

A Markdown Interpreter for T_EX

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

Version 3.11.3-0-ge29029b2
2025-05-28

Contents

1	Introduction	1	3	Implementation	167
1.1	Requirements	2	3.1	Lua Implementation . . .	167
1.2	Feedback	6	3.2	Plain T _E X Implementation	381
1.3	Acknowledgements	7	3.3	L ^A T _E X Implementation . .	424
2	Interfaces	7	3.4	ConT _E Xt Implementation	465
2.1	Lua Interface	7			
2.2	Plain T _E X Interface	53		References	476
2.3	L ^A T _E X Interface	153			
2.4	ConT _E Xt Interface	162		Index	477

List of Figures

1	A block diagram of the Markdown package	8
2	A sequence diagram of typesetting a document using the T _E X interface . .	49
3	A sequence diagram of typesetting a document using the Lua CLI	50
4	An example directed graph	76
5	An example mindmap	77
6	An example UML sequence diagram	78
7	The banner of the Markdown package	79
8	A pushdown automaton that recognizes T _E X comments	261

1 Introduction

The Markdown package¹ converts CommonMark² 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 converters, the Markdown package does not require any external programs, and makes it easy to redefine how each and every markdown element is rendered. Creative abuse of the markdown syntax is encouraged. 😊

This document is a technical documentation for the Markdown package. It consists of three sections. This section introduces the package and outlines its prerequisites.

¹See <https://ctan.org/pkg/markdown>.

²See <https://commonmark.org/>.

Section 2 describes the interfaces exposed by the package. Section 3 describes the implementation of the package. The technical documentation contains only a limited number of tutorials and code examples. You can find more of these in the user manual.³

```

1 local metadata = {
2   version   = "(((VERSION)))",
3   comment   = "A module for the conversion from markdown "
4             .. "to plain TeX",
5   author    = "John MacFarlane, Hans Hagen, Vít Starý Novotný, "
6             .. "Andrej Genčur",
7   copyright = {"2009-2016 John MacFarlane, Hans Hagen",
8             "2016-2024 Vít Starý Novotný, Andrej Genčur"},
9   license    = "LPPL 1.3c"
10 }
11
12 if not modules then modules = { } end
13 modules['markdown'] = metadata

```

1.1 Requirements

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

1.1.1 Lua Requirements

The Lua part of the package requires that the following Lua modules are available from within the LuaTeX engine (though not necessarily in the LuaMetaTeX engine).

LPeg ≥ 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 ≥ 0.10 is included in LuaTeX $\geq 0.72.0$ (TeX Live ≥ 2013).

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

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

```
15 local md5 = require("md5")
```

Kpathsea A package that implements the loading of third-party Lua libraries and looking up files in the TeX directory structure.

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

```
16 ;(function()
```

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

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

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

lua-uni-algos A package that implements Unicode case-folding in TeX Live ≥ 2020 .

```
24 hard lua-uni-algos
25 local uni_algos = require("lua-uni-algos")
```

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

```
26 hard lua-tinyyaml
```

1.1.2 Plain TeX Requirements

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

expl3 A package that enables the expl3 language [2] from the L^AT_EX3 kernel in TeX Live ≤ 2019 . It is used to implement reflection capabilities that allow us to enumerate and inspect high-level concepts such as options, renderers, and renderer prototypes.

```
27 hard l3kernel
28 \unprotect
29 \expandafter\ifx\csname ExplSyntaxOn\endcsname\relax
30   \input expl3-generic
31 \fi
```

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

```
32 hard lt3luabridge
```

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

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

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

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

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

1.1.3 L^ATeX Requirements

The L^ATeX part of the package requires that the L^ATeX 2_ε format is loaded, a TeX engine that extends ϵ -TeX, and all the plain TeX prerequisites (see Section 1.1.2).

```
33 \NeedsTeXFormat{LaTeX2e}
34 \RequirePackage{expl3}
```

The following packages are soft prerequisites. They are only used to provide default token renderer prototypes (see sections 2.2.6 and 3.3.4) or L^ATeX themes (see Section 2.3.4) and will not be loaded if the option **plain** has been enabled (see Section 2.2.2.3):

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

```
35 soft url
```

graphicx A package that provides the **\includegraphics** macro for the typesetting of images. Furthermore, it also provides a key–value interface that is used in the default renderer prototypes for image attribute contexts.

36 `soft graphics`

enumitem and paralist Packages that provide macros for the default renderer prototypes for tight and fancy lists.

The package `paralist` will be used unless the option `experimental` has been enabled, in which case, the package `enumitem` will be used. Furthermore, enabling any test phase [3] will also cause `enumitem` to be used. In a future major version, `enumitem` will replace `paralist` altogether.

37 `soft enumitem`

38 `soft paralist`

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

39 `soft fancyvrb`

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

40 `soft csvsimple`

41 `soft pgf` # required by ``csvsimple``, which loads ``pgfkeys.sty``

42 `soft tools` # required by ``csvsimple``, which loads ``shellesc.sty``

43 `soft etoolbox` # required by ``csvsimple``, which loads ``etoolbox.sty``

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

44 `soft amsmath`

45 `soft amssymb`

graphicx A package that provides extended support for graphics. It is used in the `witiko/diagrams`, and `witiko/graphicx/http` plain T_EX themes, see Section 2.2.3.

46 `soft graphics`

47 `soft epstopdf` # required by ``graphics`` and ``graphicx``, which load ``epsopdf-base.sty``

48 `soft epstopdf-pkg` # required by ``graphics`` and ``graphicx``, which load ``epsopdf-base.sty``

soul and xcolor Packages that are used in the default renderer prototypes for strikeouts and marked text in pdfT_EX.

49 `soft soul`

50 `soft xcolor`

lua-ul and luacolor Packages that are used in the default renderer prototypes for strike-throughs and marked text in Lua \TeX .

```
51 soft lua-ul
52 soft luacolor
```

ltxcmds A package that is used to detect whether the minted and listings packages are loaded in the default renderer prototype for fenced code blocks.

```
53 soft ltxcmds
```

luaxml A package that is used to convert HTML to \LaTeX in the default renderer prototypes for content blocks, raw blocks, and inline raw spans.

```
54 soft luaxml
```

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

```
55 soft verse
```

1.1.4 Con \TeX t Prerequisites

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

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

1.2 Feedback

Please use the Markdown project page on 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 \TeX - \LaTeX Stack Exchange.⁵ community question answering web site under the `markdown` tag.

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

⁵See <https://tex.stackexchange.com>.

1.3 Acknowledgements

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

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

Funding by the Faculty of Informatics at the Masaryk University in Brno [4] 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 L^AT_EX 2_ε, 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.

2 Interfaces

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

Since neither T_EX nor Lua provide interfaces as a language construct, the separation to interfaces and implementations is a *gentlemen's agreement*. It serves as a means of structuring this documentation and as a promise to the user that if they only access the package through the interface, the future minor versions of the package should remain backwards compatible.

Figure 1 shows the high-level structure of the Markdown package: The translation from markdown to T_EX *token renderers* is exposed by the Lua layer. The plain T_EX layer exposes the conversion capabilities of Lua as T_EX macros. The L^AT_EX and ConT_EXt layers provide syntactic sugar on top of plain T_EX macros. The user can interface with any and all layers.

2.1 Lua Interface

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

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

```
56 local M = {metadata = metadata}
```

2.1.1 Conversion from Markdown to Plain T_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`

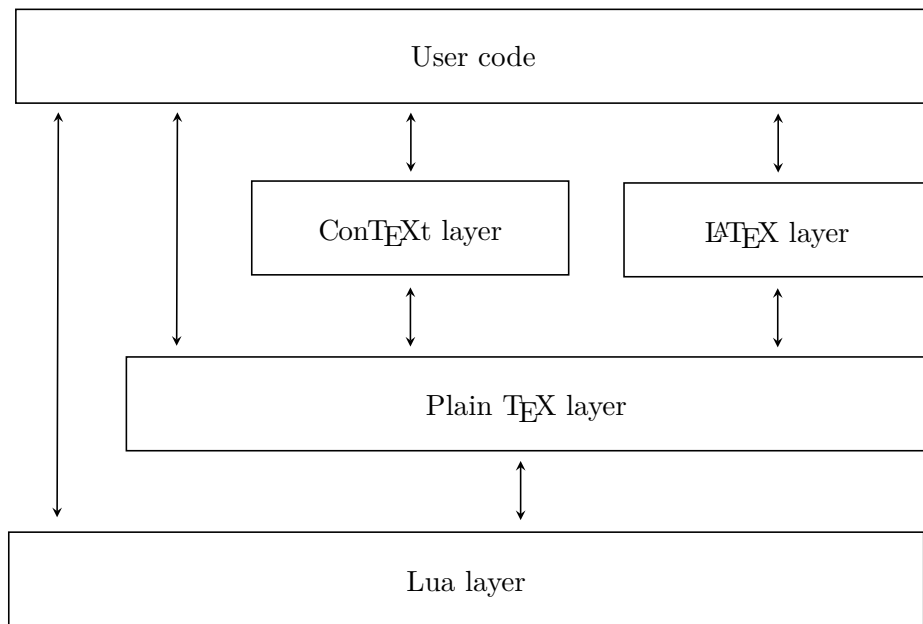


Figure 1: A block diagram of the Markdown package

that contains options recognized by the Lua interface (see Section 2.1.3). The `options` parameter is optional; when unspecified, the behaviour will be the same as if `options` were an empty table.

The following example Lua code converts the markdown string `Hello *world*!` to a 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.

```

57 local walkable_syntax = {

```



```

58   Block = {
59       "Blockquote",
60       "Verbatim",
61       "ThematicBreak",
62       "BulletList",
63       "OrderedList",
64       "DisplayHtml",
65       "Heading",
66   },
67   BlockOrParagraph = {
68       "Block",
69       "Paragraph",
70       "Plain",
71   },
72   Inline = {
73       "Str",
74       "Space",
75       "Endline",
76       "EndlineBreak",
77       "LinkAndEmph",
78       "Code",
79       "AutoLinkUrl",
80       "AutoLinkEmail",
81       "AutoLinkRelativeReference",
82       "InlineHtml",
83       "HtmlEntity",
84       "EscapedChar",
85       "Smart",
86       "Symbol",
87   },
88 }

```

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

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

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

2.1.3 Options

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

```
89 local defaultOptions = {}
```

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

```
90 \ExplSyntaxOn
91 \seq_new:N \g_@@_lua_options_seq
```

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

```
92 \prop_new:N \g_@@_lua_option_types_prop
93 \prop_new:N \g_@@_default_lua_options_prop
94 \seq_new:N \g_@@_option_layers_seq
95 \tl_const:Nn \c_@@_option_layer_lua_tl { lua }
96 \seq_gput_right:NV
97   \g_@@_option_layers_seq
98   \c_@@_option_layer_lua_tl
99 \cs_new:Nn
100   \@@_add_lua_option:nnn
101   {
102     \@@_add_option:Vnnn
103     \c_@@_option_layer_lua_tl
104     { #1 }
105     { #2 }
106     { #3 }
107   }
108 \cs_new:Nn
109   \@@_add_option:nnnn
110   {
111     \seq_gput_right:cn
112       { g_@@_ #1 _options_seq }
113       { #2 }
114     \prop_gput:cnn
115       { g_@@_ #1 _option_types_prop }
116       { #2 }
117       { #3 }
118     \prop_gput:cnn
119       { g_@@_default_ #1 _options_prop }
120       { #2 }
121       { #4 }
122     \@@_typecheck_option:n
123       { #2 }
124   }
```

```

125 \cs_generate_variant:Nn
126   \@@_add_option:nnnn
127   { Vnnn }
128 \tl_const:Nn \c_@@_option_value_true_tl { true }
129 \tl_const:Nn \c_@@_option_value_false_tl { false }
130 \cs_new:Nn \@@_typecheck_option:n
131   {
132     \@@_get_option_type:nN
133     { #1 }
134     \l_tmpa_tl
135     \str_case_e:Vn
136     \l_tmpa_tl
137     {
138       { \c_@@_option_type_boolean_tl }
139       {
140         \@@_get_option_value:nN
141         { #1 }
142         \l_tmpa_tl
143         \bool_if:nF
144         {
145           \str_if_eq_p:VV
146           \l_tmpa_tl
147           \c_@@_option_value_true_tl ||
148           \str_if_eq_p:VV
149           \l_tmpa_tl
150           \c_@@_option_value_false_tl
151         }
152         {
153           \msg_error:nnnV
154           { markdown }
155           { failed-typecheck-for-boolean-option }
156           { #1 }
157           \l_tmpa_tl
158         }
159       }
160     }
161   }
162 \msg_new:nnn
163   { markdown }
164   { failed-typecheck-for-boolean-option }
165   {
166     Option~#1~has~value~#2,~
167     but~a~boolean~(true~or~false)~was~expected.
168   }
169 \cs_generate_variant:Nn
170   \str_case_e:nn
171   { Vn }

```

```

172 \cs_generate_variant:Nn
173   \msg_error:nnnn
174   { nnnV }
175 \seq_new:N
176   \g_@@_option_types_seq
177 \tl_const:Nn
178   \c_@@_option_type_clist_tl
179   { clist }
180 \seq_gput_right:NV
181   \g_@@_option_types_seq
182   \c_@@_option_type_clist_tl
183 \tl_const:Nn
184   \c_@@_option_type_counter_tl
185   { counter }
186 \seq_gput_right:NV
187   \g_@@_option_types_seq
188   \c_@@_option_type_counter_tl
189 \tl_const:Nn
190   \c_@@_option_type_boolean_tl
191   { boolean }
192 \seq_gput_right:NV
193   \g_@@_option_types_seq
194   \c_@@_option_type_boolean_tl
195 \tl_const:Nn
196   \c_@@_option_type_number_tl
197   { number }
198 \seq_gput_right:NV
199   \g_@@_option_types_seq
200   \c_@@_option_type_number_tl
201 \tl_const:Nn
202   \c_@@_option_type_path_tl
203   { path }
204 \seq_gput_right:NV
205   \g_@@_option_types_seq
206   \c_@@_option_type_path_tl
207 \tl_const:Nn
208   \c_@@_option_type_slice_tl
209   { slice }
210 \seq_gput_right:NV
211   \g_@@_option_types_seq
212   \c_@@_option_type_slice_tl
213 \tl_const:Nn
214   \c_@@_option_type_string_tl
215   { string }
216 \seq_gput_right:NV
217   \g_@@_option_types_seq
218   \c_@@_option_type_string_tl

```

```

219 \cs_new:Nn
220   \@@_get_option_type:nN
221   {
222     \bool_set_false:N
223       \l_tmpa_bool
224     \seq_map_inline:Nn
225       \g_@@_option_layers_seq
226       {
227         \prop_get:cnNT
228           { g_@@_ ##1 _option_types_prop }
229           { #1 }
230         \l_tmpa_tl
231         {
232           \bool_set_true:N
233             \l_tmpa_bool
234           \seq_map_break:
235         }
236       }
237     \bool_if:NF
238       \l_tmpa_bool
239       {
240         \msg_error:nnn
241           { markdown }
242           { undefined-option }
243           { #1 }
244       }
245     \seq_if_in:NVF
246       \g_@@_option_types_seq
247       \l_tmpa_tl
248       {
249         \msg_error:nnnV
250           { markdown }
251           { unknown-option-type }
252           { #1 }
253         \l_tmpa_tl
254       }
255     \tl_set_eq:NN
256       #2
257       \l_tmpa_tl
258   }
259 \msg_new:nnn
260   { markdown }
261   { unknown-option-type }
262   {
263     Option~#1~has~unknown~type~#2.
264   }
265 \msg_new:nnn

```

```

266 { markdown }
267 { undefined-option }
268 {
269     Option~#1~is~undefined.
270 }
271 \cs_new:Nn
272   \@@_get_default_option_value:nN
273   {
274     \bool_set_false:N
275       \l_tmpa_bool
276     \seq_map_inline:Nn
277       \g_@@_option_layers_seq
278       {
279         \prop_get:cnNT
280           { g_@@_default_ ##1 _options_prop }
281           { #1 }
282         #2
283         {
284           \bool_set_true:N
285             \l_tmpa_bool
286           \seq_map_break:
287         }
288       }
289     \bool_if:NF
290       \l_tmpa_bool
291       {
292         \msg_error:nnn
293           { markdown }
294           { undefined-option }
295           { #1 }
296       }
297   }
298 \cs_new:Nn
299   \@@_get_option_value:nN
300   {
301     \@@_option_tl_to_csname:nN
302       { #1 }
303     \l_tmpa_tl
304     \cs_if_free:cTF
305       { \l_tmpa_tl }
306       {
307         \@@_get_default_option_value:nN
308           { #1 }
309         #2
310       }
311     {
312       \@@_get_option_type:nN

```

```

313         { #1 }
314         \l_tmpa_tl
315     \str_if_eq:NNTF
316     \c_@@_option_type_counter_tl
317     \l_tmpa_tl
318     {
319         \@@_option_tl_to_csname:nN
320         { #1 }
321         \l_tmpa_tl
322         \tl_set:Nx
323         #2
324         { \the \cs:w \l_tmpa_tl \cs_end: } % noqa: W200
325     }
326     {
327         \@@_option_tl_to_csname:nN
328         { #1 }
329         \l_tmpa_tl
330         \tl_set:Nv
331         #2
332         { \l_tmpa_tl }
333     }
334 }
335 }
336 \cs_new:Nn \@@_option_tl_to_csname:nN
337 {
338     \tl_set:Nn
339     \l_tmpa_tl
340     { \str_uppercase:n { #1 } }
341     \tl_set:Nx
342     #2
343     {
344         markdownOption
345         \tl_head:f { \l_tmpa_tl }
346         \tl_tail:n { #1 }
347     }
348 }

```

To make it easier to support different coding styles in the interface, engines, we define the `\@@_with_various_cases:nn` function that allows us to generate different variants of a string using different cases.

```

349 \cs_new:Nn \@@_with_various_cases:nn
350 {
351     \seq_clear:N
352     \l_tmpa_seq
353     \seq_map_inline:Nn
354     \g_@@_cases_seq
355     {

```

```

356         \tl_set:Nn
357         \l_tmpa_tl
358         { #1 }
359         \use:c { ##1 }
360         \l_tmpa_tl
361         \seq_put_right:NV
362         \l_tmpa_seq
363         \l_tmpa_tl
364     }
365     \seq_map_inline:Nn
366     \l_tmpa_seq
367     { #2 }
368 }

```

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

```

369 \cs_new:Nn \@@_with_various_cases_break:
370 {
371     \seq_map_break:
372 }

```

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

```

373 \seq_new:N \g_@@_cases_seq
374 \cs_new:Nn \@@_camel_case:N
375 {
376     \regex_replace_all:nnN
377     { _ ([a-z]) }
378     { \c { str_uppercase:n } \cB\{ \1 \cE\} }
379     #1
380     \tl_set:Nx
381     #1
382     { #1 }
383 }
384 \seq_gput_right:Nn \g_@@_cases_seq { @@_camel_case:N }
385 \cs_new:Nn \@@_snake_case:N
386 {
387     \regex_replace_all:nnN
388     { ([a-z])([A-Z]) }
389     { \1 _ \c { str_lowercase:n } \cB\{ \2 \cE\} }
390     #1
391     \tl_set:Nx
392     #1
393     { #1 }
394 }
395 \seq_gput_right:Nn \g_@@_cases_seq { @@_snake_case:N }

```


2.1.4 General Behavior

`eagerCache=true, false`

default: `true`

`true` Converted markdown documents will be cached in `cacheDir`. This can be useful for post-processing the converted documents and for recovering historical versions of the documents from the cache. Furthermore, it can also significantly improve the processing speed for documents that require multiple compilation runs, since each markdown document is only converted once. However, it also produces a large number of auxiliary files on the disk and obscures the output of the Lua command-line interface when it is used for plumbing.

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

`false` Converted markdown documents will not be cached. This decreases the number of auxiliary files that we produce and makes it easier to use the Lua command-line interface for plumbing. However, it makes it impossible to post-process the converted documents and recover historical versions of the documents from the cache. Furthermore, it can significantly reduce the processing speed for documents that require multiple compilation runs, since each markdown document is converted multiple times needlessly.

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

```
396 \@@_add_lua_option:nnn
397   { eagerCache }
398   { boolean }
399   { true }
400 defaultOptions.eagerCache = true
```

`experimental=true, false`

default: `false`

`true` Experimental features that are planned to be the new default in the next major release of the Markdown package will be enabled.

At the moment, this just means that the version `experimental` of the theme `witiko/markdown/defaults` will be loaded and warnings for hard-deprecated features will become errors. However, the effects may extend to other areas in the future as well.

`false` Experimental features will be disabled.

```

401 \@@_add_lua_option:nnn
402   { experimental }
403   { boolean }
404   { false }
405 defaultOptions.experimental = false

```

`singletonCache=true, false`

default: `true`

- true** Conversion functions produced by the function `new(options)` will be cached in an LRU cache of size 1 keyed by `options`. This is more time- and space-efficient than always producing a new conversion function but may expose bugs related to the idempotence of conversion functions. This has been the default behavior since version 3.0.0 of the Markdown package.
- false** Every call to the function `new(options)` will produce a new conversion function that will not be cached. This is slower than caching conversion functions and may expose bugs related to memory leaks in the creation of conversion functions, see also #226 (comment)⁶. This was the default behavior until version 3.0.0 of the Markdown package.

```

406 \@@_add_lua_option:nnn
407   { singletonCache }
408   { boolean }
409   { true }
410 defaultOptions.singletonCache = true
411 local singletonCache = {
412   convert = nil,
413   options = nil,
414 }

```

`unicodeNormalization=true, false`

default: `true`

- true** Markdown documents will be normalized using one of the four Unicode normalization forms⁷ before conversion. The Unicode normalization norm used is determined by option `unicodeNormalizationForm`.
- false** Markdown documents will not be Unicode-normalized before conversion.

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

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

```

415 \@@_add_lua_option:nnn
416   { unicodeNormalization }
417   { boolean }
418   { true }

419 defaultOptions.unicodeNormalization = true

```

`unicodeNormalizationForm=nfc, nfd, nfkc, nfkd`
 default: `nfc`

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

```

420 \@@_add_lua_option:nnn
421   { unicodeNormalizationForm }
422   { string }
423   { nfc }

424 defaultOptions.unicodeNormalizationForm = "nfc"

```

2.1.5 File and Directory Names

`cacheDir=⟨path⟩` default: `.`

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

```

425 \str_new:N
426   \g_@@_unquoted_jobname_str
427 \str_set:NV
428   \g_@@_unquoted_jobname_str
429   \c_sys_jobname_str
430 \regex_replace_all:nnN
431   { \A ("|') ( .* ) ("|') \Z }
432   { \2 }
433   \g_@@_unquoted_jobname_str
434 \@@_add_lua_option:nnn
435   { cacheDir }
436   { path }
437   {
438     \markdownOptionOutputDir
439     / _markdown_
440     \str_use:N
441     \g_@@_unquoted_jobname_str
442   }
443 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 2.2.5.9 for more information.

```

444 \@@_add_lua_option:nnn
445   { contentBlocksLanguageMap }
446   { path }
447   { markdown-languages.json }
448 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 3.1.7) and user-defined syntax extensions (see Section 2.1.2) have been applied.

```

449 \@@_add_lua_option:nnn
450   { debugExtensionsFileName }
451   { path }
452   {

```

```

453     \markdownOptionOutputDir
454     /
455     \str_use:N
456     \g_@@_unquoted_jobname_str
457     .debug-extensions.json
458   }

459 defaultOptions.debugExtensionsFileName = "debug-extensions.json"

```

frozenCacheFileName=*<path>* default: frozenCache.tex

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

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

```

460 \@@_add_lua_option:nnn
461   { frozenCacheFileName }
462   { path }
463   { \markdownOptionCacheDir / frozenCache.tex }

464 defaultOptions.frozenCacheFileName = "frozenCache.tex"

```

2.1.6 Parser Options

autoIdentifiers=true, false default: false

true Enable the Pandoc auto identifiers syntax extension⁸:

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

123 Sesame Street

false Disable the Pandoc auto identifiers syntax extension.

See also the option **gfmAutoIdentifiers**.

```

465 \@@_add_lua_option:nnn
466   { autoIdentifiers }
467   { boolean }
468   { false }

469 defaultOptions.autoIdentifiers = false

```

⁸See https://pandoc.org/MANUAL.html#extension-auto_identifiers.

`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.

```
470 \@@_add_lua_option:nnn
471 { blankBeforeBlockquote }
472 { boolean }
473 { false }

474 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.

