> | 51 |
| <code>\markdownRendererAttributeKeyValue</code> | 51 |
| <code>\markdownRendererBlockHtmlCommentBegin</code> | 65 |
| <code>\markdownRendererBlockHtmlCommentEnd</code> | 65 |
| <code>\markdownRendererBlockQuoteBegin</code> | 52 |
| <code>\markdownRendererBlockQuoteEnd</code> | 52 |
| <code>\markdownRendererBracketedSpanAttributeContextBegin</code> | 53 |
| <code>\markdownRendererBracketedSpanAttributeContextEnd</code> | 53 |
| <code>\markdownRendererCite</code> | 77, 84 |
| <code>\markdownRendererCodeSpan</code> | 56 |
| <code>\markdownRendererContentBlock</code> | 56, 57 |
| <code>\markdownRendererContentBlockCode</code> | 57 |
| <code>\markdownRendererContentBlockOnlineImage</code> | 57 |
| <code>\markdownRendererDisplayMath</code> | 83 |
| <code>\markdownRendererDlBegin</code> | 58 |
| <code>\markdownRendererDlBeginTight</code> | 58 |
| <code>\markdownRendererDlDefinitionBegin</code> | 59 |
| <code>\markdownRendererDlDefinitionEnd</code> | 59 |
| <code>\markdownRendererDlEnd</code> | 60 |
| <code>\markdownRendererDlEndTight</code> | 60 |
| <code>\markdownRendererDlItem</code> | 58 |
| <code>\markdownRendererDlItemEnd</code> | 59 |
| <code>\markdownRendererDocumentBegin</code> | 70 |
| <code>\markdownRendererDocumentEnd</code> | 70 |
| <code>\markdownRendererEllipsis</code> | 35, 60 |
| <code>\markdownRendererEmphasis</code> | 61, 107 |
| <code>\markdownRendererFancyOlBegin</code> | 72, 73 |
| <code>\markdownRendererFancyOlBeginTight</code> | 73 |
| <code>\markdownRendererFancyOlEnd</code> | 76 |
| <code>\markdownRendererFancyOlEndTight</code> | 76 |
| <code>\markdownRendererFancyOlItem</code> | 74 |
| <code>\markdownRendererFancyOlItemEnd</code> | 74 |
| <code>\markdownRendererFancyOlItemWithNumber</code> | 75 |
| <code>\markdownRendererFencedCodeAttributeContextBegin</code> | 61 |
| <code>\markdownRendererFencedCodeAttributeContextEnd</code> | 61 |

| | |
|--|---------|
| <code>\markdownRendererFencedDivAttributeContextBegin</code> | 62 |
| <code>\markdownRendererFencedDivAttributeContextEnd</code> | 62 |
| <code>\markdownRendererFootnote</code> | 71, 90 |
| <code>\markdownRendererFootnotePrototype</code> | 71, 90 |
| <code>\markdownRendererHalfTickedBox</code> | 85 |
| <code>\markdownRendererHardLineBreak</code> | 68 |
| <code>\markdownRendererHeaderAttributeContextBegin</code> | 63 |
| <code>\markdownRendererHeaderAttributeContextEnd</code> | 63 |
| <code>\markdownRendererHeadingFive</code> | 65 |
| <code>\markdownRendererHeadingFour</code> | 64 |
| <code>\markdownRendererHeadingOne</code> | 63 |
| <code>\markdownRendererHeadingSix</code> | 65 |
| <code>\markdownRendererHeadingThree</code> | 64 |
| <code>\markdownRendererHeadingTwo</code> | 64 |
| <code>\markdownRendererHorizontalRule</code> | 84, 90 |
| <code>\markdownRendererHorizontalRulePrototype</code> | 84, 90 |
| <code>\markdownRendererImage</code> | 67 |
| <code>\markdownRendererInlineHtmlComment</code> | 65 |
| <code>\markdownRendererInlineHtmlTag</code> | 66 |
| <code>\markdownRendererInlineMath</code> | 83 |
| <code>\markdownRendererInputBlockHtmlElement</code> | 66 |
| <code>\markdownRendererInputFencedCode</code> | 56 |
| <code>\markdownRendererInputRawBlock</code> | 77 |
| <code>\markdownRendererInputRawInline</code> | 77 |
| <code>\markdownRendererInputVerbatim</code> | 55 |
| <code>\markdownRendererInterblockSeparator</code> | 67 |
| <code>\markdownRendererJekyllDataBegin</code> | 86 |
| <code>\markdownRendererJekyllDataBoolean</code> | 88 |
| <code>\markdownRendererJekyllDataEmpty</code> | 89 |
| <code>\markdownRendererJekyllDataEnd</code> | 86 |
| <code>\markdownRendererJekyllDataMappingBegin</code> | 86 |
| <code>\markdownRendererJekyllDataMappingEnd</code> | 87 |
| <code>\markdownRendererJekyllDataNumber</code> | 88 |
| <code>\markdownRendererJekyllDataSequenceBegin</code> | 87 |
| <code>\markdownRendererJekyllDataSequenceEnd</code> | 88 |
| <code>\markdownRendererJekyllDataString</code> | 89 |
| <code>\markdownRendererLineBlockBegin</code> | 68 |
| <code>\markdownRendererLineBlockEnd</code> | 68 |
| <code>\markdownRendererLineBreak</code> | 68 |
| <code>\markdownRendererLineBreakPrototype</code> | 68 |
| <code>\markdownRendererLink</code> | 69, 107 |
| <code>\markdownRendererNbsp</code> | 70 |