```
475 \@@_add_lua_option:nnn
476 { blankBeforeCodeFence }
477 { boolean }
478 { false }

479 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.

```
480 \@@_add_lua_option:nnn
481 { blankBeforeDivFence }
482 { boolean }
483 { false }

484 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.

```
485 \@@_add_lua_option:nnn
486 { blankBeforeHeading }
487 { boolean }
488 { false }

489 defaultOptions.blankBeforeHeading = false
```

`blankBeforeList=true, false` default: false

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

```
490 \@@_add_lua_option:nnn
491 { blankBeforeList }
492 { boolean }
493 { false }

494 defaultOptions.blankBeforeList = false
```

`bracketedSpans=true, false` default: false

- true** Enable the Pandoc bracketed span syntax extension⁹:

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

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

```
495 \@@_add_lua_option:nnn
496 { bracketedSpans }
497 { boolean }
498 { false }

499 defaultOptions.bracketedSpans = false
```

⁹See https://pandoc.org/MANUAL.html#extension-bracketed_spans.

`breakableBlockquotes=true, false`

default: true

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

```
500 \@@_add_lua_option:nnn
501 { breakableBlockquotes }
502 { boolean }
503 { true }

504 defaultOptions.breakableBlockquotes = true
```

`citationNbsps=true, false`

default: false

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

```
505 \@@_add_lua_option:nnn
506 { citationNbsps }
507 { boolean }
508 { true }

509 defaultOptions.citationNbsps = true
```

`citations=true, false`

default: false

- `true` Enable the Pandoc citation syntax extension¹⁰:

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,

¹⁰See <https://pandoc.org/MANUAL.html#extension-citations>.


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

false Disable the Pandoc citation syntax extension.

```
510 \@@_add_lua_option:nnn
511   { citations }
512   { boolean }
513   { false }

514 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.``
```

```
515 \@@_add_lua_option:nnn
516   { codeSpans }
517   { boolean }
518   { true }

519 defaultOptions.codeSpans = true
```

contentBlocks=true, false

default: false

true

: Enable the iA Writer content blocks syntax extension [5]:

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

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

```
520 \@@_add_lua_option:nnn
521   { contentBlocks }
522   { boolean }
523   { false }

524 defaultOptions.contentBlocks = false
```

**contentLevel=**block, inline

default: block

**block**      Treat content as a sequence of blocks.

```
- this is a list
- it contains two items
```

**inline**      Treat all content as inline content.

```
- this is a text
- not a list
```

```
525 \@@_add_lua_option:nnn
526   { contentLevel }
527   { string }
528   { block }

529 defaultOptions.contentLevel = "block"
```

**debugExtensions=**true, false

default: false

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

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

```
530 \@@_add_lua_option:nnn
531   { debugExtensions }
532   { boolean }
533   { false }

534 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.

```
535 \@@_add_lua_option:nnn
536   { definitionLists }
537   { boolean }
538   { false }

539 defaultOptions.definitionLists = false
```

`ensureJekyllData=true, false`

default: false

**false** When the `jekyllData` and `expectJekyllData` options are enabled, then a markdown document may begin directly with YAML metadata and may contain nothing but YAML metadata. Otherwise, the markdown document is processed as markdown text.

**true** When the `jekyllData` and `expectJekyllData` options are enabled, then a markdown document must begin directly with YAML metadata and must contain nothing but YAML metadata. Otherwise, an error is produced.

```
540 \@@_add_lua_option:nnn
541   { ensureJekyllData }
542   { boolean }
543   { false }

544 defaultOptions.ensureJekyllData = false
```

`expectJekyllData=true, false`

default: `false`

`false`

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

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

`true`

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

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

```

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

```

```

545 \@@_add_lua_option:nnn
546   { expectJekyllData }
547   { boolean }
548   { false }

549 defaultOptions.expectJekyllData = false

```

`extensions=<filenames>`

The filenames of user-defined syntax extensions that will be applied to the markdown reader. If the kpathsea library is available, files will be searched for not only in the current working directory but also in the T<sub>E</sub>X directory structure.

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

```

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

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

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

```

```
return strike_through
```

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

```
550 metadata.user_extension_api_version = 2
551 metadata.grammar_version = 4
```

Any changes to the syntax extension API or grammar will cause the corresponding current version to be incremented. After Markdown 3.0.0, any changes to the API and the grammar will be either backwards-compatible or constitute a breaking change that will cause the major version of the Markdown package to increment (to 4.0.0).

The `finalize_grammar` field is a function that finalizes the grammar of markdown using the interface of a Lua `reader` object, such as the `reader->insert_pattern` and `reader->add_special_character` methods, see Section 2.1.2.

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

`fancyLists=true, false`

default: `false`

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

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

`false` Disable the Pandoc fancy list syntax extension.

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

---

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

`fencedCode=true, false`

default: `true`

`true` Enable the commonmark fenced code block extension:

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

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

`false` Disable the commonmark fenced code block extension.

```
565 \@@_add_lua_option:nnn
566   { fencedCode }
567   { boolean }
568   { true }

569 defaultOptions.fencedCode = true
```

`fencedCodeAttributes=true, false`

default: `false`

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

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

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

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

```

570 \@@_add_lua_option:nnn
571   { fencedCodeAttributes }
572   { boolean }
573   { false }

574 defaultOptions.fencedCodeAttributes = false

```

**fencedDivs**=true, false

default: false

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

```

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

And another.
:::::

```

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

```

575 \@@_add_lua_option:nnn
576   { fencedDivs }
577   { boolean }
578   { false }

579 defaultOptions.fencedDivs = false

```

**finalizeCache**=true, false

default: false

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

The frozen cache makes it possible to later typeset a plain T<sub>E</sub>X document that contains markdown documents without invoking Lua using the **frozenCache** plain T<sub>E</sub>X option. As a result, the plain T<sub>E</sub>X document becomes more portable, but further changes in the order and the content of markdown documents will not be reflected.

```

580 \@@_add_lua_option:nnn
581   { finalizeCache }
582   { boolean }
583   { false }

584 defaultOptions.finalizeCache = false

```

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



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

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

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

```
585 \@@_add_lua_option:nnn
586   { frozenCacheCounter }
587   { counter }
588   { 0 }

589 defaultOptions.frozenCacheCounter = 0
```

`gfmAutoIdentifiers`=true, false default: false

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

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

```
# 123 Sesame Street
```

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

See also the option `autoIdentifiers`.

```
590 \@@_add_lua_option:nnn
591   { gfmAutoIdentifiers }
592   { boolean }
593   { false }

594 defaultOptions.gfmAutoIdentifiers = false
```

`hashEnumerators`=true, false default: false

**true** Enable the use of hash symbols (#) as ordered item list markers:

```
#. Bird
#. McHale
#. Parish
```

**false** Disable the use of hash symbols (#) as ordered item list markers.

---

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

```

595 \@@_add_lua_option:nnn
596   { hashEnumerators }
597   { boolean }
598   { false }

599 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.

```

600 \@@_add_lua_option:nnn
601   { headerAttributes }
602   { boolean }
603   { false }

604 defaultOptions.headerAttributes = false

```

`html=true, false`

default: true

**true** Enable the recognition of inline HTML tags, block HTML elements, HTML comments, HTML instructions, and entities in the input. Inline HTML tags, block HTML elements and HTML comments will be rendered, HTML instructions will be ignored, and HTML entities will be replaced with the corresponding Unicode codepoints.

**false** Disable the recognition of HTML markup. Any HTML markup in the input will be rendered as plain text.

```

605 \@@_add_lua_option:nnn
606   { html }
607   { boolean }
608   { true }

609 defaultOptions.html = true

```

`hybrid=true, false`

default: `false`

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

The `hybrid` option makes it difficult to untangle TeX input from markdown text, which makes documents written with the `hybrid` option less interoperable and more difficult to read for authors. Therefore, the option has been soft-deprecated in version 3.7.1 of the Markdown package: It will never be removed but using it prints a warning and is discouraged.

Consider one of the following better alternatives for mixing TeX and markdown:

- With the `contentBlocks` option, authors can move large blocks of TeX code to separate files and include them in their markdown documents as external resources:

```
Here is a mathematical formula:  
  
/math-formula.tex
```

- With the `rawAttribute` option, authors can denote raw text spans and code blocks that will be interpreted as TeX code:

```
`$H_2 O$`{=tex} is a liquid.  
  
Here is a mathematical formula:  
... {=tex}  
\[distance[i] =  
    \begin{dcases}  
        a & b \\  
        c & d  
    \end{dcases}  
\]  
...
```

- With options `texMathDollars`, `texMathSingleBackslash`, and `texMathDoubleBackslash`, authors can freely type T<sub>E</sub>X commands between dollar signs or backslash-escaped brackets:

```
$H_2 O$ is a liquid.

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

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

`inlineCodeAttributes=true, false` default: false

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

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

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

```
615 \@@_add_lua_option:nnn
616   { inlineCodeAttributes }
617   { boolean }
618   { false }
619 defaultOptions.inlineCodeAttributes = false
```

`inlineNotes=true, false` default: false

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

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

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

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

`false`      Disable the Pandoc inline note syntax extension.

```
620 \@@_add_lua_option:nnn
621   { inlineNotes }
622   { boolean }
623   { false }
624 defaultOptions.inlineNotes = false
```

`jeekyllData=true, false`      default: false

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

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

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

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

```
625 \@@_add_lua_option:nnn
626   { jeekyllData }
627   { boolean }
628   { false }
629 defaultOptions.jekyllData = false
```

`linkAttributes=true, false`      default: false

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

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

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

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

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

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

```
630 \@@_add_lua_option:nnn
631   { linkAttributes }
632   { boolean }
633   { false }

634 defaultOptions.linkAttributes = false
```

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

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

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

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

```
635 \@@_add_lua_option:nnn
636   { lineBlocks }
637   { boolean }
638   { false }

639 defaultOptions.lineBlocks = false
```

**mark=true, false**      default: false

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

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

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

```
640 \@@_add_lua_option:nnn
641   { mark }
642   { boolean }
643   { false }

644 defaultOptions.mark = false
```

---

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

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

`notes=true, false`

default: false

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

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

[^1]: Here is the note.

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

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

        { some.code }

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

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

`false` Disable the Pandoc note syntax extension.

```
645 \@@_add_lua_option:nnn
646   { notes }
647   { boolean }
648   { false }

649 defaultOptions.notes = false
```

`pipeTables=true, false`

default: false

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

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

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

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

```

650 \@@_add_lua_option:nnn
651   { pipeTables }
652   { boolean }
653   { false }

654 defaultOptions.pipeTables = false

```

`preserveTabs=true, false`

default: true

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

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

```

655 \@@_add_lua_option:nnn
656   { preserveTabs }
657   { boolean }
658   { true }

659 defaultOptions.preserveTabs = true

```

`rawAttribute=true, false`

default: false

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

```

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

```

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

```

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

```

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

**false**     Disable the Pandoc raw attribute syntax extension.

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



```

660 \@@_add_lua_option:nnn
661   { rawAttribute }
662   { boolean }
663   { false }

664 defaultOptions.rawAttribute = false

```

`relativeReferences=true, false`

default: false

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

I conclude in Section <#conclusion>.

**Conclusion {#conclusion}**

=====

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

`false` Disable relative references in autolinks.

```

665 \@@_add_lua_option:nnn
666   { relativeReferences }
667   { boolean }
668   { false }

669 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.

```

670 \@@_add_lua_option:nnn
671   { shiftHeadings }
672   { number }
673   { 0 }

674 defaultOptions.shiftHeadings = 0

```

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

`slice`=*<the beginning and the end of a slice>* default: `^ $`

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

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

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

```
675 \@@_add_lua_option:nnn
676   { slice }
677   { slice }
678   { ^~$ }
679 defaultOptions.slice = "^ $"
```

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

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

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

```
680 \@@_add_lua_option:nnn
681   { smartEllipses }
682   { boolean }
683   { false }
684 defaultOptions.smartEllipses = false
```

`startNumber`=`true, false` default: `true`

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

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

```
685 \@@_add_lua_option:nnn
686 { startNumber }
687 { boolean }
688 { true }
689 defaultOptions.startNumber = true
```

**strikeThrough**=true, false default: false

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

This ~~is deleted text.~~

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

```
690 \@@_add_lua_option:nnn
691 { strikeThrough }
692 { boolean }
693 { false }
694 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.

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

---

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

`subscripts=true, false` default: false

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

H~2~0 is a liquid.

`false` Disable the Pandoc subscript syntax extension.

```
700 \@@_add_lua_option:nnn
701 { subscripts }
702 { boolean }
703 { false }

704 defaultOptions.subscripts = false
```

`superscripts=true, false` default: false

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

2^10^ is 1024.

`false` Disable the Pandoc superscript syntax extension.

```
705 \@@_add_lua_option:nnn
706 { superscripts }
707 { boolean }
708 { false }

709 defaultOptions.superscripts = false
```

`tableAttributes=true, false` default: false

`true`

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

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

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

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

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

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

```
710 \@@_add_lua_option:nnn
711   { tableAttributes }
712   { boolean }
713   { false }

714 defaultOptions.tableAttributes = false
```

**tableCaptions**=true, false default: false

**true**

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

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

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

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

```
715 \@@_add_lua_option:nnn
716 { tableCaptions }
717 { boolean }
718 { false }

719 defaultOptions.tableCaptions = false
```

**taskLists**=true, false default: false

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

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

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

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

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

```

720 \@@_add_lua_option:nnn
721 { taskLists }
722 { boolean }
723 { false }
724 defaultOptions.taskLists = false

```

`texComments=true, false`

default: false

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

```

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

```

Always enabled when `hybrid` is enabled.

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

```

725 \@@_add_lua_option:nnn
726 { texComments }
727 { boolean }
728 { false }
729 defaultOptions.texComments = false

```

`texMathDollars=true, false`

default: false

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

```

inline math: $E=mc^2$

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

```

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

```

730 \@@_add_lua_option:nnn
731 { texMathDollars }
732 { boolean }
733 { false }
734 defaultOptions.texMathDollars = false

```

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

`texMathDoubleBackslash=true, false` default: false

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

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

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

```
735 \@@_add_lua_option:nnn
736 { texMathDoubleBackslash }
737 { boolean }
738 { false }
739 defaultOptions.texMathDoubleBackslash = false
```

`texMathSingleBackslash=true, false` default: false

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

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

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

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

`tightLists=true, false` default: true

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

---

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

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

```

- This is
- a tight
- unordered list.

- This is

 not a tight

- unordered list.

```

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

```

745 \@@_add_lua_option:nnn
746 { tightLists }
747 { boolean }
748 { true }

749 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.

```

750 \@@_add_lua_option:nnn
751 { underscores }
752 { boolean }
753 { true }
754 \ExplSyntaxOff

755 defaultOptions.underscores = true

```

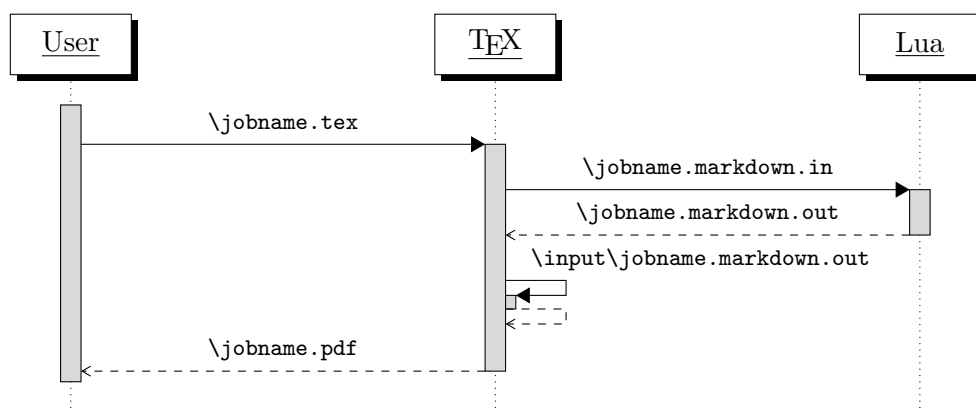


### 2.1.7 Command-Line Interface

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

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

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

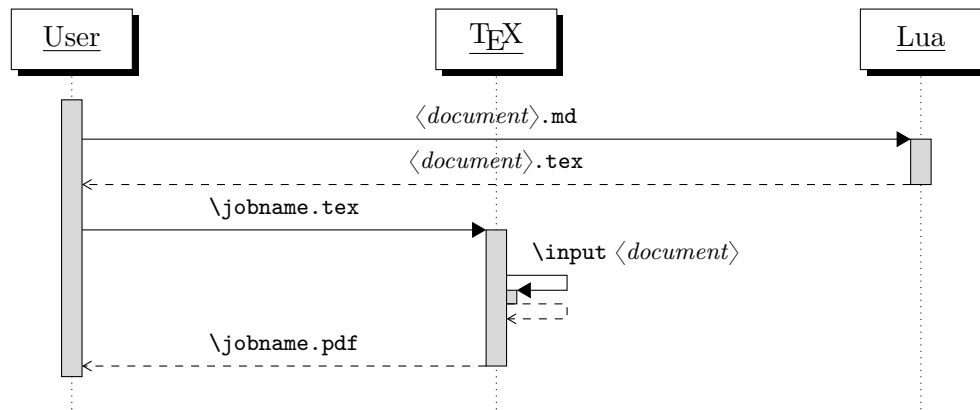


**Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the  $\text{\TeX}$  interface**

```

756 .TH MARKDOWN2TEX 1 "(((LASTMODIFIED)))"
757 .SH NAME
758 markdown2tex \- convert .md files to .tex
759 .SH SYNOPSIS
</lua-cli-manpage> <*lua-cli>
760 local HELP_STRING = "Usage: " .. [[
</lua-cli> <*lua-cli,lua-cli-manpage>
761 markdown2tex [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
762
</lua-cli,lua-cli-manpage> <*lua-cli-manpage>
763 .SH DESCRIPTION

```



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

```

764 % \end{macrocode}
765 </lua-cli-manpage>
766 <*lua-cli, lua-cli-manpage>
767 % \begin{macrocode}
768 OPTIONS are documented in Section 2.2.1 of the Markdown Package User
769 Manual (https://ctan.org/pkg/markdown).
770
771 When OUTPUT_FILE is unspecified, the result of the conversion will be
772 written to the standard output. When INPUT_FILE is also unspecified, the
773 result of the conversion will be read from the standard input.
774 % \end{macrocode}
775 </lua-cli, lua-cli-manpage>
776 <*lua-cli>
777 % \begin{macrocode}
778
779 Report bugs to: witiko@mail.muni.cz
780 Markdown package home page: <https://github.com/witiko/markdown>]]
781
782 local VERSION_STRING = [[
783 markdown2tex (Markdown)]] .. metadata.version .. [[
784
785 Copyright (C)]] .. table.concat(metadata.copyright,
786 "\nCopyright (C) ") .. [[
787
788 License:]] .. metadata.license
789
790 local function warn(s)
791 io.stderr:write("Warning: " .. s .. "\n")
792 end

```

```

793
794 local function error(s)
795 io.stderr:write("Error: " .. s .. "\n")
796 os.exit(1)
797 end

```

To make it easier to copy-and-paste options from Pandoc [6] 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 [7] also show that snake\_case is faster to read than camelCase.

```

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

```

826 if arg[i] == "--" then
827 process_options = false
828 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.

```

829 elseif arg[i]:match("=") then
830 local key, value = arg[i]:match("(.-)=(.*)")
831 if defaultOptions[key] == nil and
832 various_case_options[key] ~= nil then
833 key = various_case_options[key]
834 end

```

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

```

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

```

859 elseif arg[i] == "--help" or arg[i] == "-h" then
860 print(HELP_STRING)
861 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.

```

862 elseif arg[i] == "--version" or arg[i] == "-v" then
863 print(VERSION_STRING)
864 os.exit()
865 end
866 end

```

The first argument that matches none of the above patterns is assumed to be the input filename. The input filename should correspond to the Markdown document that is going to be converted to a T<sub>E</sub>X document.

```

867 if input_filename == nil then
868 input_filename = arg[i]

```

The first argument that matches none of the above patterns is assumed to be the output filename. The output filename should correspond to the T<sub>E</sub>X document that will result from the conversion.

```

869 elseif output_filename == nil then
870 output_filename = arg[i]
871 else
872 error('Unexpected argument: "' .. arg[i] .. '".')
873 end
874 ::continue::
875 end

```

The command-line Lua interface is implemented by the files [markdown-cli.lua](#) and [markdown2tex.lua](#), which can be invoked from the command line as follows:

```
markdown2tex cacheDir=. -- hello.md hello.tex
```

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

```
\input hello
```

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

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

```

876 \def\markdownLastModified{((LASTMODIFIED))}%
877 \def\markdownVersion{((VERSION))}%

```

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

```
\input markdown
```

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

### 2.2.1 Typesetting Markdown and YAML

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

#### 2.2.1.1 Typesetting Markdown and YAML directly

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

```
878 \let\markdownBegin\relax
879 \let\markdownEnd\relax
```

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

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

Another limitation concerns spaces at the right end of an input line. In markdown, these are used to produce a forced line break. However, any such spaces are removed before the lines enter the input buffer of T<sub>E</sub>X [8, p. 46]. As a corollary, the `\markdownBegin` macro also ignores them.

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

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

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

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

```
\input markdown
\markdownBegin
```

```

Hello **world** ...
\markdownEnd
\bye

```

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

```

880 \let\uyamlBegin\uyamlBegin\relax
881 \def\uyamlEnd{\uyamlEnd\endgroup}

```

The `\yamlBegin` and `\yamlEnd` macros are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

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

```

\input markdown
\uyamlBegin
title: _Hello_ **world** ...
author: John Doe
\uyamlEnd
\bye

```

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

```

\input markdown
\uyamlSetup{jekyllData, expectJekyllData, ensureJekyllData}
\uyamlBegin
title: _Hello_ **world** ...
author: John Doe
\uyamlEnd
\bye

```

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

```

882 \let\markinline\relax

```

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

```

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

```

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

```

\input markdown
\markdownSetup{contentLevel=inline}
\markdownBegin
Hello world ...
\markdownEnd
\bye

```

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

### 2.2.1.2 Typesetting Markdown and YAML from external documents

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

```
883 \let\markdownInput\relax
```

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

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

```

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

```

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

```

884 \def\yamlInput#1{%
885 \begingroup
886 \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
887 \markdownInput{#1}%
888 \endgroup
889 }%

```

The macro `\yamlInput` is also not subject to the limitations of the `\markdownBegin` and `\markdownEnd` macros.

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



```

\input markdown
\yamlInput{hello.yml}
\bye

```

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

```

\input markdown
\yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}
\markdownInput{hello.yml}
\bye

```

### 2.2.1.3 Typesetting TeX from inside Markdown and YAML documents

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

```
890 \let\markdownEscape\relax
```

### 2.2.2 Options

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

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

```

891 \ExplSyntaxOn
892 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
893 \cs_generate_variant:Nn
894 \tl_const:Nn
895 { NV }
896 \tl_if_exist:NF
897 \c_@@_top_layer_tl
898 {
899 \tl_const:NV
900 \c_@@_top_layer_tl
901 \c_@@_option_layer_plain_tex_tl
902 }

```

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

```
903 \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.

```

904 \prop_new:N \g_@@_plain_tex_option_types_prop
905 \prop_new:N \g_@@_default_plain_tex_options_prop
906 \seq_gput_right:NV
907 \g_@@_option_layers_seq
908 \c_@@_option_layer_plain_tex_tl
909 \cs_new:Nn
910 \@@_add_plain_tex_option:nnn
911 {
912 \@@_add_option:Vnnn
913 \c_@@_option_layer_plain_tex_tl
914 { #1 }
915 { #2 }
916 { #3 }
917 }

```

The plain TeX options may be also be specified via the `\markdownSetup` macro. Here, the plain TeX options are represented by a comma-delimited list of `<key>=<value>` pairs. For boolean options, the `=<value>` part is optional, and `<key>` will be interpreted as `<key>=true` if the `=<value>` part has been omitted. The `\markdownSetup` macro receives the options to set up as its only argument.

```

918 \cs_new:Nn
919 \@@_setup:n
920 {
921 \keys_set:nn
922 { markdown/options }
923 { #1 }
924 }
925 \cs_gset_eq:NN
926 \markdownSetup
927 \@@_setup:n

```

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

```

928 \cs_gset_eq:NN
929 \yamlSetup
930 \markdownSetup

```

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

```

931 \prg_new_conditional:Nnn
932 \@@_if_option:n
933 { TF, T, F }
934 {

```

```

935 \@@_get_option_type:nN
936 { #1 }
937 \l_tmpa_tl
938 \str_if_eq:NNF
939 \l_tmpa_tl
940 \c_@@_option_type_boolean_tl
941 {
942 \msg_error:nnxx
943 { markdown }
944 { expected-boolean-option }
945 { #1 }
946 { \l_tmpa_tl }
947 }
948 \@@_get_option_value:nN
949 { #1 }
950 \l_tmpa_tl
951 \str_if_eq:NNTF
952 \l_tmpa_tl
953 \c_@@_option_value_true_tl
954 { \prg_return_true: }
955 { \prg_return_false: }
956 }
957 \msg_new:nnn
958 { markdown }
959 { expected-boolean-option }
960 {
961 Option~#1~has~type~#2,~
962 but~a~boolean~was~expected.
963 }
964 \let
965 \markdownIfOption
966 \@@_if_option:nTF

```

### 2.2.2.1 Finalizing and Freezing the Cache

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

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

```

967 \@@_add_plain_tex_option:nnn
968 { frozenCache }

```

```

969 { boolean }
970 { false }

```

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

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

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

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

```

971 \@@_add_plain_tex_option:nnn
972 { inputTempFileName }
973 { path }
974 {
975 \str_use:N
976 \g_@@_unquoted_jobname_str
977 .markdown.in
978 }

```

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

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

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

```

979 \@@_add_plain_tex_option:nnn
980 { outputDir }
981 { path }
982 { . }

```

### 2.2.2.3 No default token renderer prototypes

The Markdown package provides default definitions for token renderer prototypes using the `witiko/markdown/defaults` theme (see Section `sec:#themes`). Although

these default definitions provide a useful starting point for authors, they use extra resources, especially with higher-level T<sub>E</sub>X formats such as L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt. Furthermore, the default definitions may change at any time, which may pose a problem for maintainers of Markdown themes and templates who may require a stable output.

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

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

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

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

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

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

```
983 \@@_add_plain_tex_option:nnn
984 { plain }
985 { boolean }
986 { false }
```

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

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

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

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

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

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

```

987 \@@_add_plain_tex_option:nnn
988 { noDefaults }
989 { boolean }
990 { false }

```

#### 2.2.2.4 Miscellaneous Options

The `\markdownOptionStripPercentSigns` macro controls whether a percent sign (%) at the beginning of a line will be discarded when buffering Markdown input (see sections 3.2.5 and 3.2.6) or not. Notably, this enables the use of markdown when writing T<sub>E</sub>X package documentation using the Doc L<sup>A</sup>T<sub>E</sub>X package [9] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```

991 \seq_gput_right:Nn
992 \g_@@_plain_tex_options_seq
993 { stripPercentSigns }
994 \prop_gput:Nnn
995 \g_@@_plain_tex_option_types_prop
996 { stripPercentSigns }
997 { boolean }
998 \prop_gput:Nnx
999 \g_@@_default_plain_tex_options_prop
1000 { stripPercentSigns }
1001 { false }

```

#### 2.2.2.5 Generating Plain T<sub>E</sub>X Option Macros and Key-Values

We define the command `\@@_define_option_commands_and_keyvals:` that defines plain T<sub>E</sub>X macros and the key-value interface of the `\markdownSetup` macro for the above plain T<sub>E</sub>X options.

The command also defines macros and key-values that map directly to the options recognized by the Lua interface, such as `\markdownOptionHybrid` for the `hybrid` Lua option (see Section 2.1.3), which are not processed by the plain T<sub>E</sub>X implementation, only passed along to Lua.

Furthermore, the command also defines options and key-values for subsequently loaded layers that correspond to higher-level T<sub>E</sub>X formats such as L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt.

For the macros that correspond to the non-boolean options recognized by the Lua interface, the same limitations apply here in the case of the `inputTempFileName` macro.

```

1002 \cs_new:Nn
1003 \@@_define_option_commands_and_keyvals:
1004 {
1005 \seq_map_inline:Nn

```

```

1006 \g_@@_option_layers_seq
1007 {
1008 \seq_map_inline:cn
1009 { g_@@_ ##1 _options_seq }
1010 {
1011 \@@_define_option_command:n
1012 { #####1 }

```

To make it easier to copy-and-paste options from Pandoc [6] 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 [7] also show that `snake_case` is faster to read than camelCase.

```

1013 \@@_with_various_cases:nn
1014 { #####1 }
1015 {
1016 \@@_define_option_keyval:nnn
1017 { ##1 }
1018 { #####1 }
1019 { #####1 }
1020 }
1021 }
1022 }
1023 }
1024 \cs_new:Nn
1025 \@@_define_option_command:n
1026 {

```

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

```

1027 \str_if_eq:nnTF
1028 { #1 }
1029 { outputDir }
1030 { \@@_define_option_command_output_dir: }
1031 {

```

Do not override options defined before loading the package.

```

1032 \@@_option_tl_to_csname:nN
1033 { #1 }
1034 \l_tmpa_tl
1035 \cs_if_exist:cF
1036 { \l_tmpa_tl }
1037 {
1038 \@@_get_default_option_value:nN
1039 { #1 }
1040 \l_tmpa_tl
1041 \@@_set_option_value:nV
1042 { #1 }
1043 \l_tmpa_tl

```

```

1044 }
1045 }
1046 }
1047 \ExplSyntaxOff
1048 \input lt3luabridge.tex

```

Use the `lt3luabridge` library to determine the default value of the `\markdownOptionOutputDir` macro by using one of the following:

1. The `status.output_directory` variable [1, Section 10.2], which is available since LuaTeX 1.18.0 from TeX Live 2024 and in other TeX distributions like MikTeX since ca March 2024. We are only able to read this variable in LuaTeX and not other TeX engines.
2. The `TEXMF_OUTPUT_DIRECTORY` environmental variable, which is available since TeX Live 2024. We are only able to read this variable in TeX Live and not some other TeX distributions like MikTeX.

```

1049 \ExplSyntaxOn
1050 \cs_new:Nn
1051 \@@_define_option_command_output_dir:
1052 {
1053 \cs_if_free:NT
1054 \markdownOptionOutputDir
1055 {
1056 \bool_if:nTF
1057 {
1058 \cs_if_exist_p:N
1059 \luabridge_tl_set:Nn &&
1060 (
1061 \int_compare_p:nNn
1062 { \g_luabridge_method_int }
1063 =
1064 { \c_luabridge_method_directlua_int } ||
1065 \sys_if_shell_unrestricted_p:
1066)
1067 }
1068 {

```

Set most catcodes to category 12 (other) to ensure that special characters in the output directory name such as backslashes (`\`) are not interpreted as control sequences.

```

1069 \group_begin:
1070 \cctab_select:N
1071 \c_str_cctab
1072 \luabridge_tl_set:Nn
1073 \l_tmpa_tl
1074 {

```



```

1075 print(
1076 (status.output_directory)
1077 or~os.getenv("TEXMF_OUTPUT_DIRECTORY")
1078 or~"."
1079)
1080 }
1081 \tl_gset:NV
1082 \markdownOptionOutputDir
1083 \l_tmpa_tl
1084 \group_end:
1085 }
1086 {
1087 \tl_gset:Nn
1088 \markdownOptionOutputDir
1089 { . }
1090 }
1091 }
1092 }
1093 \cs_new:Nn
1094 \@@_set_option_value:nn
1095 {
1096 \@@_define_option:n
1097 { #1 }
1098 \@@_get_option_type:nN
1099 { #1 }
1100 \l_tmpa_tl
1101 \str_if_eq:NNTF
1102 \c_@@_option_type_counter_tl
1103 \l_tmpa_tl
1104 {
1105 \@@_option_tl_to_csname:nN
1106 { #1 }
1107 \l_tmpa_tl
1108 \int_gset:cn
1109 { \l_tmpa_tl }
1110 { #2 }
1111 }
1112 {
1113 \@@_option_tl_to_csname:nN
1114 { #1 }
1115 \l_tmpa_tl
1116 \cs_set:cpn
1117 { \l_tmpa_tl }
1118 { #2 }
1119 }
1120 }
1121 \cs_generate_variant:Nn

```

```

1122 \@@_set_option_value:nn
1123 { nV }
1124 \cs_new:Nn
1125 \@@_define_option:n
1126 {
1127 \@@_option_tl_to_csname:nN
1128 { #1 }
1129 \l_tmpa_tl
1130 \cs_if_free:cT
1131 { \l_tmpa_tl }
1132 {
1133 \@@_get_option_type:nN
1134 { #1 }
1135 \l_tmpb_tl
1136 \str_if_eq:NNT
1137 \c_@@_option_type_counter_tl
1138 \l_tmpb_tl
1139 {
1140 \@@_option_tl_to_csname:nN
1141 { #1 }
1142 \l_tmpa_tl
1143 \int_new:c
1144 { \l_tmpa_tl }
1145 }
1146 }
1147 }
1148 \cs_new:Nn
1149 \@@_define_option_keyval:nnn
1150 {
1151 \prop_get:cnN
1152 { g_@@_ #1 _option_types_prop }
1153 { #2 }
1154 \l_tmpa_tl
1155 \str_if_eq:VVTF
1156 \l_tmpa_tl
1157 \c_@@_option_type_boolean_tl
1158 {
1159 \keys_define:nn
1160 { markdown/options }
1161 {

```

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

```

1162 #3 .code:n = {
1163 \tl_set:Nx
1164 \l_tmpa_tl
1165 {

```

```

1166 \str_case:nnF
1167 { ##1 }
1168 {
1169 { yes } { true }
1170 { no } { false }
1171 }
1172 { ##1 }
1173 }
1174 \@@_set_option_value:nV
1175 { #2 }
1176 \l_tmpa_tl
1177 },
1178 #3 .default:n = { true },
1179 }
1180 }
1181 {
1182 \keys_define:nn
1183 { markdown/options }
1184 {
1185 #3 .code:n = {
1186 \@@_set_option_value:nn
1187 { #2 }
1188 { ##1 }
1189 },
1190 }
1191 }

```

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.

```

1192 \str_if_eq:VVT
1193 \l_tmpa_tl
1194 \c_@@_option_type_clist_tl
1195 {
1196 \tl_set:Nn
1197 \l_tmpa_tl
1198 { #3 }
1199 \tl_reverse:N
1200 \l_tmpa_tl
1201 \str_if_eq:enF
1202 {
1203 \tl_head:V
1204 \l_tmpa_tl
1205 }
1206 { s }

```

```

1207 {
1208 \msg_error:nnn
1209 { markdown }
1210 { malformed-name-for-clist-option }
1211 { #3 }
1212 }
1213 \tl_set:Nx
1214 \l_tmpa_tl
1215 {
1216 \tl_tail:V
1217 \l_tmpa_tl
1218 }
1219 \tl_reverse:N
1220 \l_tmpa_tl
1221 \tl_put_right:Nn
1222 \l_tmpa_tl
1223 {
1224 .code:n = {
1225 \@@_get_option_value:nN
1226 { #2 }
1227 \l_tmpa_tl
1228 \clist_set:NV
1229 \l_tmpa_clist
1230 { \l_tmpa_tl , { ##1 } }
1231 \@@_set_option_value:nV
1232 { #2 }
1233 \l_tmpa_clist
1234 }
1235 }
1236 \keys_define:nV
1237 { markdown/options }
1238 \l_tmpa_tl
1239 }
1240 }
1241 \cs_generate_variant:Nn
1242 \clist_set:Nn
1243 { NV }
1244 \cs_generate_variant:Nn
1245 \keys_define:nn
1246 { nV }
1247 \cs_generate_variant:Nn
1248 \@@_set_option_value:nn
1249 { nV }
1250 \prg_generate_conditional_variant:Nnn
1251 \str_if_eq:nn
1252 { en }
1253 { p, F }

```

```

1254 \msg_new:nnn
1255 { markdown }
1256 { malformed-name-for-clist-option }
1257 {
1258 Clist-option-name~#1~does~not~end~with~-s.
1259 }

```

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

```

1260 \str_if_eq:VVT
1261 \c_@@_top_layer_tl
1262 \c_@@_option_layer_plain_tex_tl
1263 {
1264 \@@_define_option_commands_and_keyvals:
1265 }
1266 \ExplSyntaxOff

```

### 2.2.3 Themes

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

The key-values `theme=<theme name>` and `import=<theme name>`, optionally followed by `@<theme version>`, load a  $\text{\TeX}$  document (further referred to as *a theme*) named `markdowntheme<munged theme name>.tex`, where the *munged theme name* is the *theme name* after the substitution of all forward slashes (/) for an underscore (\_). The theme name must be *qualified* and contain no underscores or at signs (@). Themes are inspired by the Beamer  $\text{\LaTeX}$  package, which provides similar functionality with its `\usetheme` macro [10, Section 15.1].

A theme name is qualified if and only if it contains at least one forward slash. Theme names must be qualified to minimize naming conflicts between different themes with a similar purpose. The preferred format of a theme name is `<theme author>/<theme purpose>/<private naming scheme>`, where the *private naming scheme* may contain additional forward slashes. For example, a theme by a user `witiko` for the MU theme of the Beamer document class may have the name `witiko/beamer/MU`.

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

If `@<theme version>` is specified after `<theme name>`, then the text *theme version* will be available in the macro `\markdownThemeVersion` when the theme is loaded. If `@<theme version>` is not specified, the macro `\markdownThemeVersion` will contain the text `latest` [11].

```

1267 \ExplSyntaxOn
1268 \keys_define:nn
1269 { markdown/options }
1270 {
1271 theme .code:n = {
1272 \@@_set_theme:n
1273 { #1 }
1274 },
1275 import .code:n = {
1276 \tl_set:Nn
1277 \l_tmpa_tl
1278 { #1 }

```

To ensure that keys containing forward slashes get passed correctly, we replace all forward slashes in the input with backslash tokens with category code letter and then undo the replacement. This means that if any unbraced backslash tokens with category code letter exist in the input, they will be replaced with forward slashes. However, this should be extremely rare.

```

1279 \tl_replace_all:NnV
1280 \l_tmpa_tl
1281 { / }
1282 \c_backslash_str
1283 \keys_set:nV
1284 { markdown/options/import }
1285 \l_tmpa_tl
1286 },
1287 }

```

To keep track of the current theme when themes are nested, we will maintain the stacks `\g_@@_theme_names_seq` and `\g_@@_theme_versions_seq` stack of theme names and versions, respectively. For convenience, the name of the current theme and version is also available in the macros `\g_@@_current_theme_tl` and `\markdownThemeVersion`, respectively.

```

1288 \seq_new:N
1289 \g_@@_theme_names_seq
1290 \seq_new:N
1291 \g_@@_theme_versions_seq
1292 \tl_new:N
1293 \g_@@_current_theme_tl
1294 \tl_gset:Nn
1295 \g_@@_current_theme_tl
1296 { }
1297 \seq_gput_right:NV
1298 \g_@@_theme_names_seq
1299 \g_@@_current_theme_tl
1300 \cs_new:Npn
1301 \markdownThemeVersion

```

```

1302 { }
1303 \seq_gput_right:NV
1304 \g_@@_theme_versions_seq
1305 \g_@@_current_theme_tl
1306 \cs_new:Nn
1307 \@@_set_theme:n
1308 {

```

First, we validate the theme name.

```

1309 \str_if_in:nnF
1310 { #1 }
1311 { / }
1312 {
1313 \msg_error:nnn
1314 { markdown }
1315 { unqualified-theme-name }
1316 { #1 }
1317 }
1318 \str_if_in:nnT
1319 { #1 }
1320 { _ }
1321 {
1322 \msg_error:nnn
1323 { markdown }
1324 { underscores-in-theme-name }
1325 { #1 }
1326 }

```

Next, we extract the theme version.

```

1327 \str_if_in:nnTF
1328 { #1 }
1329 { @ }
1330 {
1331 \regex_extract_once:nnN
1332 { (.*?) @ (.*?) }
1333 { #1 }
1334 \l_tmpa_seq
1335 \seq_gpop_left:NN
1336 \l_tmpa_seq
1337 \l_tmpa_tl
1338 \seq_gpop_left:NN
1339 \l_tmpa_seq
1340 \l_tmpa_tl
1341 \tl_gset:NV
1342 \g_@@_current_theme_tl
1343 \l_tmpa_tl
1344 \seq_gpop_left:NN
1345 \l_tmpa_seq

```

```

1346 \l_tmpa_tl
1347 \cs_gset:Npe
1348 \markdownThemeVersion
1349 {
1350 \tl_use:N
1351 \l_tmpa_tl
1352 }
1353 }
1354 {
1355 \tl_gset:Nn
1356 \g_@@_current_theme_tl
1357 { #1 }
1358 \cs_gset:Npn
1359 \markdownThemeVersion
1360 { latest }
1361 }

```

Next, we munge the theme name.

```

1362 \str_set:NV
1363 \l_tmpa_str
1364 \g_@@_current_theme_tl
1365 \str_replace_all:Nnn
1366 \l_tmpa_str
1367 { / }
1368 { _ }

```

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

```

1369 \tl_set:NV
1370 \l_tmpa_tl
1371 \g_@@_current_theme_tl
1372 \tl_put_right:Nn
1373 \g_@@_current_theme_tl
1374 { / }
1375 \seq_gput_right:NV
1376 \g_@@_theme_names_seq
1377 \g_@@_current_theme_tl
1378 \seq_gput_right:NV
1379 \g_@@_theme_versions_seq
1380 \markdownThemeVersion
1381 \@@_load_theme:VeV
1382 \l_tmpa_tl
1383 { \markdownThemeVersion }
1384 \l_tmpa_str

```

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

```

1385 \seq_gpop_right:NN

```



```

1386 \g_@@_theme_names_seq
1387 \l_tmpa_tl
1388 \seq_get_right:NN
1389 \g_@@_theme_names_seq
1390 \l_tmpa_tl
1391 \tl_gset:NV
1392 \g_@@_current_theme_tl
1393 \l_tmpa_tl
1394 \seq_gpop_right:NN
1395 \g_@@_theme_versions_seq
1396 \l_tmpa_tl
1397 \seq_get_right:NN
1398 \g_@@_theme_versions_seq
1399 \l_tmpa_tl
1400 \cs_gset:Npe
1401 \markdownThemeVersion
1402 {
1403 \tl_use:N
1404 \l_tmpa_tl
1405 }
1406 }
1407 \msg_new:nnnn
1408 { markdown }
1409 { unqualified-theme-name }
1410 { Won't load theme with unqualified name #1 }
1411 { Theme names must contain at least one forward slash }
1412 \msg_new:nnnn
1413 { markdown }
1414 { underscores-in-theme-name }
1415 { Won't load theme with an underscore in its name #1 }
1416 { Theme names must not contain underscores in their names }
1417 \cs_generate_variant:Nn
1418 \tl_replace_all:Nnn
1419 { NnV }
1420 \cs_generate_variant:Nn
1421 \cs_gset:Npn
1422 { Npe }

```

We also define the prop `\g_@@_plain_tex_built_in_themes_prop` that contains the code of built-in themes. This is a packaging optimization, so that built-in themes does not need to be distributed in many small files.

```

1423 \prop_new:N
1424 \g_@@_plain_tex_built_in_themes_prop

```

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

**witiko/diagrams** A theme that typesets fenced code blocks with the infostrings `dot`, `mermaid`, and `plantuml` as figures with diagrams produced with the

command `dot` from Graphviz tools, the command `mmdc` from the npm package `@mermaid-js/mermaid-cli`, and the command `plantuml` from the package PlantUML, respectively. The key-value attribute `caption` can be used to specify the caption of the figure. The remaining attributes are treated as image attributes.

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

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

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

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

```

        ((?))
        sub<br/>idea 4
        ((?))
    ...

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

 ' Diagram title
 title Simple Request-Response Flow

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

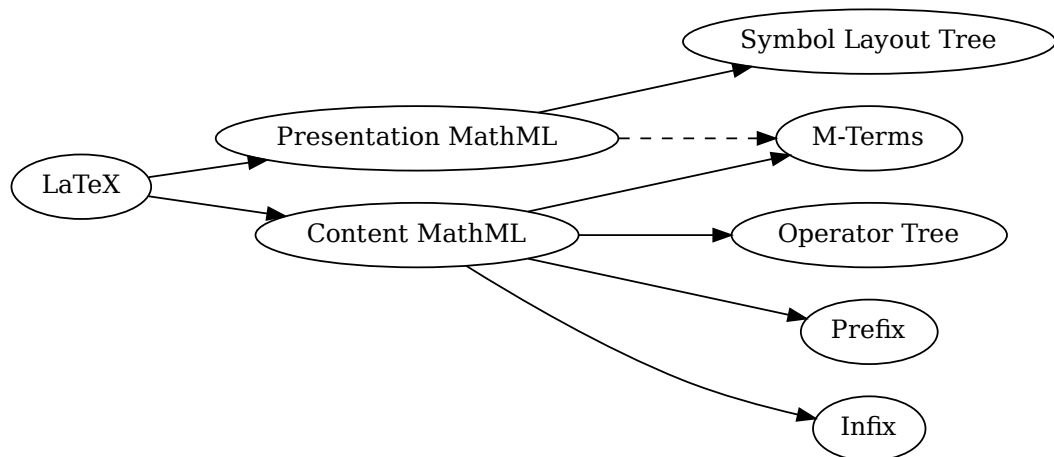
 alt Request is valid
 S -> DB: Query Data
 DB -> S: Return Data
 S -> C: Respond with Data
 else Request is invalid
 S -> C: Return Error
 end
 @enduml
 ...