| | |
|--|-----------------------|
| <code>\markdownRendererNote</code> | 71 |
| <code>\markdownRendererOlBegin</code> | 72 |
| <code>\markdownRendererOlBeginTight</code> | 72 |
| <code>\markdownRendererOlEnd</code> | 75 |
| <code>\markdownRendererOlEndTight</code> | 75 |
| <code>\markdownRendererOlItem</code> | 35, 73 |
| <code>\markdownRendererOlItemEnd</code> | 73 |
| <code>\markdownRendererOlItemWithNumber</code> | 35, 74 |
| <code>\markdownRendererReplacementCharacter</code> | 78 |
| <code>\markdownRendererSectionBegin</code> | 78 |
| <code>\markdownRendererSectionEnd</code> | 78 |
| <code>\markdownRendererStrikeThrough</code> | 81 |
| <code>\markdownRendererStrongEmphasis</code> | 61 |
| <code>\markdownRendererSubscript</code> | 82 |
| <code>\markdownRendererSuperscript</code> | 82 |
| <code>\markdownRendererTable</code> | 82 |
| <code>\markdownRendererTextCite</code> | 83 |
| <code>\markdownRendererThematicBreak</code> | 84 |
| <code>\markdownRendererTickedBox</code> | 85 |
| <code>\markdownRendererUlBegin</code> | 53 |
| <code>\markdownRendererUlBeginTight</code> | 54 |
| <code>\markdownRendererUlEnd</code> | 55 |
| <code>\markdownRendererUlEndTight</code> | 55 |
| <code>\markdownRendererUlItem</code> | 54 |
| <code>\markdownRendererUlItemEnd</code> | 54 |
| <code>\markdownRendererUntickedBox</code> | 85 |
| <code>\markdownSetup</code> | 95, 95, 262, 267 |
| <code>\markdownSetupSnippet</code> | 95, 95 |
| <code>\markdownWarning</code> | 91 |
|
 | |
| <code>new</code> | 8, 238 |
| <code>notes</code> | 31, 71 |
|
 | |
| <code>outputTempFileName</code> | 48, 259, 260 |
|
 | |
| <code>parsers</code> | 175, 185, 186 |
| <code>parsers.commented_line</code> | 177 |
| <code>pipeTables</code> | 6, 32, 37, 82 |
| <code>preserveTabs</code> | 32, 36, 186 |
|
 | |
| <code>rawAttribute</code> | 33, 33, 77 |
| <code>reader</code> | 8, 113, 175, 185, 205 |
| <code>reader->add_special_character</code> | 8, 9, 200 |