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

```

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

The theme requires a Unix-like operating system with GNU Diffutils, Graphviz, the npm package [@mermaid-js/mermaid-cli](#), and PlantUML installed. All these packages are already included in the Docker image [witiko/markdown](#); consult [Dockerfile](#) to see how they are installed. The theme also requires shell access unless the [frozenCache](#) plain T<sub>E</sub>X option is enabled.



**Figure 4: An example directed graph**

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

```

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

```

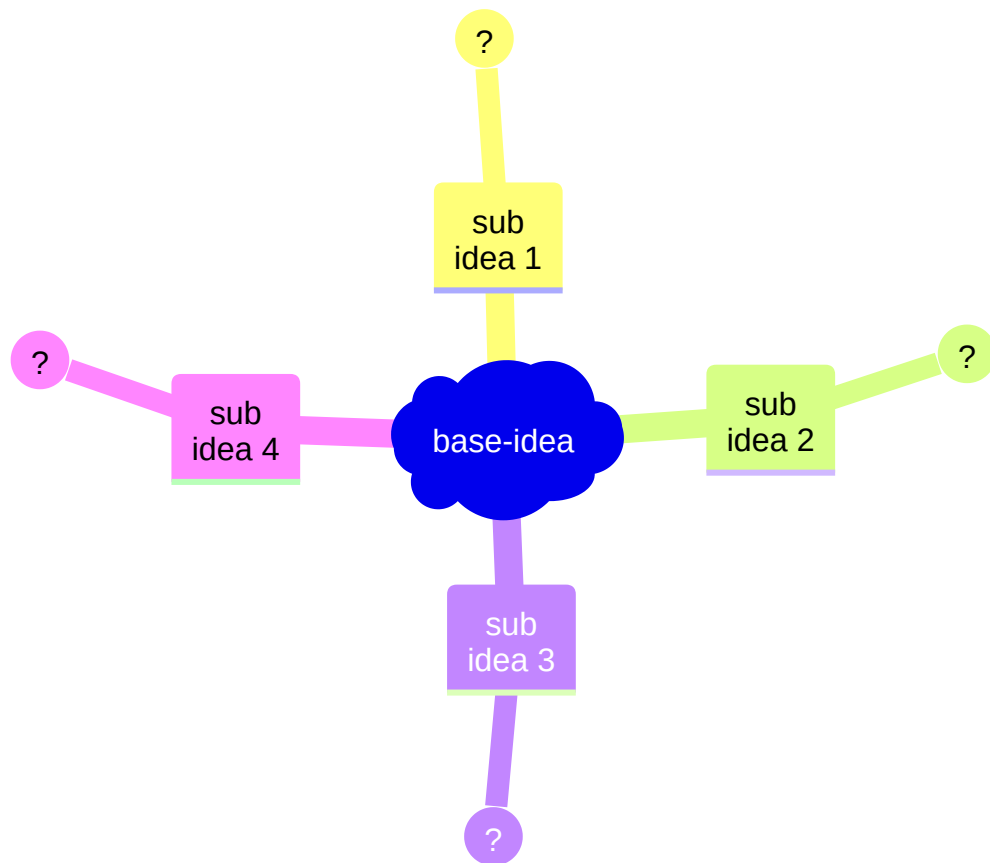
Typesetting the above document produces the output shown in Figure 7. The theme requires the catchfile  $\LaTeX$  package and a Unix-like operating system with GNU Coreutils `md5sum` and either GNU Wget or cURL installed. The theme also requires shell access unless the `frozenCache` plain  $\TeX$  option is enabled.

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

```

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

```



**Figure 5: An example mindmap**

## Simple Request-Response Flow

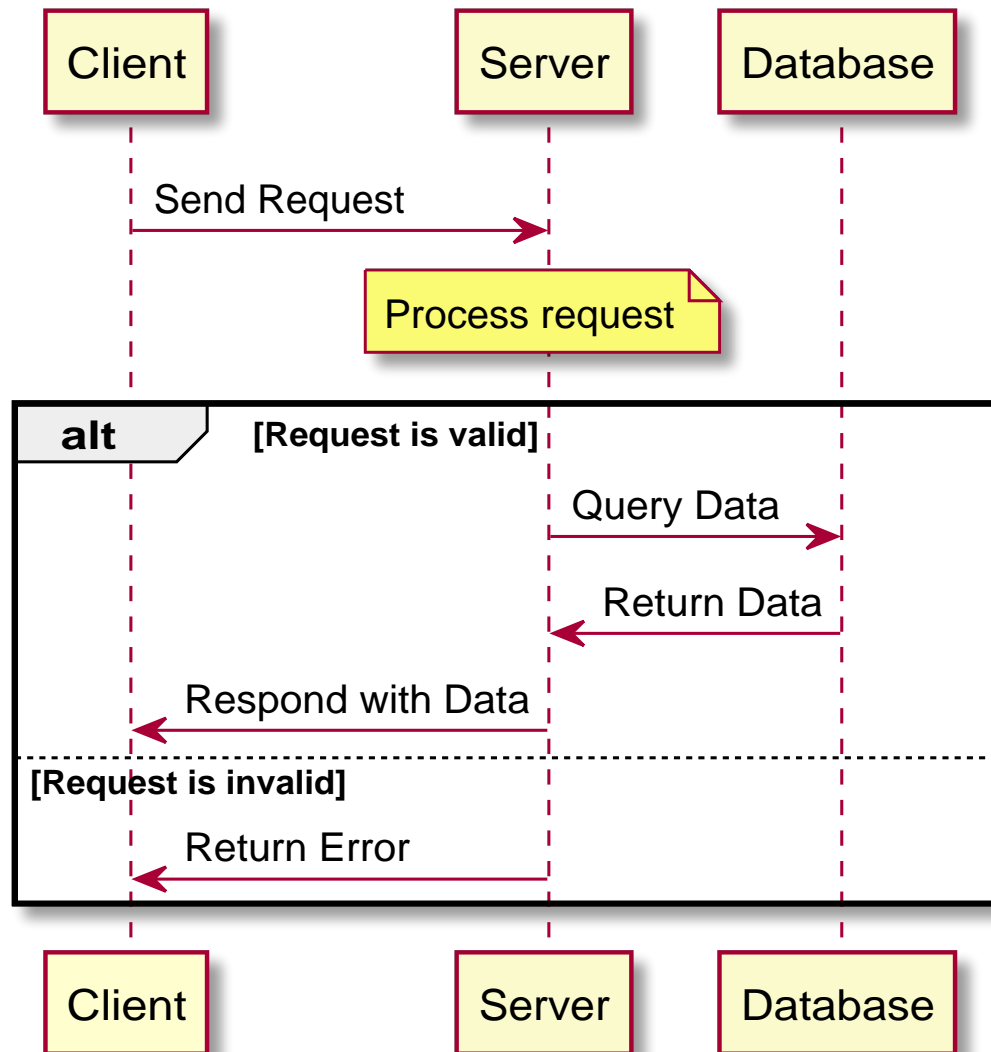


Figure 6: An example UML sequence diagram

```

\documentclass{book}
\usepackage{markdown}
\markdownSetup{pipeTables,tableCaptions}
\begin{document}
\begin{markdown}
Introduction
=====
Section
Subsection
Hello *Markdown*!

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

: Table
\end{markdown}
\end{document}

```



# Chapter 1

## Introduction

1.1 Section

1.1.1 Subsection

Hello *Markdown*!

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

Table 1.1: Table

**Figure 7: The banner of the Markdown package**

```

\markdownEnd
\bye

```

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

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

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

### 2.2.4 Snippets

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

```

1425 \prop_new:N
1426 \g_@@_snippets_prop
1427 \cs_new:Nn

```

```

1428 \@@_setup_snippet:nn
1429 {
1430 \tl_if_empty:nT
1431 { #1 }
1432 {
1433 \msg_error:nnn
1434 { markdown }
1435 { empty-snippet-name }
1436 { #1 }
1437 }
1438 \tl_set:NV
1439 \l_tmpa_tl
1440 \g_@@_current_theme_tl
1441 \tl_put_right:Nn
1442 \l_tmpa_tl
1443 { #1 }
1444 \@@_if_snippet_exists:nT
1445 { #1 }
1446 {
1447 \msg_warning:nnV
1448 { markdown }
1449 { redefined-snippet }
1450 \l_tmpa_tl
1451 }
1452 \keys_precompile:nnN
1453 { markdown/options }
1454 { #2 }
1455 \l_tmpb_tl
1456 \prop_gput:NVV
1457 \g_@@_snippets_prop
1458 \l_tmpa_tl
1459 \l_tmpb_tl
1460 }
1461 \cs_gset_eq:NN
1462 \markdownSetupSnippet
1463 \@@_setup_snippet:nn
1464 \msg_new:nnnn
1465 { markdown }
1466 { empty-snippet-name }
1467 { Empty~snippet~name~#1 }
1468 { Pick~a~non-empty~name~for~your~snippet }
1469 \msg_new:nnn
1470 { markdown }
1471 { redefined-snippet }
1472 { Redefined~snippet~#1 }

```

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



```

1473 \tl_new:N
1474 \l_@@_current_snippet_tl
1475 \prg_new_conditional:Nnn
1476 \@@_if_snippet_exists:n
1477 { TF, T, F }
1478 {
1479 \tl_set:NV
1480 \l_@@_current_snippet_tl
1481 \g_@@_current_theme_tl
1482 \tl_put_right:Nn
1483 \l_@@_current_snippet_tl
1484 { #1 }
1485 \prop_if_in:NVTF
1486 \g_@@_snippets_prop
1487 \l_@@_current_snippet_tl
1488 { \prg_return_true: }
1489 { \prg_return_false: }
1490 }
1491 \cs_gset_eq:NN
1492 \markdownIfSnippetExists
1493 \@@_if_snippet_exists:nTF

```

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

```

1494 \keys_define:nn
1495 { markdown/options }
1496 {
1497 snippet .code:n = {
1498 \tl_set:NV
1499 \l_tmpa_tl
1500 \g_@@_current_theme_tl
1501 \tl_put_right:Nn
1502 \l_tmpa_tl
1503 { #1 }
1504 \@@_if_snippet_exists:nTF
1505 { #1 }
1506 {
1507 \prop_get:NVN
1508 \g_@@_snippets_prop
1509 \l_tmpa_tl
1510 \l_tmpb_tl
1511 \tl_use:N
1512 \l_tmpb_tl
1513 }
1514 {
1515 \msg_error:nnV
1516 { markdown }
1517 { undefined-snippet }
1518 \l_tmpa_tl

```

```

1519 }
1520 }
1521 }
1522 \msg_new:nnn
1523 { markdown }
1524 { undefined-snippet }
1525 { Can't~invoke~undefined~snippet~#1 }
1526 \ExplSyntaxOff

```

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

```

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

```

The following ordered list will be preceded by arabic numerals:

1. wahid
2. aithnayn

```

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

```

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```

\end{markdown}

```

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

```

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

```

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```
\end{markdown}
```

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

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

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```
\end{markdown}
```

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

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

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```
\end{markdown}
```

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

```

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

```

```

1527 \ExplSyntaxOn
1528 \tl_new:N
1529 \l_@@_import_current_theme_tl
1530 \keys_define:nn
1531 { markdown/options/import }
1532 {

```

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

```

1533 unknown .default:n = {},
1534 unknown .code:n = {

```

To ensure that keys containing forward slashes get passed correctly, we replace all forward slashes in the input with backslash tokens with category code letter and then undo the replacement. This means that if any unbraced backslash tokens with category code letter exist in the input, they will be replaced with forward slashes. However, this should be extremely rare.

```

1535 \tl_set_eq:NN
1536 \l_@@_import_current_theme_tl
1537 \l_keys_key_str
1538 \tl_replace_all:NVn
1539 \l_@@_import_current_theme_tl
1540 \c_backslash_str
1541 { / }

```

Here, we import the snippets.

```

1542 \clist_map_inline:nn
1543 { #1 }
1544 {
1545 \regex_extract_once:nnNTF
1546 { ^(.*)\s+as\s+(.*)$ }

```

```

1547 { ##1 }
1548 \l_tmpa_seq
1549 {
1550 \seq_pop:NN
1551 \l_tmpa_seq
1552 \l_tmpa_tl
1553 \seq_pop:NN
1554 \l_tmpa_seq
1555 \l_tmpa_tl
1556 \seq_pop:NN
1557 \l_tmpa_seq
1558 \l_tmpb_tl
1559 }
1560 {
1561 \tl_set:Nn
1562 \l_tmpa_tl
1563 { ##1 }
1564 \tl_set:Nn
1565 \l_tmpb_tl
1566 { ##1 }
1567 }
1568 \tl_put_left:Nn
1569 \l_tmpa_tl
1570 { / }
1571 \tl_put_left:NV
1572 \l_tmpa_tl
1573 \l_@@_import_current_theme_tl
1574 \@@_setup_snippet:Vx
1575 \l_tmpb_tl
1576 { snippet = { \l_tmpa_tl } }
1577 }

```

Here, we load the theme.

```

1578 \@@_set_theme:V
1579 \l_@@_import_current_theme_tl
1580 },
1581 }
1582 \cs_generate_variant:Nn
1583 \tl_replace_all:Nnn
1584 { NVn }
1585 \cs_generate_variant:Nn
1586 \@@_set_theme:n
1587 { V }
1588 \cs_generate_variant:Nn
1589 \@@_setup_snippet:nn
1590 { Vx }

```

### 2.2.5 Token Renderers

The following  $\TeX$  macros may occur inside the output of the converter functions exposed by the Lua interface (see Section 2.1.1) and represent the parsed markdown tokens. These macros are intended to be redefined by the user who is typesetting a document. By default, they point to the corresponding prototypes (see Section 2.2.6).

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

```
1591 \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.

```
1592 \prop_new:N \g_@@_renderer_arities_prop
```

```
1593 \ExplSyntaxOff
```

#### 2.2.5.1 Attribute Renderers

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

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

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

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

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

```
1594 \ExplSyntaxOn
```

```
1595 \cs_gset_protected:Npn
```

```
1596 \markdownRendererAttributeIdentifier
```

```
1597 {
```

```
1598 \markdownRendererAttributeIdentifierPrototype
```

```
1599 }
```

```
1600 \seq_gput_right:Nn
```

```
1601 \g_@@_renderers_seq
```

```

1602 { attributeIdentifier }
1603 \prop_gput:Nnn
1604 \g_@@_renderer_arities_prop
1605 { attributeIdentifier }
1606 { 1 }
1607 \cs_gset_protected:Npn
1608 \markdownRendererAttributeClassName
1609 {
1610 \markdownRendererAttributeClassNamePrototype
1611 }
1612 \seq_gput_right:Nn
1613 \g_@@_renderers_seq
1614 { attributeClassName }
1615 \prop_gput:Nnn
1616 \g_@@_renderer_arities_prop
1617 { attributeClassName }
1618 { 1 }
1619 \cs_gset_protected:Npn
1620 \markdownRendererAttributeKeyValue
1621 {
1622 \markdownRendererAttributeKeyValuePrototype
1623 }
1624 \seq_gput_right:Nn
1625 \g_@@_renderers_seq
1626 { attributeKeyValue }
1627 \prop_gput:Nnn
1628 \g_@@_renderer_arities_prop
1629 { attributeKeyValue }
1630 { 2 }
1631 \ExplSyntaxOff

```

### 2.2.5.2 Block Quote Renderers

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

```

1632 \ExplSyntaxOn
1633 \cs_gset_protected:Npn
1634 \markdownRendererBlockQuoteBegin
1635 {
1636 \markdownRendererBlockQuoteBeginPrototype
1637 }
1638 \seq_gput_right:Nn
1639 \g_@@_renderers_seq
1640 { blockQuoteBegin }
1641 \prop_gput:Nnn
1642 \g_@@_renderer_arities_prop
1643 { blockQuoteBegin }

```

```

1644 { 0 }
1645 \ExplSyntaxOff

```

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

```

1646 \ExplSyntaxOn
1647 \cs_gset_protected:Npn
1648 \markdownRendererBlockQuoteEnd
1649 {
1650 \markdownRendererBlockQuoteEndPrototype
1651 }
1652 \seq_gput_right:Nn
1653 \g_@@_renderers_seq
1654 { blockQuoteEnd }
1655 \prop_gput:Nnn
1656 \g_@@_renderer_arities_prop
1657 { blockQuoteEnd }
1658 { 0 }
1659 \ExplSyntaxOff

```

### 2.2.5.3 Bracketed Spans Attribute Context Renderers

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

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

```

1660 \ExplSyntaxOn
1661 \cs_gset_protected:Npn
1662 \markdownRendererBracketedSpanAttributeContextBegin
1663 {
1664 \markdownRendererBracketedSpanAttributeContextBeginPrototype
1665 }
1666 \seq_gput_right:Nn
1667 \g_@@_renderers_seq
1668 { bracketedSpanAttributeContextBegin }
1669 \prop_gput:Nnn
1670 \g_@@_renderer_arities_prop
1671 { bracketedSpanAttributeContextBegin }
1672 { 0 }
1673 \cs_gset_protected:Npn
1674 \markdownRendererBracketedSpanAttributeContextEnd
1675 {
1676 \markdownRendererBracketedSpanAttributeContextEndPrototype
1677 }
1678 \seq_gput_right:Nn

```



```

1679 \g_@@_renderers_seq
1680 { bracketedSpanAttributeContextEnd }
1681 \prop_gput:Nnn
1682 \g_@@_renderer_arities_prop
1683 { bracketedSpanAttributeContextEnd }
1684 { 0 }
1685 \ExplSyntaxOff

```

#### 2.2.5.4 Bullet List Renderers

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

```

1686 \ExplSyntaxOn
1687 \cs_gset_protected:Npn
1688 \markdownRendererUlBegin
1689 {
1690 \markdownRendererUlBeginPrototype
1691 }
1692 \seq_gput_right:Nn
1693 \g_@@_renderers_seq
1694 { ulBegin }
1695 \prop_gput:Nnn
1696 \g_@@_renderer_arities_prop
1697 { ulBegin }
1698 { 0 }
1699 \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.

```

1700 \ExplSyntaxOn
1701 \cs_gset_protected:Npn
1702 \markdownRendererUlBeginTight
1703 {
1704 \markdownRendererUlBeginTightPrototype
1705 }
1706 \seq_gput_right:Nn
1707 \g_@@_renderers_seq
1708 { ulBeginTight }
1709 \prop_gput:Nnn
1710 \g_@@_renderer_arities_prop
1711 { ulBeginTight }
1712 { 0 }
1713 \ExplSyntaxOff

```

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

```

1714 \ExplSyntaxOn
1715 \cs_gset_protected:Npn
1716 \markdownRendererUListItem
1717 {
1718 \markdownRendererUListItemPrototype
1719 }
1720 \seq_gput_right:Nn
1721 \g_@@_renderers_seq
1722 { ulItem }
1723 \prop_gput:Nnn
1724 \g_@@_renderer_arities_prop
1725 { ulItem }
1726 { 0 }
1727 \ExplSyntaxOff

```

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

```

1728 \ExplSyntaxOn
1729 \cs_gset_protected:Npn
1730 \markdownRendererUItemEnd
1731 {
1732 \markdownRendererUItemEndPrototype
1733 }
1734 \seq_gput_right:Nn
1735 \g_@@_renderers_seq
1736 { ulItemEnd }
1737 \prop_gput:Nnn
1738 \g_@@_renderer_arities_prop
1739 { ulItemEnd }
1740 { 0 }
1741 \ExplSyntaxOff

```

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

```

1742 \ExplSyntaxOn
1743 \cs_gset_protected:Npn
1744 \markdownRendererUEnd
1745 {
1746 \markdownRendererUEndPrototype
1747 }
1748 \seq_gput_right:Nn
1749 \g_@@_renderers_seq
1750 { ulEnd }

```

```

1751 \prop_gput:Nnn
1752 \g_@@_renderer_arities_prop
1753 { ulEnd }
1754 { 0 }
1755 \ExplSyntaxOff

```

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

```

1756 \ExplSyntaxOn
1757 \cs_gset_protected:Npn
1758 \markdownRendererUEndTight
1759 {
1760 \markdownRendererUEndTightPrototype
1761 }
1762 \seq_gput_right:Nn
1763 \g_@@_renderers_seq
1764 { ulEndTight }
1765 \prop_gput:Nnn
1766 \g_@@_renderer_arities_prop
1767 { ulEndTight }
1768 { 0 }
1769 \ExplSyntaxOff

```

#### 2.2.5.5 Citation Renderers

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

```

1770 \ExplSyntaxOn
1771 \cs_gset_protected:Npn
1772 \markdownRendererCite
1773 {
1774 \markdownRendererCitePrototype
1775 }
1776 \seq_gput_right:Nn
1777 \g_@@_renderers_seq
1778 { cite }
1779 \prop_gput:Nnn
1780 \g_@@_renderer_arities_prop
1781 { cite }
1782 { 1 }

```

```
1783 \ExplSyntaxOff
```

The `\markdownRendererTextCite` macro represents a string of one or more text citations. This macro will only be produced, when the `citations` option is enabled. The macro receives parameters in the same format as the `\markdownRendererCite` macro.

```
1784 \ExplSyntaxOn
1785 \cs_gset_protected:Npn
1786 \markdownRendererTextCite
1787 {
1788 \markdownRendererTextCitePrototype
1789 }
1790 \seq_gput_right:Nn
1791 \g_@@_renderers_seq
1792 { textCite }
1793 \prop_gput:Nnn
1794 \g_@@_renderer_arities_prop
1795 { textCite }
1796 { 1 }
1797 \ExplSyntaxOff
```

#### 2.2.5.6 Code Block Renderers

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

```
1798 \ExplSyntaxOn
1799 \cs_gset_protected:Npn
1800 \markdownRendererInputVerbatim
1801 {
1802 \markdownRendererInputVerbatimPrototype
1803 }
1804 \seq_gput_right:Nn
1805 \g_@@_renderers_seq
1806 { inputVerbatim }
1807 \prop_gput:Nnn
1808 \g_@@_renderer_arities_prop
1809 { inputVerbatim }
1810 { 1 }
1811 \ExplSyntaxOff
```

The `\markdownRendererInputFencedCode` macro represents a fenced code block. This macro will only be produced, when the `fencedCode` option is enabled. The macro receives three arguments that correspond to the filename of a file containing the code block contents, the fully escaped code fence infostring that can be directly typeset, and the raw code fence infostring that can be used outside typesetting.

```

1812 \ExplSyntaxOn
1813 \cs_gset_protected:Npn
1814 \markdownRendererInputFencedCode
1815 {
1816 \markdownRendererInputFencedCodePrototype
1817 }
1818 \seq_gput_right:Nn
1819 \g_@@_renderers_seq
1820 { inputFencedCode }
1821 \prop_gput:Nnn
1822 \g_@@_renderer_arities_prop
1823 { inputFencedCode }
1824 { 3 }
1825 \ExplSyntaxOff

```

### 2.2.5.7 Code Span Renderer

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

```

1826 \ExplSyntaxOn
1827 \cs_gset_protected:Npn
1828 \markdownRendererCodeSpan
1829 {
1830 \markdownRendererCodeSpanPrototype
1831 }
1832 \seq_gput_right:Nn
1833 \g_@@_renderers_seq
1834 { codeSpan }
1835 \prop_gput:Nnn
1836 \g_@@_renderer_arities_prop
1837 { codeSpan }
1838 { 1 }
1839 \ExplSyntaxOff

```

### 2.2.5.8 Code Span Attribute Context Renderers

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

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

```

1840 \ExplSyntaxOn
1841 \cs_gset_protected:Npn
1842 \markdownRendererCodeSpanAttributeContextBegin
1843 {
1844 \markdownRendererCodeSpanAttributeContextBeginPrototype

```

```

1845 }
1846 \seq_gput_right:Nn
1847 \g_@@_renderers_seq
1848 { codeSpanAttributeContextBegin }
1849 \prop_gput:Nnn
1850 \g_@@_renderer_arities_prop
1851 { codeSpanAttributeContextBegin }
1852 { 0 }
1853 \cs_gset_protected:Npn
1854 \markdownRendererCodeSpanAttributeContextEnd
1855 {
1856 \markdownRendererCodeSpanAttributeContextEndPrototype
1857 }
1858 \seq_gput_right:Nn
1859 \g_@@_renderers_seq
1860 { codeSpanAttributeContextEnd }
1861 \prop_gput:Nnn
1862 \g_@@_renderer_arities_prop
1863 { codeSpanAttributeContextEnd }
1864 { 0 }
1865 \ExplSyntaxOff

```

### 2.2.5.9 Content Block Renderers

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

```

1866 \ExplSyntaxOn
1867 \cs_gset_protected:Npn
1868 \markdownRendererContentBlock
1869 {
1870 \markdownRendererContentBlockPrototype
1871 }
1872 \seq_gput_right:Nn
1873 \g_@@_renderers_seq
1874 { contentBlock }
1875 \prop_gput:Nnn
1876 \g_@@_renderer_arities_prop
1877 { contentBlock }
1878 { 4 }
1879 \ExplSyntaxOff

```

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

```

1880 \ExplSyntaxOn
1881 \cs_gset_protected:Npn
1882 \markdownRendererContentBlockOnlineImage
1883 {
1884 \markdownRendererContentBlockOnlineImagePrototype
1885 }
1886 \seq_gput_right:Nn
1887 \g_@@_renderers_seq
1888 { contentBlockOnlineImage }
1889 \prop_gput:Nnn
1890 \g_@@_renderer_arities_prop
1891 { contentBlockOnlineImage }
1892 { 4 }
1893 \ExplSyntaxOff

```

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

Note that you will need to place a `markdown-languages.json` file inside your working directory or inside your local T<sub>E</sub>X directory structure. In this file, you will define a mapping between filename extensions and the language names recognized by your favorite syntax highlighter; there may exist other creative uses beside syntax highlighting. The `Languages.json` file provided by Sotkov [5] is a good starting point.

```

1894 \ExplSyntaxOn
1895 \cs_gset_protected:Npn
1896 \markdownRendererContentBlockCode
1897 {
1898 \markdownRendererContentBlockCodePrototype
1899 }
1900 \seq_gput_right:Nn
1901 \g_@@_renderers_seq
1902 { contentBlockCode }
1903 \prop_gput:Nnn
1904 \g_@@_renderer_arities_prop
1905 { contentBlockCode }
1906 { 5 }
1907 \ExplSyntaxOff

```

---

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

### 2.2.5.10 Definition List Renderers

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

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

```
1908 \ExplSyntaxOn
1909 \cs_gset_protected:Npn
1910 \markdownRendererDlBegin
1911 {
1912 \markdownRendererDlBeginPrototype
1913 }
1914 \seq_gput_right:Nn
1915 \g_@@_renderers_seq
1916 { dlBegin }
1917 \prop_gput:Nnn
1918 \g_@@_renderer_arities_prop
1919 { dlBegin }
1920 { 0 }
1921 \ExplSyntaxOff
```

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

```
1922 \ExplSyntaxOn
1923 \cs_gset_protected:Npn
1924 \markdownRendererDlBeginTight
1925 {
1926 \markdownRendererDlBeginTightPrototype
1927 }
1928 \seq_gput_right:Nn
1929 \g_@@_renderers_seq
1930 { dlBeginTight }
1931 \prop_gput:Nnn
1932 \g_@@_renderer_arities_prop
1933 { dlBeginTight }
1934 { 0 }
1935 \ExplSyntaxOff
```

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

```
1936 \ExplSyntaxOn
1937 \cs_gset_protected:Npn
1938 \markdownRendererDlItem
```



```

1939 {
1940 \markdownRendererDlItemPrototype
1941 }
1942 \seq_gput_right:Nn
1943 \g_@@_renderers_seq
1944 { dlItem }
1945 \prop_gput:Nnn
1946 \g_@@_renderer_arities_prop
1947 { dlItem }
1948 { 1 }
1949 \ExplSyntaxOff

```

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

```

1950 \ExplSyntaxOn
1951 \cs_gset_protected:Npn
1952 \markdownRendererDlItemEnd
1953 {
1954 \markdownRendererDlItemEndPrototype
1955 }
1956 \seq_gput_right:Nn
1957 \g_@@_renderers_seq
1958 { dlItemEnd }
1959 \prop_gput:Nnn
1960 \g_@@_renderer_arities_prop
1961 { dlItemEnd }
1962 { 0 }
1963 \ExplSyntaxOff

```

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

```

1964 \ExplSyntaxOn
1965 \cs_gset_protected:Npn
1966 \markdownRendererDlDefinitionBegin
1967 {
1968 \markdownRendererDlDefinitionBeginPrototype
1969 }
1970 \seq_gput_right:Nn
1971 \g_@@_renderers_seq
1972 { dlDefinitionBegin }
1973 \prop_gput:Nnn
1974 \g_@@_renderer_arities_prop
1975 { dlDefinitionBegin }
1976 { 0 }
1977 \ExplSyntaxOff

```

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

```

1978 \ExplSyntaxOn
1979 \cs_gset_protected:Npn
1980 \markdownRendererDlDefinitionEnd
1981 {
1982 \markdownRendererDlDefinitionEndPrototype
1983 }
1984 \seq_gput_right:Nn
1985 \g_@@_renderers_seq
1986 { dlDefinitionEnd }
1987 \prop_gput:Nnn
1988 \g_@@_renderer_arities_prop
1989 { dlDefinitionEnd }
1990 { 0 }
1991 \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.

```

1992 \ExplSyntaxOn
1993 \cs_gset_protected:Npn
1994 \markdownRendererDlEnd
1995 {
1996 \markdownRendererDlEndPrototype
1997 }
1998 \seq_gput_right:Nn
1999 \g_@@_renderers_seq
2000 { dlEnd }
2001 \prop_gput:Nnn
2002 \g_@@_renderer_arities_prop
2003 { dlEnd }
2004 { 0 }
2005 \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.

```

2006 \ExplSyntaxOn
2007 \cs_gset_protected:Npn
2008 \markdownRendererDlEndTight
2009 {
2010 \markdownRendererDlEndTightPrototype
2011 }
2012 \seq_gput_right:Nn

```

```

2013 \g_@@_renderers_seq
2014 { dlEndTight }
2015 \prop_gput:Nnn
2016 \g_@@_renderer_arities_prop
2017 { dlEndTight }
2018 { 0 }
2019 \ExplSyntaxOff

```

### 2.2.5.11 Ellipsis Renderer

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

```

2020 \ExplSyntaxOn
2021 \cs_gset_protected:Npn
2022 \markdownRendererEllipsis
2023 {
2024 \markdownRendererEllipsisPrototype
2025 }
2026 \seq_gput_right:Nn
2027 \g_@@_renderers_seq
2028 { ellipsis }
2029 \prop_gput:Nnn
2030 \g_@@_renderer_arities_prop
2031 { ellipsis }
2032 { 0 }
2033 \ExplSyntaxOff

```

### 2.2.5.12 Emphasis Renderers

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

```

2034 \ExplSyntaxOn
2035 \cs_gset_protected:Npn
2036 \markdownRendererEmphasis
2037 {
2038 \markdownRendererEmphasisPrototype
2039 }
2040 \seq_gput_right:Nn
2041 \g_@@_renderers_seq
2042 { emphasis }
2043 \prop_gput:Nnn
2044 \g_@@_renderer_arities_prop
2045 { emphasis }
2046 { 1 }
2047 \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.

```

2048 \ExplSyntaxOn
2049 \cs_gset_protected:Npn
2050 \markdownRendererStrongEmphasis
2051 {
2052 \markdownRendererStrongEmphasisPrototype
2053 }
2054 \seq_gput_right:Nn
2055 \g_@@_renderers_seq
2056 { strongEmphasis }
2057 \prop_gput:Nnn
2058 \g_@@_renderer_arities_prop
2059 { strongEmphasis }
2060 { 1 }
2061 \ExplSyntaxOff

```

### 2.2.5.13 Fenced Code Attribute Context Renderers

The following macros are only produced, when the `fencedCode` and `fencedCodeAttributes` options are enabled.

The `\markdownRendererFencedCodeAttributeContextBegin` and `\markdownRendererFencedCodeAttributeContextEnd` macros represent the beginning and the end of a context in which the attributes of a fenced code apply. The macros receive no arguments.

```

2062 \ExplSyntaxOn
2063 \cs_gset_protected:Npn
2064 \markdownRendererFencedCodeAttributeContextBegin
2065 {
2066 \markdownRendererFencedCodeAttributeContextBeginPrototype
2067 }
2068 \seq_gput_right:Nn
2069 \g_@@_renderers_seq
2070 { fencedCodeAttributeContextBegin }
2071 \prop_gput:Nnn
2072 \g_@@_renderer_arities_prop
2073 { fencedCodeAttributeContextBegin }
2074 { 0 }
2075 \cs_gset_protected:Npn
2076 \markdownRendererFencedCodeAttributeContextEnd
2077 {
2078 \markdownRendererFencedCodeAttributeContextEndPrototype
2079 }
2080 \seq_gput_right:Nn
2081 \g_@@_renderers_seq
2082 { fencedCodeAttributeContextEnd }

```

```

2083 \prop_gput:Nnn
2084 \g_@@_renderer_arities_prop
2085 { fencedCodeAttributeContextEnd }
2086 { 0 }
2087 \ExplSyntaxOff

```

#### 2.2.5.14 Fenced Div Attribute Context Renderers

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

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

```

2088 \ExplSyntaxOn
2089 \cs_gset_protected:Npn
2090 \markdownRendererFencedDivAttributeContextBegin
2091 {
2092 \markdownRendererFencedDivAttributeContextBeginPrototype
2093 }
2094 \seq_gput_right:Nn
2095 \g_@@_renderers_seq
2096 { fencedDivAttributeContextBegin }
2097 \prop_gput:Nnn
2098 \g_@@_renderer_arities_prop
2099 { fencedDivAttributeContextBegin }
2100 { 0 }
2101 \cs_gset_protected:Npn
2102 \markdownRendererFencedDivAttributeContextEnd
2103 {
2104 \markdownRendererFencedDivAttributeContextEndPrototype
2105 }
2106 \seq_gput_right:Nn
2107 \g_@@_renderers_seq
2108 { fencedDivAttributeContextEnd }
2109 \prop_gput:Nnn
2110 \g_@@_renderer_arities_prop
2111 { fencedDivAttributeContextEnd }
2112 { 0 }
2113 \ExplSyntaxOff

```

#### 2.2.5.15 Header Attribute Context Renderers

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

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

```

2114 \ExplSyntaxOn
2115 \cs_gset_protected:Npn
2116 \markdownRendererHeaderAttributeContextBegin
2117 {
2118 \markdownRendererHeaderAttributeContextBeginPrototype
2119 }
2120 \seq_gput_right:Nn
2121 \g_@@_renderers_seq
2122 { headerAttributeContextBegin }
2123 \prop_gput:Nnn
2124 \g_@@_renderer_arities_prop
2125 { headerAttributeContextBegin }
2126 { 0 }
2127 \cs_gset_protected:Npn
2128 \markdownRendererHeaderAttributeContextEnd
2129 {
2130 \markdownRendererHeaderAttributeContextEndPrototype
2131 }
2132 \seq_gput_right:Nn
2133 \g_@@_renderers_seq
2134 { headerAttributeContextEnd }
2135 \prop_gput:Nnn
2136 \g_@@_renderer_arities_prop
2137 { headerAttributeContextEnd }
2138 { 0 }
2139 \ExplSyntaxOff

```

### 2.2.5.16 Heading Renderers

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

```

2140 \ExplSyntaxOn
2141 \cs_gset_protected:Npn
2142 \markdownRendererHeadingOne
2143 {
2144 \markdownRendererHeadingOnePrototype
2145 }
2146 \seq_gput_right:Nn
2147 \g_@@_renderers_seq
2148 { headingOne }
2149 \prop_gput:Nnn
2150 \g_@@_renderer_arities_prop
2151 { headingOne }
2152 { 1 }
2153 \ExplSyntaxOff

```

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

```

2154 \ExplSyntaxOn
2155 \cs_gset_protected:Npn
2156 \markdownRendererHeadingTwo
2157 {
2158 \markdownRendererHeadingTwoPrototype
2159 }
2160 \seq_gput_right:Nn
2161 \g_@@_renderers_seq
2162 { headingTwo }
2163 \prop_gput:Nnn
2164 \g_@@_renderer_arities_prop
2165 { headingTwo }
2166 { 1 }
2167 \ExplSyntaxOff

```

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

```

2168 \ExplSyntaxOn
2169 \cs_gset_protected:Npn
2170 \markdownRendererHeadingThree
2171 {
2172 \markdownRendererHeadingThreePrototype
2173 }
2174 \seq_gput_right:Nn
2175 \g_@@_renderers_seq
2176 { headingThree }
2177 \prop_gput:Nnn
2178 \g_@@_renderer_arities_prop
2179 { headingThree }
2180 { 1 }
2181 \ExplSyntaxOff

```

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

```

2182 \ExplSyntaxOn
2183 \cs_gset_protected:Npn
2184 \markdownRendererHeadingFour
2185 {
2186 \markdownRendererHeadingFourPrototype
2187 }
2188 \seq_gput_right:Nn
2189 \g_@@_renderers_seq
2190 { headingFour }
2191 \prop_gput:Nnn

```

```

2192 \g_@@_renderer_arities_prop
2193 { headingFour }
2194 { 1 }
2195 \ExplSyntaxOff

```

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

```

2196 \ExplSyntaxOn
2197 \cs_gset_protected:Npn
2198 \markdownRendererHeadingFive
2199 {
2200 \markdownRendererHeadingFivePrototype
2201 }
2202 \seq_gput_right:Nn
2203 \g_@@_renderers_seq
2204 { headingFive }
2205 \prop_gput:Nnn
2206 \g_@@_renderer_arities_prop
2207 { headingFive }
2208 { 1 }
2209 \ExplSyntaxOff

```

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

```

2210 \ExplSyntaxOn
2211 \cs_gset_protected:Npn
2212 \markdownRendererHeadingSix
2213 {
2214 \markdownRendererHeadingSixPrototype
2215 }
2216 \seq_gput_right:Nn
2217 \g_@@_renderers_seq
2218 { headingSix }
2219 \prop_gput:Nnn
2220 \g_@@_renderer_arities_prop
2221 { headingSix }
2222 { 1 }
2223 \ExplSyntaxOff

```

#### 2.2.5.17 Inline HTML Comment Renderer

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

```

2224 \ExplSyntaxOn

```



```

2225 \cs_gset_protected:Npn
2226 \markdownRendererInlineHtmlComment
2227 {
2228 \markdownRendererInlineHtmlCommentPrototype
2229 }
2230 \seq_gput_right:Nn
2231 \g_@@_renderers_seq
2232 { inlineHtmlComment }
2233 \prop_gput:Nnn
2234 \g_@@_renderer_arities_prop
2235 { inlineHtmlComment }
2236 { 1 }
2237 \ExplSyntaxOff

```

### 2.2.5.18 HTML Tag and Element Renderers

The `\markdownRendererInlineHtmlTag` macro represents an opening, closing, or empty inline HTML tag. This macro will only be produced, when the `html` option is enabled. The macro receives a single argument that corresponds to the contents of the HTML tag.

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

```

2238 \ExplSyntaxOn
2239 \cs_gset_protected:Npn
2240 \markdownRendererInlineHtmlTag
2241 {
2242 \markdownRendererInlineHtmlTagPrototype
2243 }
2244 \seq_gput_right:Nn
2245 \g_@@_renderers_seq
2246 { inlineHtmlTag }
2247 \prop_gput:Nnn
2248 \g_@@_renderer_arities_prop
2249 { inlineHtmlTag }
2250 { 1 }
2251 \cs_gset_protected:Npn
2252 \markdownRendererInputBlockHtmlElement
2253 {
2254 \markdownRendererInputBlockHtmlElementPrototype
2255 }
2256 \seq_gput_right:Nn
2257 \g_@@_renderers_seq
2258 { inputBlockHtmlElement }
2259 \prop_gput:Nnn

```

```

2260 \g_@@_renderer_arities_prop
2261 { inputBlockHtmlElement }
2262 { 1 }
2263 \ExplSyntaxOff

```

### 2.2.5.19 Image Renderer

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

```

2264 \ExplSyntaxOn
2265 \cs_gset_protected:Npn
2266 \markdownRendererImage
2267 {
2268 \markdownRendererImagePrototype
2269 }
2270 \seq_gput_right:Nn
2271 \g_@@_renderers_seq
2272 { image }
2273 \prop_gput:Nnn
2274 \g_@@_renderer_arities_prop
2275 { image }
2276 { 4 }
2277 \ExplSyntaxOff

```

### 2.2.5.20 Image Attribute Context Renderers

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

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

```

2278 \ExplSyntaxOn
2279 \cs_gset_protected:Npn
2280 \markdownRendererImageAttributeContextBegin
2281 {
2282 \markdownRendererImageAttributeContextBeginPrototype
2283 }
2284 \seq_gput_right:Nn
2285 \g_@@_renderers_seq
2286 { imageAttributeContextBegin }
2287 \prop_gput:Nnn
2288 \g_@@_renderer_arities_prop
2289 { imageAttributeContextBegin }
2290 { 0 }
2291 \cs_gset_protected:Npn

```

```

2292 \markdownRendererImageAttributeContextEnd
2293 {
2294 \markdownRendererImageAttributeContextEndPrototype
2295 }
2296 \seq_gput_right:Nn
2297 \g_@@_renderers_seq
2298 { imageAttributeContextEnd }
2299 \prop_gput:Nnn
2300 \g_@@_renderer_arities_prop
2301 { imageAttributeContextEnd }
2302 { 0 }
2303 \ExplSyntaxOff

```

### 2.2.5.21 Interblock Separator Renderers

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

```

2304 \ExplSyntaxOn
2305 \cs_gset_protected:Npn
2306 \markdownRendererInterblockSeparator
2307 {
2308 \markdownRendererInterblockSeparatorPrototype
2309 }
2310 \seq_gput_right:Nn
2311 \g_@@_renderers_seq
2312 { interblockSeparator }
2313 \prop_gput:Nnn
2314 \g_@@_renderer_arities_prop
2315 { interblockSeparator }
2316 { 0 }
2317 \ExplSyntaxOff

```

Users can use more than one blank line to delimit two block to indicate the end of a series of blocks that make up a logical paragraph. This produces a paragraph separator instead of an interblock separator. Between some blocks, such as markdown paragraphs, a paragraph separator is always produced.

The `\markdownRendererParagraphSeparator` macro represents a paragraph separator. The macro receives no arguments.

```

2318 \ExplSyntaxOn
2319 \cs_gset_protected:Npn
2320 \markdownRendererParagraphSeparator
2321 {
2322 \markdownRendererParagraphSeparatorPrototype
2323 }
2324 \seq_gput_right:Nn
2325 \g_@@_renderers_seq

```

```

2326 { paragraphSeparator }
2327 \prop_gput:Nnn
2328 \g_@@_renderer_arities_prop
2329 { paragraphSeparator }
2330 { 0 }
2331 \ExplSyntaxOff

```

### 2.2.5.22 Line Block Renderers

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

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

```

2332 \ExplSyntaxOn
2333 \cs_gset_protected:Npn
2334 \markdownRendererLineBlockBegin
2335 {
2336 \markdownRendererLineBlockBeginPrototype
2337 }
2338 \seq_gput_right:Nn
2339 \g_@@_renderers_seq
2340 { lineBlockBegin }
2341 \prop_gput:Nnn
2342 \g_@@_renderer_arities_prop
2343 { lineBlockBegin }
2344 { 0 }
2345 \cs_gset_protected:Npn
2346 \markdownRendererLineBlockEnd
2347 {
2348 \markdownRendererLineBlockEndPrototype
2349 }
2350 \seq_gput_right:Nn
2351 \g_@@_renderers_seq
2352 { lineBlockEnd }
2353 \prop_gput:Nnn
2354 \g_@@_renderer_arities_prop
2355 { lineBlockEnd }
2356 { 0 }
2357 \ExplSyntaxOff

```

### 2.2.5.23 Line Break Renderers

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

```

2358 \ExplSyntaxOn
2359 \cs_gset_protected:Npn
2360 \markdownRendererSoftLineBreak

```

```

2361 {
2362 \markdownRendererSoftLineBreakPrototype
2363 }
2364 \seq_gput_right:Nn
2365 \g_@@_renderers_seq
2366 { softLineBreak }
2367 \prop_gput:Nnn
2368 \g_@@_renderer_arities_prop
2369 { softLineBreak }
2370 { 0 }
2371 \ExplSyntaxOff

```

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

```

2372 \ExplSyntaxOn
2373 \cs_gset_protected:Npn
2374 \markdownRendererHardLineBreak
2375 {
2376 \markdownRendererHardLineBreakPrototype
2377 }
2378 \seq_gput_right:Nn
2379 \g_@@_renderers_seq
2380 { hardLineBreak }
2381 \prop_gput:Nnn
2382 \g_@@_renderer_arities_prop
2383 { hardLineBreak }
2384 { 0 }
2385 \ExplSyntaxOff

```

#### 2.2.5.24 Link Renderer

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

```

2386 \ExplSyntaxOn
2387 \cs_gset_protected:Npn
2388 \markdownRendererLink
2389 {
2390 \markdownRendererLinkPrototype
2391 }
2392 \seq_gput_right:Nn
2393 \g_@@_renderers_seq
2394 { link }
2395 \prop_gput:Nnn
2396 \g_@@_renderer_arities_prop
2397 { link }
2398 { 4 }

```

2399 \ExplSyntaxOff

### 2.2.5.25 Link Attribute Context Renderers

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

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

```
2400 \ExplSyntaxOn
2401 \cs_gset_protected:Npn
2402 \markdownRendererLinkAttributeContextBegin
2403 {
2404 \markdownRendererLinkAttributeContextBeginPrototype
2405 }
2406 \seq_gput_right:Nn
2407 \g_@@_renderers_seq
2408 { linkAttributeContextBegin }
2409 \prop_gput:Nnn
2410 \g_@@_renderer_arities_prop
2411 { linkAttributeContextBegin }
2412 { 0 }
2413 \cs_gset_protected:Npn
2414 \markdownRendererLinkAttributeContextEnd
2415 {
2416 \markdownRendererLinkAttributeContextEndPrototype
2417 }
2418 \seq_gput_right:Nn
2419 \g_@@_renderers_seq
2420 { linkAttributeContextEnd }
2421 \prop_gput:Nnn
2422 \g_@@_renderer_arities_prop
2423 { linkAttributeContextEnd }
2424 { 0 }
2425 \ExplSyntaxOff
```

### 2.2.5.26 Marked Text Renderer

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

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

```
2426 \ExplSyntaxOn
2427 \cs_gset_protected:Npn
2428 \markdownRendererMark
2429 {
2430 \markdownRendererMarkPrototype
2431 }
2432 \ExplSyntaxOff
```

```

2431 }
2432 \seq_gput_right:Nn
2433 \g_@@_renderers_seq
2434 { mark }
2435 \prop_gput:Nnn
2436 \g_@@_renderer_arities_prop
2437 { mark }
2438 { 1 }
2439 \ExplSyntaxOff

```

### 2.2.5.27 Markdown Document Renderers

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

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

```

2440 \ExplSyntaxOn
2441 \cs_gset_protected:Npn
2442 \markdownRendererDocumentBegin
2443 {
2444 \markdownRendererDocumentBeginPrototype
2445 }
2446 \seq_gput_right:Nn
2447 \g_@@_renderers_seq
2448 { documentBegin }
2449 \prop_gput:Nnn
2450 \g_@@_renderer_arities_prop
2451 { documentBegin }
2452 { 0 }
2453 \cs_gset_protected:Npn
2454 \markdownRendererDocumentEnd
2455 {
2456 \markdownRendererDocumentEndPrototype
2457 }
2458 \seq_gput_right:Nn
2459 \g_@@_renderers_seq
2460 { documentEnd }
2461 \prop_gput:Nnn
2462 \g_@@_renderer_arities_prop
2463 { documentEnd }
2464 { 0 }
2465 \ExplSyntaxOff

```

### 2.2.5.28 Non-Breaking Space Renderer

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

```
2466 \ExplSyntaxOn
2467 \cs_gset_protected:Npn
2468 \markdownRendererNbsp
2469 {
2470 \markdownRendererNbspPrototype
2471 }
2472 \seq_gput_right:Nn
2473 \g_@@_renderers_seq
2474 { nbsp }
2475 \prop_gput:Nnn
2476 \g_@@_renderer_arities_prop
2477 { nbsp }
2478 { 0 }
2479 \ExplSyntaxOff
```

### 2.2.5.29 Note Renderer

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

```
2480 \def\markdownRendererNote{%
2481 \markdownRendererNotePrototype}%
2482 \ExplSyntaxOn
2483 \seq_gput_right:Nn
2484 \g_@@_renderers_seq
2485 { note }
2486 \prop_gput:Nnn
2487 \g_@@_renderer_arities_prop
2488 { note }
2489 { 1 }
2490 \ExplSyntaxOff
```

### 2.2.5.30 Ordered List Renderers

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

```
2491 \ExplSyntaxOn
2492 \cs_gset_protected:Npn
2493 \markdownRendererOlBegin
2494 {
2495 \markdownRendererOlBeginPrototype
2496 }
```



```

2497 \seq_gput_right:Nn
2498 \g_@@_renderers_seq
2499 { olBegin }
2500 \prop_gput:Nnn
2501 \g_@@_renderar_aritys_prop
2502 { olBegin }
2503 { 0 }
2504 \ExplSyntaxOff

```

The `\markdownRenderarOlBeginTight` 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.

```

2505 \ExplSyntaxOn
2506 \cs_gset_protected:Npn
2507 \markdownRenderarOlBeginTight
2508 {
2509 \markdownRenderarOlBeginTightPrototype
2510 }
2511 \seq_gput_right:Nn
2512 \g_@@_renderers_seq
2513 { olBeginTight }
2514 \prop_gput:Nnn
2515 \g_@@_renderar_aritys_prop
2516 { olBeginTight }
2517 { 0 }
2518 \ExplSyntaxOff

```

The `\markdownRenderarFancyOlBegin` 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`).

```

2519 \ExplSyntaxOn
2520 \cs_gset_protected:Npn
2521 \markdownRenderarFancyOlBegin
2522 {
2523 \markdownRenderarFancyOlBeginPrototype
2524 }
2525 \seq_gput_right:Nn
2526 \g_@@_renderers_seq
2527 { fancyOlBegin }
2528 \prop_gput:Nnn
2529 \g_@@_renderar_aritys_prop
2530 { fancyOlBegin }

```

```

2531 { 2 }
2532 \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.

```

2533 \ExplSyntaxOn
2534 \cs_gset_protected:Npn
2535 \markdownRendererFancyOlBeginTight
2536 {
2537 \markdownRendererFancyOlBeginTightPrototype
2538 }
2539 \seq_gput_right:Nn
2540 \g_@@_renderers_seq
2541 { fancyOlBeginTight }
2542 \prop_gput:Nnn
2543 \g_@@_renderer_arities_prop
2544 { fancyOlBeginTight }
2545 { 2 }
2546 \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.

```

2547 \ExplSyntaxOn
2548 \cs_gset_protected:Npn
2549 \markdownRendererOlItem
2550 {
2551 \markdownRendererOlItemPrototype
2552 }
2553 \seq_gput_right:Nn
2554 \g_@@_renderers_seq
2555 { olItem }
2556 \prop_gput:Nnn
2557 \g_@@_renderer_arities_prop
2558 { olItem }
2559 { 0 }
2560 \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.