| | |
|--------------------------------|-------------------------------------|
| reader->create_parser | 187 |
| reader->finalize_grammar | 197 |
| reader->initialize_named_group | 200 |
| reader->insert_pattern | 8, 9, 197, 202 |
| reader->normalize_tag | 186 |
| reader->options | 185 |
| reader->parser_functions | 187 |
| reader->parser_functions.name | 187 |
| reader->parsers | 185, 185, 186 |
| reader->update_rule | 197, 199, 202 |
| reader->writer | 185 |
| reader.new | 185, 185 |
| relativeReferences | 33 |
| | |
| \setupmarkdown | 111, 111 |
| shiftHeadings | 6, 34 |
| slice | 6, 34, 163, 171, 172 |
| smartEllipses | 35, 60, 106 |
| \startmarkdown | 110, 110, 289 |
| startNumber | 35, 73–75 |
| \stopmarkdown | 110, 110, 289 |
| strikeThrough | 35, 81, 287 |
| stripIndent | 36, 187 |
| stripPercentSigns | 255 |
| subscripts | 36, 82 |
| superscripts | 36, 82 |
| syntax | 198, 202 |
| | |
| tableCaptions | 6, 37 |
| taskLists | 37, 85, 276 |
| texComments | 38, 187 |
| texMathDollars | 38, 83 |
| tightLists | 39, 54, 55, 58, 60, 72, 73, 76, 269 |
| | |
| underscores | 39 |
| util.cache | 114, 114 |
| util.cache_verbatim | 114 |
| util.encode_json_string | 114 |
| util.err | 114 |
| util.escaper | 117 |
| util.expand_tabs_in_line | 115 |
| util.flatten | 116 |
| util.intersperse | 117 |

| | |
|----------------------------------|-------------------------|
| util.lookup_files | 115 |
| util.map | 117 |
| util.pathname | 118 |
| util.rope_last | 116 |
| util.rope_to_string | 116 |
| util.table_copy | 114 |
| util.walk | 115, 116 |
| walkable_syntax | 8, 16, 21, 197, 199–202 |
| writer | 113, 113, 162, 205 |
| writer->active_attributes | 170, 170–172 |
| writer->attributes | 170 |
| writer->block_html_comment | 168 |
| writer->block_html_element | 168 |
| writer->blockquote | 169 |
| writer->bulletitem | 167 |
| writer->bulletlist | 166 |
| writer->citations | 206 |
| writer->code | 166 |
| writer->contentblock | 210 |
| writer->defer_call | 174, 174 |
| writer->definitionlist | 213 |
| writer->display_math | 234 |
| writer->div_begin | 222 |
| writer->div_end | 222 |
| writer->document | 169 |
| writer->ellipsis | 164 |
| writer->emphasis | 168 |
| writer->escape | 165 |
| writer->escape_minimal | 165 |
| writer->escape_programmatic_text | 165 |
| writer->escape_typographic_text | 165 |
| writer->escaped_chars | 165, 165 |
| writer->escaped_minimal_strings | 164, 165 |
| writer->escaped_strings | 165 |
| writer->escaped_uri_chars | 164, 165 |
| writer->fancyitem | 215 |
| writer->fancylist | 215 |
| writer->fencedCode | 219 |
| writer->get_state | 174 |
| writer->hard_line_break | 164 |
| writer->heading | 172 |

| | |
|-----------------------------|---------------|
| writer->identifier | 166 |
| writer->image | 166 |
| writer->inline_html_comment | 168 |
| writer->inline_html_tag | 168 |
| writer->inline_math | 234 |
| writer->interblocksep | 164 |
| writer->is_writing | 163, 163 |
| writer->jekyllData | 236 |
| writer->lineblock | 226 |
| writer->link | 166 |
| writer->math | 166 |
| writer->nbsp | 164 |
| writer->note | 227 |
| writer->options | 163 |
| writer->ordereditem | 168 |
| writer->orderedlist | 167 |
| writer->pack | 164, 204 |
| writer->paragraph | 164 |
| writer->plain | 164 |
| writer->pop_attributes | 170, 171, 172 |
| writer->push_attributes | 170, 171, 172 |
| writer->rawBlock | 219 |
| writer->rawInline | 232 |
| writer->set_state | 174 |
| writer->slice_begin | 163 |
| writer->slice_end | 163 |
| writer->space | 164 |
| writer->span | 206 |
| writer->strike_through | 232 |
| writer->string | 166 |
| writer->strong | 169 |
| writer->subscript | 233 |
| writer->suffix | 164 |
| writer->superscript | 234 |
| writer->table | 230 |
| writer->thematic_break | 164 |
| writer->checkbox | 169 |
| writer->uri | 166 |
| writer->verbatim | 169 |
| writer.new | 162, 163, 163 |