```

2561 \ExplSyntaxOn

```

```

2562 \cs_gset_protected:Npn
2563 \markdownRendererOlItemEnd
2564 {
2565 \markdownRendererOlItemEndPrototype
2566 }
2567 \seq_gput_right:Nn
2568 \g_@@_renderers_seq
2569 { olItemEnd }
2570 \prop_gput:Nnn
2571 \g_@@_renderer_arities_prop
2572 { olItemEnd }
2573 { 0 }
2574 \ExplSyntaxOff

```

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

```

2575 \ExplSyntaxOn
2576 \cs_gset_protected:Npn
2577 \markdownRendererOlItemWithNumber
2578 {
2579 \markdownRendererOlItemWithNumberPrototype
2580 }
2581 \seq_gput_right:Nn
2582 \g_@@_renderers_seq
2583 { olItemWithNumber }
2584 \prop_gput:Nnn
2585 \g_@@_renderer_arities_prop
2586 { olItemWithNumber }
2587 { 1 }
2588 \ExplSyntaxOff

```

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

```

2589 \ExplSyntaxOn
2590 \cs_gset_protected:Npn
2591 \markdownRendererFancyOlItem
2592 {
2593 \markdownRendererFancyOlItemPrototype
2594 }
2595 \seq_gput_right:Nn
2596 \g_@@_renderers_seq
2597 { fancyOlItem }
2598 \prop_gput:Nnn

```

```

2599 \g_@@_renderer_arities_prop
2600 { fancyO1Item }
2601 { 0 }
2602 \ExplSyntaxOff

```

The `\markdownRendererFancyO1ItemEnd` macro represents the end of an item in a fancy ordered list. This macro will only be produced, when the `fancyLists` option is enabled. The macro receives no arguments.

```

2603 \ExplSyntaxOn
2604 \cs_gset_protected:Npn
2605 \markdownRendererFancyO1ItemEnd
2606 {
2607 \markdownRendererFancyO1ItemEndPrototype
2608 }
2609 \seq_gput_right:Nn
2610 \g_@@_renderers_seq
2611 { fancyO1ItemEnd }
2612 \prop_gput:Nnn
2613 \g_@@_renderer_arities_prop
2614 { fancyO1ItemEnd }
2615 { 0 }
2616 \ExplSyntaxOff

```

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

```

2617 \ExplSyntaxOn
2618 \cs_gset_protected:Npn
2619 \markdownRendererFancyO1ItemWithNumber
2620 {
2621 \markdownRendererFancyO1ItemWithNumberPrototype
2622 }
2623 \seq_gput_right:Nn
2624 \g_@@_renderers_seq
2625 { fancyO1ItemWithNumber }
2626 \prop_gput:Nnn
2627 \g_@@_renderer_arities_prop
2628 { fancyO1ItemWithNumber }
2629 { 1 }
2630 \ExplSyntaxOff

```

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

```

2631 \ExplSyntaxOn
2632 \cs_gset_protected:Npn
2633 \markdownRendererOlEnd
2634 {
2635 \markdownRendererOlEndPrototype
2636 }
2637 \seq_gput_right:Nn
2638 \g_@@_renderers_seq
2639 { olEnd }
2640 \prop_gput:Nnn
2641 \g_@@_renderer_arities_prop
2642 { olEnd }
2643 { 0 }
2644 \ExplSyntaxOff

```

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

```

2645 \ExplSyntaxOn
2646 \cs_gset_protected:Npn
2647 \markdownRendererOlEndTight
2648 {
2649 \markdownRendererOlEndTightPrototype
2650 }
2651 \seq_gput_right:Nn
2652 \g_@@_renderers_seq
2653 { olEndTight }
2654 \prop_gput:Nnn
2655 \g_@@_renderer_arities_prop
2656 { olEndTight }
2657 { 0 }
2658 \ExplSyntaxOff

```

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

```

2659 \ExplSyntaxOn
2660 \cs_gset_protected:Npn
2661 \markdownRendererFancyOlEnd
2662 {
2663 \markdownRendererFancyOlEndPrototype
2664 }
2665 \seq_gput_right:Nn
2666 \g_@@_renderers_seq

```

```

2667 { fancyO1End }
2668 \prop_gput:Nnn
2669 \g_@@_renderer_arities_prop
2670 { fancyO1End }
2671 { 0 }
2672 \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.

```

2673 \ExplSyntaxOn
2674 \cs_gset_protected:Npn
2675 \markdownRendererFancyO1EndTight
2676 {
2677 \markdownRendererFancyO1EndTightPrototype
2678 }
2679 \seq_gput_right:Nn
2680 \g_@@_renderers_seq
2681 { fancyO1EndTight }
2682 \prop_gput:Nnn
2683 \g_@@_renderer_arities_prop
2684 { fancyO1EndTight }
2685 { 0 }
2686 \ExplSyntaxOff

```

### 2.2.5.31 Raw Content Renderers

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

```

2687 \ExplSyntaxOn
2688 \cs_gset_protected:Npn
2689 \markdownRendererInputRawInline
2690 {
2691 \markdownRendererInputRawInlinePrototype
2692 }
2693 \seq_gput_right:Nn
2694 \g_@@_renderers_seq
2695 { inputRawInline }
2696 \prop_gput:Nnn
2697 \g_@@_renderer_arities_prop
2698 { inputRawInline }
2699 { 2 }
2700 \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.

```

2701 \ExplSyntaxOn
2702 \cs_gset_protected:Npn
2703 \markdownRendererInputRawBlock
2704 {
2705 \markdownRendererInputRawBlockPrototype
2706 }
2707 \seq_gput_right:Nn
2708 \g_@@_renderers_seq
2709 { inputRawBlock }
2710 \prop_gput:Nnn
2711 \g_@@_renderer_arities_prop
2712 { inputRawBlock }
2713 { 2 }
2714 \ExplSyntaxOff

```

### 2.2.5.32 Section Renderers

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

```

2715 \ExplSyntaxOn
2716 \cs_gset_protected:Npn
2717 \markdownRendererSectionBegin
2718 {
2719 \markdownRendererSectionBeginPrototype
2720 }
2721 \seq_gput_right:Nn
2722 \g_@@_renderers_seq
2723 { sectionBegin }
2724 \prop_gput:Nnn
2725 \g_@@_renderer_arities_prop
2726 { sectionBegin }
2727 { 0 }
2728 \cs_gset_protected:Npn
2729 \markdownRendererSectionEnd
2730 {
2731 \markdownRendererSectionEndPrototype
2732 }
2733 \seq_gput_right:Nn
2734 \g_@@_renderers_seq
2735 { sectionEnd }
2736 \prop_gput:Nnn
2737 \g_@@_renderer_arities_prop

```

```

2738 { sectionEnd }
2739 { 0 }
2740 \ExplSyntaxOff

```

### 2.2.5.33 Replacement Character Renderers

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

```

2741 \ExplSyntaxOn
2742 \cs_gset_protected:Npn
2743 \markdownRendererReplacementCharacter
2744 {
2745 \markdownRendererReplacementCharacterPrototype
2746 }
2747 \seq_gput_right:Nn
2748 \g_@@_renderers_seq
2749 { replacementCharacter }
2750 \prop_gput:Nnn
2751 \g_@@_renderer_arities_prop
2752 { replacementCharacter }
2753 { 0 }
2754 \ExplSyntaxOff

```

### 2.2.5.34 Special Character Renderers

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

```

2755 \ExplSyntaxOn
2756 \cs_gset_protected:Npn
2757 \markdownRendererLeftBrace
2758 {
2759 \markdownRendererLeftBracePrototype
2760 }
2761 \seq_gput_right:Nn
2762 \g_@@_renderers_seq
2763 { leftBrace }
2764 \prop_gput:Nnn
2765 \g_@@_renderer_arities_prop
2766 { leftBrace }
2767 { 0 }
2768 \cs_gset_protected:Npn
2769 \markdownRendererRightBrace
2770 {
2771 \markdownRendererRightBracePrototype
2772 }
2773 \seq_gput_right:Nn

```



```

2774 \g_@@_renderers_seq
2775 { rightBrace }
2776 \prop_gput:Nnn
2777 \g_@@_renderer_arities_prop
2778 { rightBrace }
2779 { 0 }
2780 \cs_gset_protected:Npn
2781 \markdownRendererDollarSign
2782 {
2783 \markdownRendererDollarSignPrototype
2784 }
2785 \seq_gput_right:Nn
2786 \g_@@_renderers_seq
2787 { dollarSign }
2788 \prop_gput:Nnn
2789 \g_@@_renderer_arities_prop
2790 { dollarSign }
2791 { 0 }
2792 \cs_gset_protected:Npn
2793 \markdownRendererPercentSign
2794 {
2795 \markdownRendererPercentSignPrototype
2796 }
2797 \seq_gput_right:Nn
2798 \g_@@_renderers_seq
2799 { percentSign }
2800 \prop_gput:Nnn
2801 \g_@@_renderer_arities_prop
2802 { percentSign }
2803 { 0 }
2804 \cs_gset_protected:Npn
2805 \markdownRendererAmpersand
2806 {
2807 \markdownRendererAmpersandPrototype
2808 }
2809 \seq_gput_right:Nn
2810 \g_@@_renderers_seq
2811 { ampersand }
2812 \prop_gput:Nnn
2813 \g_@@_renderer_arities_prop
2814 { ampersand }
2815 { 0 }
2816 \cs_gset_protected:Npn
2817 \markdownRendererUnderscore
2818 {
2819 \markdownRendererUnderscorePrototype
2820 }

```

```

2821 \seq_gput_right:Nn
2822 \g_@@_renderers_seq
2823 { underscore }
2824 \prop_gput:Nnn
2825 \g_@@_renderer_arities_prop
2826 { underscore }
2827 { 0 }
2828 \cs_gset_protected:Npn
2829 \markdownRendererHash
2830 {
2831 \markdownRendererHashPrototype
2832 }
2833 \seq_gput_right:Nn
2834 \g_@@_renderers_seq
2835 { hash }
2836 \prop_gput:Nnn
2837 \g_@@_renderer_arities_prop
2838 { hash }
2839 { 0 }
2840 \cs_gset_protected:Npn
2841 \markdownRendererCircumflex
2842 {
2843 \markdownRendererCircumflexPrototype
2844 }
2845 \seq_gput_right:Nn
2846 \g_@@_renderers_seq
2847 { circumflex }
2848 \prop_gput:Nnn
2849 \g_@@_renderer_arities_prop
2850 { circumflex }
2851 { 0 }
2852 \cs_gset_protected:Npn
2853 \markdownRendererBackslash
2854 {
2855 \markdownRendererBackslashPrototype
2856 }
2857 \seq_gput_right:Nn
2858 \g_@@_renderers_seq
2859 { backslash }
2860 \prop_gput:Nnn
2861 \g_@@_renderer_arities_prop
2862 { backslash }
2863 { 0 }
2864 \cs_gset_protected:Npn
2865 \markdownRendererTilde
2866 {
2867 \markdownRendererTildePrototype

```

```

2868 }
2869 \seq_gput_right:Nn
2870 \g_@@_renderers_seq
2871 { tilde }
2872 \prop_gput:Nnn
2873 \g_@@_renderer_arities_prop
2874 { tilde }
2875 { 0 }
2876 \cs_gset_protected:Npn
2877 \markdownRendererPipe
2878 {
2879 \markdownRendererPipePrototype
2880 }
2881 \seq_gput_right:Nn
2882 \g_@@_renderers_seq
2883 { pipe }
2884 \prop_gput:Nnn
2885 \g_@@_renderer_arities_prop
2886 { pipe }
2887 { 0 }
2888 \ExplSyntaxOff

```

### 2.2.5.35 Strike-Through Renderer

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

```

2889 \ExplSyntaxOn
2890 \cs_gset_protected:Npn
2891 \markdownRendererStrikeThrough
2892 {
2893 \markdownRendererStrikeThroughPrototype
2894 }
2895 \seq_gput_right:Nn
2896 \g_@@_renderers_seq
2897 { strikeThrough }
2898 \prop_gput:Nnn
2899 \g_@@_renderer_arities_prop
2900 { strikeThrough }
2901 { 1 }
2902 \ExplSyntaxOff

```

### 2.2.5.36 Subscript Renderer

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

```

2903 \ExplSyntaxOn
2904 \cs_gset_protected:Npn
2905 \markdownRendererSubscript
2906 {
2907 \markdownRendererSubscriptPrototype
2908 }
2909 \seq_gput_right:Nn
2910 \g_@@_renderers_seq
2911 { subscript }
2912 \prop_gput:Nnn
2913 \g_@@_renderer_arities_prop
2914 { subscript }
2915 { 1 }

```

#### 2.2.5.37 Superscript Renderer

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

```

2916 \cs_gset_protected:Npn
2917 \markdownRendererSuperscript
2918 {
2919 \markdownRendererSuperscriptPrototype
2920 }
2921 \seq_gput_right:Nn
2922 \g_@@_renderers_seq
2923 { superscript }
2924 \prop_gput:Nnn
2925 \g_@@_renderer_arities_prop
2926 { superscript }
2927 { 1 }
2928 \ExplSyntaxOff

```

#### 2.2.5.38 Table Attribute Context Renderers

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

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

```

2929 \ExplSyntaxOn
2930 \cs_gset_protected:Npn
2931 \markdownRendererTableAttributeContextBegin

```

```

2932 {
2933 \markdownRendererTableAttributeContextBeginPrototype
2934 }
2935 \seq_gput_right:Nn
2936 \g_@@_renderers_seq
2937 { tableAttributeContextBegin }
2938 \prop_gput:Nnn
2939 \g_@@_renderer_arities_prop
2940 { tableAttributeContextBegin }
2941 { 0 }
2942 \cs_gset_protected:Npn
2943 \markdownRendererTableAttributeContextEnd
2944 {
2945 \markdownRendererTableAttributeContextEndPrototype
2946 }
2947 \seq_gput_right:Nn
2948 \g_@@_renderers_seq
2949 { tableAttributeContextEnd }
2950 \prop_gput:Nnn
2951 \g_@@_renderer_arities_prop
2952 { tableAttributeContextEnd }
2953 { 0 }
2954 \ExplSyntaxOff

```

### 2.2.5.39 Table Renderer

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

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

```

2955 \ExplSyntaxOn
2956 \cs_gset_protected:Npn
2957 \markdownRendererTable
2958 {
2959 \markdownRendererTablePrototype
2960 }
2961 \seq_gput_right:Nn
2962 \g_@@_renderers_seq
2963 { table }

```

```

2964 \prop_gput:Nnn
2965 \g_@@_renderer_arities_prop
2966 { table }
2967 { 3 }
2968 \ExplSyntaxOff

```

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

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

```

2969 \ExplSyntaxOn
2970 \cs_gset_protected:Npn
2971 \markdownRendererInlineMath
2972 {
2973 \markdownRendererInlineMathPrototype
2974 }
2975 \seq_gput_right:Nn
2976 \g_@@_renderers_seq
2977 { inlineMath }
2978 \prop_gput:Nnn
2979 \g_@@_renderer_arities_prop
2980 { inlineMath }
2981 { 1 }
2982 \cs_gset_protected:Npn
2983 \markdownRendererDisplayMath
2984 {
2985 \markdownRendererDisplayMathPrototype
2986 }
2987 \seq_gput_right:Nn
2988 \g_@@_renderers_seq
2989 { displayMath }
2990 \prop_gput:Nnn
2991 \g_@@_renderer_arities_prop
2992 { displayMath }
2993 { 1 }
2994 \ExplSyntaxOff

```

#### 2.2.5.41 Thematic Break Renderer

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

```

2995 \ExplSyntaxOn
2996 \cs_gset_protected:Npn

```

```

2997 \markdownRendererThematicBreak
2998 {
2999 \markdownRendererThematicBreakPrototype
3000 }
3001 \seq_gput_right:Nn
3002 \g_@@_renderers_seq
3003 { thematicBreak }
3004 \prop_gput:Nnn
3005 \g_@@_renderer_arities_prop
3006 { thematicBreak }
3007 { 0 }
3008 \ExplSyntaxOff

```

#### 2.2.5.42 Tickbox Renderers

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

```

3009 \ExplSyntaxOn
3010 \cs_gset_protected:Npn
3011 \markdownRendererTickedBox
3012 {
3013 \markdownRendererTickedBoxPrototype
3014 }
3015 \seq_gput_right:Nn
3016 \g_@@_renderers_seq
3017 { tickedBox }
3018 \prop_gput:Nnn
3019 \g_@@_renderer_arities_prop
3020 { tickedBox }
3021 { 0 }
3022 \cs_gset_protected:Npn
3023 \markdownRendererHalfTickedBox
3024 {
3025 \markdownRendererHalfTickedBoxPrototype
3026 }
3027 \seq_gput_right:Nn
3028 \g_@@_renderers_seq
3029 { halfTickedBox }
3030 \prop_gput:Nnn
3031 \g_@@_renderer_arities_prop
3032 { halfTickedBox }
3033 { 0 }
3034 \cs_gset_protected:Npn
3035 \markdownRendererUntickedBox

```

```

3036 {
3037 \markdownRendererUntickedBoxPrototype
3038 }
3039 \seq_gput_right:Nn
3040 \g_@@_renderers_seq
3041 { untickedBox }
3042 \prop_gput:Nnn
3043 \g_@@_renderer_arities_prop
3044 { untickedBox }
3045 { 0 }
3046 \ExplSyntaxOff

```

### 2.2.5.43 Warning and Error Renderers

The `\markdownRendererWarning` and `\markdownRendererError` macros represent warnings and errors produced by the markdown parser. Both macros receive four parameters:

1. The fully escaped text of the warning or error that can be directly typeset
2. The raw text of the warning or error that can be used outside typesetting for e.g. logging the warning or error.
3. The fully escaped text with more details about the warning or error that can be directly typeset. Can be empty, unlike the first two parameters.
4. The raw text with more details about the warning or error that can be used outside typesetting for e.g. logging the warning or error. Can be empty, unlike the first two parameters.

```

3047 \ExplSyntaxOn
3048 \cs_gset_protected:Npn
3049 \markdownRendererWarning
3050 {
3051 \markdownRendererWarningPrototype
3052 }
3053 \cs_gset_protected:Npn
3054 \markdownRendererError
3055 {
3056 \markdownRendererErrorPrototype
3057 }
3058 \seq_gput_right:Nn
3059 \g_@@_renderers_seq
3060 { warning }
3061 \prop_gput:Nnn
3062 \g_@@_renderer_arities_prop
3063 { warning }
3064 { 4 }
3065 \seq_gput_right:Nn
3066 \g_@@_renderers_seq

```



```

3067 { error }
3068 \prop_gput:Nnn
3069 \g_@@_renderer_arities_prop
3070 { error }
3071 { 4 }
3072 \ExplSyntaxOff

```

#### 2.2.5.44 YAML Metadata Renderers

The `\markdownRendererJekyllDataBegin` macro represents the beginning of a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives no arguments.

```

3073 \ExplSyntaxOn
3074 \cs_gset_protected:Npn
3075 \markdownRendererJekyllDataBegin
3076 {
3077 \markdownRendererJekyllDataBeginPrototype
3078 }
3079 \seq_gput_right:Nn
3080 \g_@@_renderers_seq
3081 { jekyllDataBegin }
3082 \prop_gput:Nnn
3083 \g_@@_renderer_arities_prop
3084 { jekyllDataBegin }
3085 { 0 }
3086 \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.

```

3087 \ExplSyntaxOn
3088 \cs_gset_protected:Npn
3089 \markdownRendererJekyllDataEnd
3090 {
3091 \markdownRendererJekyllDataEndPrototype
3092 }
3093 \seq_gput_right:Nn
3094 \g_@@_renderers_seq
3095 { jekyllDataEnd }
3096 \prop_gput:Nnn
3097 \g_@@_renderer_arities_prop
3098 { jekyllDataEnd }
3099 { 0 }
3100 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataMappingBegin` macro represents the beginning of a mapping in a YAML document. This macro will only be produced when the

`jeekyllData` 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.

```

3101 \ExplSyntaxOn
3102 \cs_gset_protected:Npn
3103 \markdownRendererJekyllDataMappingBegin
3104 {
3105 \markdownRendererJekyllDataMappingBeginPrototype
3106 }
3107 \seq_gput_right:Nn
3108 \g_@@_renderers_seq
3109 { jeekyllDataMappingBegin }
3110 \prop_gput:Nnn
3111 \g_@@_renderer_arities_prop
3112 { jeekyllDataMappingBegin }
3113 { 2 }
3114 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataMappingEnd` macro represents the end of a mapping in a YAML document. This macro will only be produced when the `jeekyllData` option is enabled. The macro receives no arguments.

```

3115 \ExplSyntaxOn
3116 \cs_gset_protected:Npn
3117 \markdownRendererJekyllDataMappingEnd
3118 {
3119 \markdownRendererJekyllDataMappingEndPrototype
3120 }
3121 \seq_gput_right:Nn
3122 \g_@@_renderers_seq
3123 { jeekyllDataMappingEnd }
3124 \prop_gput:Nnn
3125 \g_@@_renderer_arities_prop
3126 { jeekyllDataMappingEnd }
3127 { 0 }
3128 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataSequenceBegin` macro represents the beginning of a sequence in a YAML document. This macro will only be produced when the `jeekyllData` 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.

```

3129 \ExplSyntaxOn
3130 \cs_gset_protected:Npn
3131 \markdownRendererJekyllDataSequenceBegin
3132 {
3133 \markdownRendererJekyllDataSequenceBeginPrototype

```

```

3134 }
3135 \seq_gput_right:Nn
3136 \g_@@_renderers_seq
3137 { jekyllDataSequenceBegin }
3138 \prop_gput:Nnn
3139 \g_@@_renderer_arities_prop
3140 { jekyllDataSequenceBegin }
3141 { 2 }
3142 \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.

```

3143 \ExplSyntaxOn
3144 \cs_gset_protected:Npn
3145 \markdownRendererJekyllDataSequenceEnd
3146 {
3147 \markdownRendererJekyllDataSequenceEndPrototype
3148 }
3149 \seq_gput_right:Nn
3150 \g_@@_renderers_seq
3151 { jekyllDataSequenceEnd }
3152 \prop_gput:Nnn
3153 \g_@@_renderer_arities_prop
3154 { jekyllDataSequenceEnd }
3155 { 0 }
3156 \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.

```

3157 \ExplSyntaxOn
3158 \cs_gset_protected:Npn
3159 \markdownRendererJekyllDataBoolean
3160 {
3161 \markdownRendererJekyllDataBooleanPrototype
3162 }
3163 \seq_gput_right:Nn
3164 \g_@@_renderers_seq
3165 { jekyllDataBoolean }
3166 \prop_gput:Nnn
3167 \g_@@_renderer_arities_prop
3168 { jekyllDataBoolean }
3169 { 2 }

```

```
3170 \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.

```
3171 \ExplSyntaxOn
3172 \cs_gset_protected:Npn
3173 \markdownRendererJekyllDataNumber
3174 {
3175 \markdownRendererJekyllDataNumberPrototype
3176 }
3177 \seq_gput_right:Nn
3178 \g_@@_renderers_seq
3179 { jekyllDataNumber }
3180 \prop_gput:Nnn
3181 \g_@@_renderer_arities_prop
3182 { jekyllDataNumber }
3183 { 2 }
3184 \ExplSyntaxOff
```

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

For each string scalar value, both macros are produced. Whereas `\markdownRendererJekyllDataTypographicString` receives the scalar value after all markdown markup and special  $\text{\TeX}$  characters in the string have been replaced by  $\text{\TeX}$  macros, `\markdownRendererJekyllDataProgrammaticString` receives the raw scalar value. Therefore, whereas the `\markdownRendererJekyllDataTypographicString` macro is more appropriate for texts that are supposed to be typeset with  $\text{\TeX}$ , such as document titles, author names, or exam questions, the `\markdownRendererJekyllDataProgrammaticString` macro is more appropriate for identifiers and other programmatic text that won't be typeset by  $\text{\TeX}$ .

```
3185 \ExplSyntaxOn
3186 \cs_gset_protected:Npn
3187 \markdownRendererJekyllDataTypographicString
3188 {
3189 \markdownRendererJekyllDataTypographicStringPrototype
3190 }
3191 \cs_gset_protected:Npn
3192 \markdownRendererJekyllDataProgrammaticString
3193 {
3194 \markdownRendererJekyllDataProgrammaticStringPrototype
3195 }
3196 \ExplSyntaxOff
```

```

3195 }
3196 \seq_gput_right:Nn
3197 \g_@@_renderers_seq
3198 { jekyllDataTypographicString }
3199 \prop_gput:Nnn
3200 \g_@@_renderer_arities_prop
3201 { jekyllDataTypographicString }
3202 { 2 }
3203 \seq_gput_right:Nn
3204 \g_@@_renderers_seq
3205 { jekyllDataProgrammaticString }
3206 \prop_gput:Nnn
3207 \g_@@_renderer_arities_prop
3208 { jekyllDataProgrammaticString }
3209 { 2 }
3210 \ExplSyntaxOff

```

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

```

3211 \ExplSyntaxOn
3212 \cs_gset:Npn
3213 \markdownRendererJekyllDataTypographicString
3214 {
3215 \cs_if_exist:NTF
3216 \markdownRendererJekyllDataString
3217 {
3218 \@@_if_option:nTF
3219 { experimental }
3220 {
3221 \markdownError
3222 {
3223 The~jekyllDataString~renderer~has~been~deprecated,~
3224 to~be~removed~in~Markdown~4.0.0
3225 }
3226 }
3227 {
3228 \markdownWarning
3229 {
3230 The~jekyllDataString~renderer~has~been~deprecated,~
3231 to~be~removed~in~Markdown~4.0.0
3232 }
3233 \markdownRendererJekyllDataString
3234 }
3235 }
3236 }

```

```

3237 \cs_if_exist:NTF
3238 \markdownRendererJekyllDataStringPrototype
3239 {
3240 \@@_if_option:NTF
3241 { experimental }
3242 {
3243 \markdownError
3244 {
3245 The~jekyllDataString~renderer~prototype~
3246 has~been~deprecated,~
3247 to~be~removed~in~Markdown~4.0.0
3248 }
3249 }
3250 {
3251 \markdownWarning
3252 {
3253 The~jekyllDataString~renderer~prototype~
3254 has~been~deprecated,~
3255 to~be~removed~in~Markdown~4.0.0
3256 }
3257 \markdownRendererJekyllDataStringPrototype
3258 }
3259 }
3260 {
3261 \markdownRendererJekyllDataTypographicStringPrototype
3262 }
3263 }
3264 }
3265 \seq_gput_right:Nn
3266 \g_@@_renderers_seq
3267 { jekyllDataString }
3268 \prop_gput:Nnn
3269 \g_@@_renderer_arities_prop
3270 { jekyllDataString }
3271 { 2 }
3272 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataEmpty` macro represents an empty scalar value in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives one argument: the scalar key in the parent structure, cast to a string following YAML serialization rules.

See also Section 2.2.6.1 for the description of the high-level `expl3` interface that you can also use to react to YAML metadata.

```

3273 \ExplSyntaxOn
3274 \cs_gset_protected:Npn
3275 \markdownRendererJekyllDataEmpty
3276 {

```

```

3277 \markdownRendererJekyllDataEmptyPrototype
3278 }
3279 \seq_gput_right:Nn
3280 \g_@@_renderers_seq
3281 { jekyllDataEmpty }
3282 \prop_gput:Nnn
3283 \g_@@_renderer_arities_prop
3284 { jekyllDataEmpty }
3285 { 1 }
3286 \ExplSyntaxOff

```

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

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

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

```

3287 \ExplSyntaxOn
3288 \cs_new:Nn \@@_define_renderers:
3289 {
3290 \seq_map_inline:Nn
3291 \g_@@_renderers_seq
3292 {
3293 \@@_define_renderer:n
3294 { ##1 }
3295 }
3296 }
3297 \cs_new:Nn \@@_define_renderer:n
3298 {
3299 \@@_renderer_tl_to_csname:nN
3300 { #1 }
3301 \l_tmpa_tl
3302 \prop_get:NnN
3303 \g_@@_renderer_arities_prop
3304 { #1 }
3305 \l_tmpb_tl
3306 \@@_define_renderer:ncV
3307 { #1 }
3308 { \l_tmpa_tl }
3309 \l_tmpb_tl
3310 }
3311 \cs_new:Nn \@@_renderer_tl_to_csname:nN

```

```

3312 {
3313 \tl_set:Nn
3314 \l_tmpa_tl
3315 { \str_uppercase:n { #1 } }
3316 \tl_set:Nx
3317 #2
3318 {
3319 markdownRenderer
3320 \tl_head:f { \l_tmpa_tl }
3321 \tl_tail:n { #1 }
3322 }
3323 }
3324 \tl_new:N
3325 \l_@@_renderer_definition_tl
3326 \bool_new:N
3327 \g_@@_appending_renderer_bool
3328 \bool_new:N
3329 \g_@@_unprotected_renderer_bool
3330 \cs_new:Nn \@@_define_renderer:nNn
3331 {
3332 \keys_define:nn
3333 { markdown/options/renderers }
3334 {
3335 #1 .code:n = {
3336 \tl_set:Nn
3337 \l_@@_renderer_definition_tl
3338 { ##1 }
3339 \regex_replace_all:nnN
3340 { \cP\#0 }
3341 { #1 }
3342 \l_@@_renderer_definition_tl
3343 \bool_if:NT
3344 \g_@@_appending_renderer_bool
3345 {
3346 \@@_tl_set_from_cs:NNn
3347 \l_tmpa_tl
3348 #2
3349 { #3 }
3350 \tl_put_left:NV
3351 \l_@@_renderer_definition_tl
3352 \l_tmpa_tl
3353 }
3354 \bool_if:NTF
3355 \g_@@_unprotected_renderer_bool
3356 {
3357 \tl_set:Nn
3358 \l_tmpa_tl

```



```

3359 { \cs_set:Npn }
3360 }
3361 {
3362 \tl_set:Nn
3363 \l_tmpa_tl
3364 { \cs_set_protected:Npn }
3365 }
3366 \exp_last_unbraced:NNV
3367 \cs_generate_from_arg_count:NNnV
3368 #2
3369 \l_tmpa_tl
3370 { #3 }
3371 \l_@@_renderer_definition_tl
3372 },
3373 }

```

If the token renderer macro has been deprecated, we undefine it.

The `\markdownRendererJekyllDataString` macro has been deprecated and will be removed in Markdown 4.0.0.

```

3374 \str_if_eq:nnT
3375 { #1 }
3376 { jekyllDataString }
3377 {
3378 \cs_undefine:N
3379 #2
3380 }
3381 }

```

We define the function `\@@_tl_set_from_cs:NNn` [12]. The function takes a token list, a control sequence with undelimited parameters, and the number of parameters the control sequence accepts, and locally assigns the replacement text of the control sequence to the token list.

```

3382 \cs_new_protected:Nn
3383 \@@_tl_set_from_cs:NNn
3384 {
3385 \tl_set:Nn
3386 \l_tmpa_tl
3387 { #2 }
3388 \int_step_inline:nn
3389 { #3 }
3390 {
3391 \exp_args:NNc
3392 \tl_put_right:Nn
3393 \l_tmpa_tl
3394 { @@_tl_set_from_cs_parameter_ ##1 }
3395 }
3396 \exp_args:NNV

```

```

3397 \tl_set:No
3398 \l_tmpb_tl
3399 \l_tmpa_tl
3400 \regex_replace_all:nnN
3401 { \cP. }
3402 { \0\0 }
3403 \l_tmpb_tl
3404 \int_step_inline:nn
3405 { #3 }
3406 {
3407 \regex_replace_all:nnN
3408 { \c { @@_tl_set_from_cs_parameter_ ##1 } }
3409 { \cP\# ##1 }
3410 \l_tmpb_tl
3411 }
3412 \tl_set:NV
3413 #1
3414 \l_tmpb_tl
3415 }
3416 \cs_generate_variant:Nn
3417 \@@_define_renderer:nNn
3418 { ncV }
3419 \cs_generate_variant:Nn
3420 \cs_generate_from_arg_count:NNnn
3421 { NNnV }
3422 \cs_generate_variant:Nn
3423 \tl_put_left:Nn
3424 { Nv }
3425 \keys_define:nn
3426 { markdown/options }
3427 {
3428 renderers .code:n = {
3429 \bool_gset_false:N
3430 \g_@@_unprotected_renderer_bool
3431 \keys_set:nn
3432 { markdown/options/renderers }
3433 { #1 }
3434 },
3435 unprotectedRenderers .code:n = {
3436 \bool_gset_true:N
3437 \g_@@_unprotected_renderer_bool
3438 \keys_set:nn
3439 { markdown/options/renderers }
3440 { #1 }
3441 },
3442 }

```

The following example code showcases a possible configuration of the `\markdownRendererLink` and `\markdownRendererEmphasis` token renderer macros.

```
\markdownSetup{
 renderers = {
 link = {#4}, % Render links as the link title.
 emphasis = {{\it #1}}, % Render emphasized text using italics.
 }
}
```

```
3443 \tl_new:N
3444 \l_@@_renderer_glob_definition_tl
3445 \seq_new:N
3446 \l_@@_renderer_glob_results_seq
3447 \regex_const:Nn
3448 \c_@@_appending_key_regex
3449 { \s*+ $ }
3450 \keys_define:nn
3451 { markdown/options/renderers }
3452 {
3453 unknown .code:n = {
```

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

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

```

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

```

```

3454 \regex_match:NVTF
3455 \c_@@_appending_key_regex
3456 \l_keys_key_str
3457 {
3458 \bool_gset_true:N
3459 \g_@@_appending_renderer_bool
3460 \tl_set:NV
3461 \l_tmpa_tl
3462 \l_keys_key_str
3463 \regex_replace_once:NnN
3464 \c_@@_appending_key_regex
3465 { }
3466 \l_tmpa_tl
3467 \tl_set:Nx
3468 \l_tmpb_tl
3469 { { \l_tmpa_tl } = }
3470 \tl_put_right:Nn
3471 \l_tmpb_tl
3472 { { #1 } }
3473 \keys_set:nV
3474 { markdown/options/renderers }
3475 \l_tmpb_tl
3476 \bool_gset_false:N
3477 \g_@@_appending_renderer_bool
3478 }

```

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

```

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

```

Wildcards and enumerations can be combined:

```

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

```

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

```

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

```

```

3479 {
3480 \@@_glob_seq:VnN
3481 \l_keys_key_str
3482 { g_@@_renderers_seq }
3483 \l_@@_renderer_glob_results_seq
3484 \seq_if_empty:NTF
3485 \l_@@_renderer_glob_results_seq
3486 {
3487 \msg_error:nnV
3488 { markdown }
3489 { undefined-renderer }
3490 \l_keys_key_str
3491 }
3492 {
3493 \tl_set:Nn
3494 \l_@@_renderer_glob_definition_tl
3495 { \exp_not:n { #1 } }
3496 \seq_map_inline:Nn
3497 \l_@@_renderer_glob_results_seq
3498 {
3499 \tl_set:Nn
3500 \l_tmpa_tl
3501 { { ##1 } = }
3502 \tl_put_right:Nx
3503 \l_tmpa_tl
3504 { { \l_@@_renderer_glob_definition_tl } }
3505 \keys_set:nV
3506 { markdown/options/renderers }
3507 \l_tmpa_tl
3508 }

```

```

3509 }
3510 }
3511 },
3512 }
3513 \msg_new:nnn
3514 { markdown }
3515 { undefined-renderer }
3516 {
3517 Renderer~#1~is~undefined.
3518 }
3519 \cs_generate_variant:Nn
3520 \@@_glob_seq:nnN
3521 { VnN }
3522 \cs_generate_variant:Nn
3523 \cs_generate_from_arg_count:NNnn
3524 { cNvV }
3525 \cs_generate_variant:Nn
3526 \msg_error:nnn
3527 { nnV }
3528 \prg_generate_conditional_variant:Nnn
3529 \regex_match:Nn
3530 { NV }
3531 { TF }
3532 \prop_new:N
3533 \g_@@_glob_cache_prop
3534 \tl_new:N
3535 \l_@@_current_glob_tl
3536 \cs_new:Nn
3537 \@@_glob_seq:nnN
3538 {
3539 \tl_set:Nn
3540 \l_@@_current_glob_tl
3541 { ^ #1 $ }
3542 \prop_get:NeNTF
3543 \g_@@_glob_cache_prop
3544 { #2 / \l_@@_current_glob_tl }
3545 \l_tmpa_clist
3546 {
3547 \seq_set_from_clist:NN
3548 #3
3549 \l_tmpa_clist
3550 }
3551 {
3552 \seq_clear:N
3553 #3
3554 \regex_replace_all:nnN
3555 { * }

```

```

3556 { .* }
3557 \l_@@_current_glob_tl
3558 \regex_set:NV
3559 \l_tmpa_regex
3560 \l_@@_current_glob_tl
3561 \seq_map_inline:cn
3562 { #2 }
3563 {
3564 \regex_match:NnT
3565 \l_tmpa_regex
3566 { ##1 }
3567 {
3568 \seq_put_right:Nn
3569 #3
3570 { ##1 }
3571 }
3572 }
3573 \clist_set_from_seq:NN
3574 \l_tmpa_clist
3575 #3
3576 \prop_gput:NeV
3577 \g_@@_glob_cache_prop
3578 { #2 / \l_@@_current_glob_tl }
3579 \l_tmpa_clist
3580 }
3581 }
3582 \cs_generate_variant:Nn
3583 \regex_set:Nn
3584 { NV }
3585 \cs_generate_variant:Nn
3586 \prop_gput:Nnn
3587 { NeV }

```

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

```

3588 \str_if_eq:VVT
3589 \c_@@_top_layer_tl
3590 \c_@@_option_layer_plain_tex_tl
3591 {
3592 \@@_define_renderers:
3593 }
3594 \ExplSyntaxOff

```

## 2.2.6 Token Renderer Prototypes

### 2.2.6.1 YAML Metadata Renderer Prototypes

For simple YAML metadata, a simple high-level interface is provided that can be programmed by setting the expl3 key-values [2] for the module `markdown/jekyllData`.

```

3595 \ExplSyntaxOn
3596 \keys_define:nn
3597 { markdown/jekyllData }
3598 { }
3599 \ExplSyntaxOff

```

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

```

3600 \ExplSyntaxOn
3601 \@@_with_various_cases:nn
3602 { jekyllDataRenderers }
3603 {
3604 \keys_define:nn
3605 { markdown/options }
3606 {
3607 #1 .code:n = {
3608 \tl_set:Nn
3609 \l_tmpa_tl
3610 { ##1 }

```

To ensure that keys containing forward slashes get passed correctly, we replace all forward slashes in the input with backslash tokens with category code letter and then undo the replacement. This means that if any unbraced backslash tokens with category code letter exist in the input, they will be replaced with forward slashes. However, this should be extremely rare.

```

3611 \tl_replace_all:NnV
3612 \l_tmpa_tl
3613 { / }
3614 \c_backslash_str
3615 \keys_set:nV
3616 { markdown/options/jekyll-data-renderers }
3617 \l_tmpa_tl
3618 },
3619 }
3620 }
3621 \keys_define:nn
3622 { markdown/options/jekyll-data-renderers }
3623 {
3624 unknown .code:n = {
3625 \tl_set_eq:NN
3626 \l_tmpa_tl
3627 \l_keys_key_str
3628 \tl_replace_all:NVn
3629 \l_tmpa_tl
3630 \c_backslash_str

```



```

3631 { / }
3632 \tl_put_right:Nn
3633 \l_tmpa_tl
3634 {
3635 .code:n = { #1 }
3636 }
3637 \keys_define:nV
3638 { markdown/jekyllData }
3639 \l_tmpa_tl
3640 }
3641 }
3642 \cs_generate_variant:Nn
3643 \keys_define:nn
3644 { nV }
3645 \ExplSyntaxOff

```

For complex YAML metadata, the option `jekyllDataKeyValue=<module>` [13] can be used to route the processing of all YAML metadata in the current  $\text{\TeX}$  group to the key-values from  $\langle module \rangle$ .

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

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

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

```

3646 \ExplSyntaxOn
3647 \cs_new:Nn \@@_define_renderer_prototypes:
3648 {
3649 \seq_map_inline:Nn
3650 \g_@@_renderers_seq
3651 {
3652 \@@_define_renderer_prototype:n
3653 { ##1 }
3654 }
3655 }
3656 \cs_new:Nn \@@_define_renderer_prototype:n
3657 {
3658 \@@_renderer_prototype_tl_to_csname:nN

```

```

3659 { #1 }
3660 \l_tmpa_tl
3661 \prop_get:NnN
3662 \g_@@_renderer_arities_prop
3663 { #1 }
3664 \l_tmpb_tl
3665 \@@_define_renderer_prototype:ncV
3666 { #1 }
3667 { \l_tmpa_tl }
3668 \l_tmpb_tl
3669 }
3670 \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
3671 {
3672 \tl_set:Nn
3673 \l_tmpa_tl
3674 { \str_uppercase:n { #1 } }
3675 \tl_set:Nx
3676 #2
3677 {
3678 markdownRenderer
3679 \tl_head:f { \l_tmpa_tl }
3680 \tl_tail:n { #1 }
3681 Prototype
3682 }
3683 }
3684 \tl_new:N
3685 \l_@@_renderer_prototype_definition_tl
3686 \bool_new:N
3687 \g_@@_appending_renderer_prototype_bool
3688 \bool_new:N
3689 \g_@@_unprotected_renderer_prototype_bool
3690 \cs_new:Nn \@@_define_renderer_prototype:nNn
3691 {
3692 \keys_define:nn
3693 { markdown/options/renderer-prototypes }
3694 {
3695 #1 .code:n = {
3696 \tl_set:Nn
3697 \l_@@_renderer_prototype_definition_tl
3698 { ##1 }
3699 \regex_replace_all:nnN
3700 { \cP\#0 }
3701 { #1 }
3702 \l_@@_renderer_prototype_definition_tl
3703 \bool_if:NT
3704 \g_@@_appending_renderer_prototype_bool
3705 {

```

```

3706 \@@_tl_set_from_cs:NNn
3707 \l_tmpa_tl
3708 #2
3709 { #3 }
3710 \tl_put_left:NV
3711 \l_@@_renderer_prototype_definition_tl
3712 \l_tmpa_tl
3713 }
3714 \bool_if:NTF
3715 \g_@@_unprotected_renderer_prototype_bool
3716 {
3717 \tl_set:Nn
3718 \l_tmpa_tl
3719 { \cs_set:Npn }
3720 }
3721 {
3722 \tl_set:Nn
3723 \l_tmpa_tl
3724 { \cs_set_protected:Npn }
3725 }
3726 \exp_last_unbraced:NNV
3727 \cs_generate_from_arg_count:NNnV
3728 #2
3729 \l_tmpa_tl
3730 { #3 }
3731 \l_@@_renderer_prototype_definition_tl
3732 },
3733 }

```

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

The `\markdownRendererJekyllDataStringPrototype` macro has been deprecated and will be removed in Markdown 4.0.0.

```

3734 \str_if_eq:nnF
3735 { #1 }
3736 { jekyllDataString }
3737 {
3738 \cs_if_free:NT
3739 #2
3740 {
3741 \cs_generate_from_arg_count:NNnn
3742 #2
3743 \cs_gset_protected:Npn
3744 { #3 }
3745 { }
3746 }
3747 }

```

```

3748 }
3749 \cs_generate_variant:Nn
3750 \@@_define_renderer_prototype:nNn
3751 { ncV }

```

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

```

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

```

```

3752 \keys_define:nn
3753 { markdown/options/renderer-prototypes }
3754 {
3755 unknown .code:n = {

```

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

```

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

```

```

 },
 },
}

```

```

3756 \regex_match:NVTF
3757 \c_@@_appending_key_regex
3758 \l_keys_key_str
3759 {
3760 \bool_gset_true:N
3761 \g_@@_appending_renderer_prototype_bool
3762 \tl_set:NV
3763 \l_tmpa_tl
3764 \l_keys_key_str
3765 \regex_replace_once:NnN
3766 \c_@@_appending_key_regex
3767 { }
3768 \l_tmpa_tl
3769 \tl_set:Nx
3770 \l_tmpb_tl
3771 { { \l_tmpa_tl } = }
3772 \tl_put_right:Nn
3773 \l_tmpb_tl
3774 { { #1 } }
3775 \keys_set:nV
3776 { markdown/options/renderer-prototypes }
3777 \l_tmpb_tl
3778 \bool_gset_false:N
3779 \g_@@_appending_renderer_prototype_bool
3780 }

```

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

```

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

```

Wildcards and enumerations can be combined:

```

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

```

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

```

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

```

```

3781 {
3782 \@@_glob_seq:VnN
3783 \l_keys_key_str
3784 { g_@@_renderers_seq }
3785 \l_@@_renderer_glob_results_seq
3786 \seq_if_empty:NTF
3787 \l_@@_renderer_glob_results_seq
3788 {
3789 \msg_error:nnV
3790 { markdown }
3791 { undefined-renderer-prototype }
3792 \l_keys_key_str
3793 }
3794 {
3795 \tl_set:Nn
3796 \l_@@_renderer_glob_definition_tl
3797 { \exp_not:n { #1 } }
3798 \seq_map_inline:Nn
3799 \l_@@_renderer_glob_results_seq
3800 {
3801 \tl_set:Nn
3802 \l_tmpa_tl
3803 { { ##1 } = }
3804 \tl_put_right:Nx
3805 \l_tmpa_tl
3806 { { \l_@@_renderer_glob_definition_tl } }
3807 \keys_set:nV
3808 { markdown/options/renderer-prototypes }
3809 \l_tmpa_tl

```

```

3810 }
3811 }
3812 }
3813 },
3814 }
3815 \msg_new:nnn
3816 { markdown }
3817 { undefined-renderer-prototype }
3818 {
3819 Renderer~prototype~#1~is~undefined.
3820 }
3821 \@@_with_various_cases:nn
3822 { rendererPrototypes }
3823 {
3824 \keys_define:nn
3825 { markdown/options }
3826 {
3827 #1 .code:n = {
3828 \bool_gset_false:N
3829 \g_@@_unprotected_renderer_prototype_bool
3830 \keys_set:nn
3831 { markdown/options/renderer-prototypes }
3832 { ##1 }
3833 },
3834 }
3835 }
3836 \@@_with_various_cases:nn
3837 { unprotectedRendererPrototypes }
3838 {
3839 \keys_define:nn
3840 { markdown/options }
3841 {
3842 #1 .code:n = {
3843 \bool_gset_true:N
3844 \g_@@_unprotected_renderer_prototype_bool
3845 \keys_set:nn
3846 { markdown/options/renderer-prototypes }
3847 { ##1 }
3848 },
3849 }
3850 }

```

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

```

3851 \str_if_eq:VVT
3852 \c_@@_top_layer_tl

```

```

3853 \c_@@_option_layer_plain_tex_tl
3854 {
3855 \@@_define_renderer_prototypes:
3856 }
3857 \ExplSyntaxOff

```

### 2.2.7 Logging Facilities

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

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

### 2.2.8 Miscellanea

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

```

3858 \let\markdownMakeOther\relax

```

The `\markdownReadAndConvert` macro implements the `\markdownBegin` and `\yamlBegin` macros. The first argument specifies the token sequence that will terminate the markdown input when the plain T<sub>E</sub>X special characters have had their category changed to *other*: `\markdownEnd` for the `\markdownBegin` macro and `\yamlEnd` for the `\yamlBegin` macro. The second argument specifies the token sequence that will actually be inserted into the document, when the ending token sequence has been found.

```

3859 \let\markdownReadAndConvert\relax
3860 \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*.

```

3861 \catcode`\|=0\catcode`\=12%
3862 |gdef|markdownBegin{%
3863 |markdownReadAndConvert{\markdownEnd}%
3864 {\|markdownEnd}}%
3865 |gdef|yamlBegin{%
3866 |begingroup
3867 |yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%

```



```

3868 |markdownReadAndConvert{\yamlEnd}%
3869 |{yamlEnd}}%
3870 |endgroup

```

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

The `code` key, which can be used to immediately expand and execute code.

```

3871 \ExplSyntaxOn
3872 \keys_define:nn
3873 { markdown/options }
3874 {
3875 code .code:n = { #1 },
3876 }
3877 \ExplSyntaxOff

```

This can be especially useful in snippets.

## 2.3 L<sup>A</sup>T<sub>E</sub>X Interface

The L<sup>A</sup>T<sub>E</sub>X interface provides L<sup>A</sup>T<sub>E</sub>X environments for the typesetting of markdown input from within L<sup>A</sup>T<sub>E</sub>X, facilities for setting Lua, plain T<sub>E</sub>X, and L<sup>A</sup>T<sub>E</sub>X options used during the conversion from markdown to plain T<sub>E</sub>X, and facilities for changing the way markdown tokens are rendered. The rest of the interface is inherited from the plain T<sub>E</sub>X interface (see Section 2.2).

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

```

3878 \ExplSyntaxOn
3879 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
3880 \cs_generate_variant:Nn
3881 \tl_const:Nn
3882 { NV }
3883 \tl_if_exist:NF
3884 \c_@@_top_layer_tl
3885 {
3886 \tl_const:NV
3887 \c_@@_top_layer_tl
3888 \c_@@_option_layer_latex_tl
3889 }
3890 \ExplSyntaxOff
3891 \input markdown/markdown

```

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

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

where  $\langle options \rangle$  are the  $\text{\LaTeX}$  interface options (see Section 2.3.3). Note that  $\langle options \rangle$  inside the `\usepackage` macro may not set the `markdownRenderers` (see Section 2.2.5.45) and `markdownRendererPrototypes` (see Section 2.2.6.2) keys. Furthermore, although the base variant of the `import` key that loads a single  $\text{\LaTeX}$  theme (see Section 2.3.4) can be used, the extended variant that can load multiple themes and import snippets from them (see Section 2.2.4) cannot. This limitation is due to the way  $\text{\LaTeX} 2_{\epsilon}$  parses package options.

### 2.3.1 Typesetting Markdown

The interface exposes the `markdown`, `markdown*`, and `yaml`  $\text{\LaTeX}$  environments, and redefines the `\markinline`, `\markdownInput`, and `\yamlInput` commands.

#### 2.3.1.1 Typesetting Markdown and YAML directly

The `markdown` and `markdown*`  $\text{\LaTeX}$  environments are aliases for the macros `\markdownBegin` and `\markdownEnd` exposed by the plain  $\text{\TeX}$  interface.

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

```
3892 \newenvironment{markdown}\relax\relax
3893 \newenvironment{markdown*}[1]\relax\relax
```

Furthermore, both environments accept  $\text{\LaTeX}$  interface options (see Section 2.3.3) as the only argument. This argument is optional for the `markdown` environment and mandatory for the `markdown*` environment.

The `markdown` and `markdown*` environments are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

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

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

You can't directly extend the `markdown`  $\text{\LaTeX}$  environment by using it in other environments as follows:

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

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

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

Specifically, the `\markdown` macro must appear at the end of the replacement before-text and must be followed by text that has not yet been ingested by TeX's input processor.

Furthermore, using the `\markdownEnd` macro in of after the replacement after-text is optional and only makes a difference if you redefined it to produce special effects before and after the `markdown` L<sup>A</sup>T<sub>E</sub>X environment.

Lastly, you can't nest the other environments. For example, the following definition would be incorrect:

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

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

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

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

```
3894 \newenvironment{yaml}\relax\relax
```

Furthermore, the environment accepts L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.3) as the only optional argument.

The `yaml` environment is subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

The following example L<sup>A</sup>T<sub>E</sub>X code showcases the usage of the `yaml` environment:

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\begin{yaml}[smartEllipses]
title: _Hello_ **world** ...
author: John Doe
\end{yaml}
\end{document}
```

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

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\begin{markdown}[
 jekyllData,
 expectJekyllData,
 ensureJekyllData,
 smartEllipses,
]
title: _Hello_ **world** ...
author: John Doe
\end{markdown}
\end{document}
```

You can't directly extend the `yaml` L<sup>A</sup>T<sub>E</sub>X environment by using it in other environments. However, you can work around this limitation by using the `\yaml` and `\yamlEnd` macros directly in the definition, similarly to the `\markdown` and `\markdownEnd` macros described previously. Unlike with the `\markdown` and `\markdownEnd` macros, The `\yamlEnd` macro `_must_` be used in or after the replacement after-text.

The `\markinline` macro accepts a single mandatory parameter containing inline markdown content and expands to the result of the conversion of the input markdown document to plain T<sub>E</sub>X. Unlike the `\markinline` macro provided by the plain T<sub>E</sub>X interface, this macro also accepts L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.3) as its optional argument. These options will only influence this markdown content.

### 2.3.1.2 Typesetting Markdown and YAML from external documents

The `\markdownInput` macro accepts a single mandatory parameter containing the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain T<sub>E</sub>X. Unlike the `\markdownInput` macro provided by the plain T<sub>E</sub>X interface, this macro also accepts L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.3) as its optional argument. These options will only influence this markdown document.

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

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

```
\end{document}
```

The `\yamlInput` macro accepts a single mandatory parameter containing the filename of a YAML document and expands to the result of the conversion of the input YAML document to plain  $\text{\TeX}$ . Unlike the `\yamlInput` macro provided by the plain  $\text{\TeX}$  interface, this macro also accepts  $\text{\LaTeX}$  interface options (see Section 2.3.3) as its optional argument. These options will only influence this YAML document.

The following example  $\text{\LaTeX}$  code showcases the usage of the `\yamlInput` macro:

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

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

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\markdownInput[
 jekyllData,
 expectJekyllData,
 ensureJekyllData,
 smartEllipses,
]{hello.yml}
\end{document}
```

### 2.3.2 Using $\text{\LaTeX}$ hooks with the Markdown package

$\text{\LaTeX}$  provides an intricate hook management system that allows users to insert extra material before and after certain  $\text{\TeX}$  macros and  $\text{\LaTeX}$  environments, among other things. [14, Section 3.1.2]

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

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\AddToHook{cmd/markdownRendererEmphasis/before}{emphasis: }
```

```

\AddToHook{env/markdown/before}{<markdown>}
\AddToHook{env/markdown/after}{</markdown>}
\begin{markdown}
foo _bar_ baz!
\end{markdown}
\end{document}

```

Processing the above example with L<sup>A</sup>T<sub>E</sub>X will produce the text “[markdown](#)foo emphasis: [\\_bar\\_ baz!](#)”, as expected.

However, using hooks to insert extra material after T<sub>E</sub>X commands only works for commands with a fixed number of parameters that don’t use currying.

If, in the above example, you explicitly defined the renderer for emphasis using `\markdownSetup` or another method that does not use currying, then you would be able to insert extra material even after the renderer:

```

\documentclass{article}
\usepackage{markdown}
\markdownSetup{renderers={emphasis={\emph{#1}}}}
\begin{document}
\AddToHook{cmd/markdownRendererEmphasis/before}{<emphasis>}
\AddToHook{cmd/markdownRendererEmphasis/after}{</emphasis>}
\AddToHook{env/markdown/before}{<markdown>}
\AddToHook{env/markdown/after}{</markdown>}
\begin{markdown}
foo _bar_ baz!
\end{markdown}
\end{document}

```

Processing the above example with L<sup>A</sup>T<sub>E</sub>X will produce the text “[markdown](#)foo [emphasis](#) [\\_bar\\_](#) [/emphasis](#) baz!”, as expected.

However, the default renderer for emphasis uses currying and calls the renderer prototype in a way that prevents the use of hooks to insert extra material after the renderer, see Section 2.2.5.12. In such a case, you would need to redefine the renderer in a way that does not use currying before you would be able to use hooks to insert extra material after it.

Hooks also cannot be used to insert extra material after renderers with a variable number of parameters such as the renderer for tables, see Section 2.2.5.39.

### 2.3.3 Options

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

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

The L<sup>A</sup>T<sub>E</sub>X options may be specified when loading the L<sup>A</sup>T<sub>E</sub>X package, when using the `markdown*` L<sup>A</sup>T<sub>E</sub>X environment or the `\markdownInput` macro (see Section 2.3), or via the `\markdownSetup` macro.

### 2.3.3.1 Finalizing and Freezing the Cache

To ensure compatibility with the `minted` package [15, Section 5.1], which supports the `finalizcache` and `frozenscache` package options with similar semantics to the `finalizeCache` and `frozenCache` plain T<sub>E</sub>X options, the Markdown package also recognizes these as aliases and accepts them as document class options. By passing `finalizcache` and `frozenscache` as document class options, you may conveniently control the behavior of both packages at once:

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

We hope that other packages will support the `finalizcache` and `frozenscache` package options in the future, so that they can become a standard interface for preparing L<sup>A</sup>T<sub>E</sub>X document sources for distribution.

```
3895 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
3896 \DeclareOption{frozenscache}{\markdownSetup{frozenCache}}
```

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

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

```
3897 \ExplSyntaxOn
3898 \str_if_eq:VVT
3899 \c_@@_top_layer_tl
3900 \c_@@_option_layer_latex_tl
3901 {
3902 \@@define_option_commands_and_keyvals:
3903 \@@define_renderers:
3904 \@@define_renderer_prototypes:
3905 }
3906 \ExplSyntaxOff
```

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

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

### 2.3.4 Themes

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

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

If the  $\text{\LaTeX}$  option with keys `theme` or `import` is (repeatedly) specified in the `\usepackage` macro, the loading of the theme(s) will be postponed in first-in-first-out order until after the Markdown  $\text{\LaTeX}$  package has been loaded. Otherwise, the theme(s) will be loaded immediately. For example, the following code would first load the Markdown package, then the theme `witiko/example/foo`, and finally the theme `witiko/example/bar`:

```
\usepackage[
 import=witiko/example/foo,
 import=witiko/example/bar,
]{markdown}
```

```
3907 \newif\ifmarkdownLaTeXLoaded
3908 \markdownLaTeXLoadedfalse
```

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

We also define the prop `\g_@@_latex_built_in_themes_prop` that contains the code of built-in themes. This is a packaging optimization, so that built-in themes does not need to be distributed in many small files.



```

3909 \ExplSyntaxOn
3910 \prop_new:N
3911 \g_@@_latex_built_in_themes_prop
3912 \ExplSyntaxOff

```

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

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

```

3913 \AtEndOfPackage{\markdownLaTeXLoadedtrue}

```

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

```

3914 \ExplSyntaxOn
3915 \str_if_eq:VVT
3916 \c_@@_top_layer_tl
3917 \c_@@_option_layer_latex_tl
3918 {
3919 \use:c
3920 { ExplSyntaxOff }
3921 \AtEndOfPackage
3922 {
3923 \@@_if_option:nF
3924 { noDefaults }
3925 {
3926 \@@_if_option:nTF
3927 { experimental }
3928 {
3929 \@@_setup:n
3930 { theme = witiko/markdown/defaults@experimental }
3931 }
3932 {
3933 \@@_setup:n
3934 { theme = witiko/markdown/defaults }
3935 }
3936 }
3937 }
3938 \use:c
3939 { ExplSyntaxOn }
3940 }
3941 \ExplSyntaxOff

```

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

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

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

```
3942 \ExplSyntaxOn
3943 \tl_const:Nn \c_@@_option_layer_context_tl { context }
3944 \cs_generate_variant:Nn
3945 \tl_const:Nn
3946 { NV }
3947 \tl_if_exist:NF
3948 \c_@@_top_layer_tl
3949 {
3950 \tl_const:NV
3951 \c_@@_top_layer_tl
3952 \c_@@_option_layer_context_tl
3953 }
3954 \ExplSyntaxOff
```

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

```
3955 \writestatus{loading}{ConTEXt User Module / markdown}%
3956 \startmodule[markdown]
3957 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
3958 \do\#\do\^\do_do\%do\~}%
3959 \input markdown/markdown
```

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

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

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

### 2.4.1 Typesetting Markdown and YAML

The interface exposes the `\startmarkdown`, `\stopmarkdown`, `\startyaml`, `\stopyaml`, `\inputmarkdown`, and `\inputyaml` macros.

#### 2.4.1.1 Typesetting Markdown and YAML directly

The `\startmarkdown` and `\stopmarkdown` macros are aliases for the macros `\markdownBegin` and `\markdownEnd` exposed by the plain T<sub>E</sub>X interface.

```
3960 \let\startmarkdown\relax
3961 \let\stopmarkdown\relax
```

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

The macros `\startmarkdown` and `\stopmarkdown` are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

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

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

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

```
3962 \let\startyaml\relax
3963 \let\stopyaml\relax
```

You may prepend your own code to the `\startyaml` macro and append your own code to the `\stopyaml` macro to produce special effects before and after the YAML document.

The macros `\startyaml` and `\stopyaml` are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

The following example ConTeXt code showcases the usage of the `\startyaml` and `\stopyaml` macros:

```
\usemodule[t] [markdown]
\starttext
\startyaml
title: _Hello_ **world** ...
author: John Doe
\stopyaml
\stoptext
```

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

```
\usemodule[t] [markdown]
\starttext
\setupyaml[jekyllData, expectJekyllData, ensureJekyllData]
\startyaml
```

```
title: _Hello_ **world** ...
author: John Doe
\stopyaml
\stoptext
```

#### 2.4.1.2 Typesetting Markdown and YAML from external documents

The `\inputmarkdown` macro aliases the macro `\markdownInput` exposed by the plain  $\text{\TeX}$  interface.

```
3964 \let\inputmarkdown\relax
```

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

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

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

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

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

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

```
3965 \let\inputyaml\relax
```

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

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

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

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

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

## 2.4.2 Options

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

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

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

```
3966 \ExplSyntaxOn
3967 \cs_new:Npn
3968 \setupmarkdown
3969 [#1]
3970 {
3971 \@@_setup:n
3972 { #1 }
3973 }
```

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

```
3974 \cs_gset_eq:NN
3975 \setupyaml
3976 \setupmarkdown
```

### 2.4.2.1 Generating Plain T<sub>E</sub>X Option Macros and Key-Values

Unlike plain T<sub>E</sub>X, we also accept caseless variants of options in line with the style of ConT<sub>E</sub>Xt.

```
3977 \cs_new:Nn \@@_caseless:N
3978 {
3979 \regex_replace_all:nnN
3980 { ([a-z])([A-Z]) }
3981 { \1 \c { str_lowercase:n } \cB\{ \2 \cE\} }
3982 #1
3983 \tl_set:Nx
3984 #1
```

```

3985 { #1 }
3986 }
3987 \seq_gput_right:Nn \g_@@_cases_seq { @@_caseless:N }

```

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

```

3988 \str_if_eq:VVT
3989 \c_@@_top_layer_tl
3990 \c_@@_option_layer_context_tl
3991 {
3992 \@@_define_option_commands_and_keyvals:
3993 \@@_define_renderers:
3994 \@@_define_renderer_prototypes:
3995 }

```

### 2.4.3 Themes

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

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

For example, to load a theme named `witiko/tilde` in your document:

```

\usemodule[t][markdown]
\setupmarkdown[import=witiko/tilde]

```

We also define the prop `\g_@@_context_built_in_themes_prop` that contains the code of built-in themes. This is a packaging optimization, so that built-in themes does not need to be distributed in many small files.

```

3996 \prop_new:N
3997 \g_@@_context_built_in_themes_prop
3998 \ExplSyntaxOff

```

Built-in ConT<sub>E</sub>Xt themes provided with the Markdown package include:

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

```
3999 \startmodule[markdownthemewitiko_markdown_defaults]
4000 \unprotect
```

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

## 3 Implementation

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

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

### 3.1 Lua Implementation

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

The Lunamark Lua module implements writers for the conversion to various other formats, such as DocBook, Groff, or HTML. These were stripped from the module and the remaining markdown reader and plain T<sub>E</sub>X writer were hidden behind the converter functions exposed by the Lua interface (see Section 2.1).

```
4001 local upper, format, length =
4002 string.upper, string.format, string.len
4003 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, Cp, any =
4004 lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
4005 lpeg.Cmt, lpeg.Cc, lpeg.Ct, lpeg.B, lpeg.Cs, lpeg.Cp, lpeg.P(1)
```

#### 3.1.1 Utility Functions

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

```
4006 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.

```
4007 function util.err(msg, exit_code)
4008 io.stderr:write("markdown.lua: " .. msg .. "\n")
4009 os.exit(exit_code or 1)
4010 end
```

The `util.cache` method used `dir`, `string`, `salt`, and `suffix` to determine a pathname. If a file with such a pathname does not exist, it gets created with `transform(string)` as its content and the result of `transform(string)` is returned as the second return value in case it's useful to the caller. Regardless, the pathname is always returned as the first return value.

```
4011 function util.cache(dir, string, salt, transform, suffix)
4012 local digest = md5.sumhexa(string .. (salt or ""))
4013 local name = util.pathname(dir, digest .. suffix)
4014 local file = io.open(name, "r")
4015 local result = nil
4016 if file == nil then -- If no cache entry exists, create a new one.
4017 file = assert(io.open(name, "w"),
4018 [[Could not open file]] .. name .. [[for writing]])
4019 result = string
4020 if transform ~= nil then
4021 result = transform(result)
4022 end
4023 assert(file:write(result))
4024 assert(file:close())
4025 end
4026 return name, result
4027 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.

```
4028 function util.cache_verbatim(dir, string)
4029 local name = util.cache(dir, string, nil, nil, ".verbatim")
4030 return name
4031 end
```

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

```
4032 function util.table_copy(t)
4033 local u = { }
4034 for k, v in pairs(t) do u[k] = v end
4035 return setmetatable(u, getmetatable(t))
4036 end
```

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

```
4037 function util.encode_json_string(s)
4038 s = s:gsub([[\\]], [[\\]])
```



```

4039 s = s:gsub([""], [{"\"}])
4040 return [{""] .. s .. [{""]}
4041 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 [16, Chapter 21].

```

4042 function util.expand_tabs_in_line(s, tabstop)
4043 local tab = tabstop or 4
4044 local corr = 0
4045 return (s:gsub(")\t", function(p)
4046 local sp = tab - (p - 1 + corr) % tab
4047 corr = corr - 1 + sp
4048 return string.rep(" ", sp)
4049 end))
4050 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.

```

4051 function util.walk(t, f)
4052 local typ = type(t)
4053 if typ == "string" then
4054 f(t)
4055 elseif typ == "table" then
4056 local i = 1
4057 local n
4058 n = t[i]
4059 while n do
4060 util.walk(n, f)
4061 i = i + 1
4062 n = t[i]
4063 end
4064 elseif typ == "function" then
4065 local ok, val = pcall(t)
4066 if ok then
4067 util.walk(val, f)
4068 end
4069 else
4070 f(tostring(t))
4071 end
4072 end

```

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

```

4073 function util.flatten(ary)
4074 local new = {}
4075 for _,v in ipairs(ary) do
4076 if type(v) == "table" then
4077 for _,w in ipairs(util.flatten(v)) do
4078 new[#new + 1] = w
4079 end
4080 else
4081 new[#new + 1] = v
4082 end
4083 end
4084 return new
4085 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.

```

4086 function util.rope_to_string(rope)
4087 local buffer = {}
4088 util.walk(rope, function(x) buffer[#buffer + 1] = x end)
4089 return table.concat(buffer)
4090 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.

```

4091 function util.rope_last(rope)
4092 if #rope == 0 then
4093 return nil
4094 else
4095 local l = rope[#rope]
4096 if type(l) == "table" then
4097 return util.rope_last(l)
4098 else
4099 return l
4100 end
4101 end
4102 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  $1 \leq i \leq \#ary$ .

```

4103 function util.intersperse(ary, x)
4104 local new = {}
4105 local l = #ary
4106 for i,v in ipairs(ary) do
4107 local n = #new
4108 new[n + 1] = v
4109 if i ~= l then
4110 new[n + 2] = x

```

```

4111 end
4112 end
4113 return new
4114 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  $1 \leq i \leq \#ary$ .

```

4115 function util.map(ary, f)
4116 local new = {}
4117 for i,v in ipairs(ary) do
4118 new[i] = f(v)
4119 end
4120 return new
4121 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.

```

4122 function util.escaper(char_escapes, string_escapes)

```

Build a string of escapable characters.

```

4123 local char_escapes_list = ""
4124 for i,_ in pairs(char_escapes) do
4125 char_escapes_list = char_escapes_list .. i
4126 end

```

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

```

4127 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.

```

4128 if string_escapes then
4129 for k,v in pairs(string_escapes) do
4130 escapable = P(k) / v + escapable
4131 end
4132 end

```

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

```
4133 local escape_string = Cs((escapable + any)^0)
```

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

```
4134 return function(s)
4135 return lpeg.match(escape_string, s)
4136 end
4137 end
```

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

```
4138 function util.pathname(dir, file)
4139 if #dir == 0 then
4140 return file
4141 else
4142 return dir .. "/" .. file
4143 end
4144 end
```

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

```
4145 function util.salt(options)
4146 local opt_string = {}
4147 for k, _ in pairs(defaultOptions) do
4148 local v = options[k]
4149 if type(v) == "table" then
4150 for _, i in ipairs(v) do
4151 opt_string[#opt_string+1] = k .. "=" .. tostring(i)
4152 end
4153 end
4154 end
```

The `cacheDir` option is disregarded.

```
4153 elseif k ~= "cacheDir" then
4154 opt_string[#opt_string+1] = k .. "=" .. tostring(v)
4155 end
4156 end
4157 table.sort(opt_string)
4158 local salt = table.concat(opt_string, ",")
4159 .. "," .. metadata.version
4160 return salt
4161 end
```

The `util.warning` method produces a warning `s` that is unrelated to any specific markdown text being processed. For warnings that are specific to a markdown text, use `writer->warning` function.

```
4162 function util.warning(s)
4163 io.stderr:write("Warning: " .. s .. "\n")
4164 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.

```
4165 local entities = {}
4166
4167 local character_entities = {
4168 ["Tab"] = 9,
4169 ["NewLine"] = 10,
4170 ["excl"] = 33,
4171 ["QUOT"] = 34,
4172 ["quot"] = 34,
4173 ["num"] = 35,
4174 ["dollar"] = 36,
4175 ["percnt"] = 37,
4176 ["AMP"] = 38,
4177 ["amp"] = 38,
4178 ["apos"] = 39,
4179 ["lpar"] = 40,
4180 ["rpar"] = 41,
4181 ["ast"] = 42,
4182 ["midast"] = 42,
4183 ["plus"] = 43,
4184 ["comma"] = 44,
4185 ["period"] = 46,
4186 ["sol"] = 47,
4187 ["colon"] = 58,
4188 ["semi"] = 59,
4189 ["LT"] = 60,
4190 ["lt"] = 60,
4191 ["nvlt"] = {60, 8402},
4192 ["bne"] = {61, 8421},
4193 ["equals"] = 61,
4194 ["GT"] = 62,
4195 ["gt"] = 62,
4196 ["nvgt"] = {62, 8402},
4197 ["quest"] = 63,
4198 ["commat"] = 64,
4199 ["lbrack"] = 91,
4200 ["lsqb"] = 91,
4201 ["bsol"] = 92,
4202 ["rbrack"] = 93,
4203 ["rsqb"] = 93,
4204 ["Hat"] = 94,
4205 ["UnderBar"] = 95,
4206 ["lowbar"] = 95,
```

```

4207 ["DiacriticalGrave"] = 96,
4208 ["grave"] = 96,
4209 ["fjlig"] = {102, 106},
4210 ["lbrace"] = 123,
4211 ["lcub"] = 123,
4212 ["VerticalLine"] = 124,
4213 ["verbar"] = 124,
4214 ["vert"] = 124,
4215 ["rbrace"] = 125,
4216 ["rcub"] = 125,
4217 ["NonBreakingSpace"] = 160,
4218 ["nbsp"] = 160,
4219 ["iexcl"] = 161,
4220 ["cent"] = 162,
4221 ["pound"] = 163,
4222 ["curren"] = 164,
4223 ["yen"] = 165,
4224 ["brvbar"] = 166,
4225 ["sect"] = 167,
4226 ["Dot"] = 168,
4227 ["DoubleDot"] = 168,
4228 ["die"] = 168,
4229 ["uml"] = 168,
4230 ["COPY"] = 169,
4231 ["copy"] = 169,
4232 ["ordf"] = 170,
4233 ["laquo"] = 171,
4234 ["not"] = 172,
4235 ["shy"] = 173,
4236 ["REG"] = 174,
4237 ["circledR"] = 174,
4238 ["reg"] = 174,
4239 ["macr"] = 175,
4240 ["strns"] = 175,
4241 ["deg"] = 176,
4242 ["PlusMinus"] = 177,
4243 ["plusmn"] = 177,
4244 ["pm"] = 177,
4245 ["sup2"] = 178,
4246 ["sup3"] = 179,
4247 ["DiacriticalAcute"] = 180,
4248 ["acute"] = 180,
4249 ["micro"] = 181,
4250 ["para"] = 182,
4251 ["CenterDot"] = 183,
4252 ["centerdot"] = 183,
4253 ["middot"] = 183,

```

4254 ["Cedilla"] = 184,  
 4255 ["cedil"] = 184,  
 4256 ["sup1"] = 185,  
 4257 ["ordm"] = 186,  
 4258 ["raquo"] = 187,  
 4259 ["frac14"] = 188,  
 4260 ["frac12"] = 189,  
 4261 ["half"] = 189,  
 4262 ["frac34"] = 190,  
 4263 ["iquest"] = 191,  
 4264 ["Agrave"] = 192,  
 4265 ["Aacute"] = 193,  
 4266 ["Acirc"] = 194,  
 4267 ["Atilde"] = 195,  
 4268 ["Auml"] = 196,  
 4269 ["Aring"] = 197,  
 4270 ["angst"] = 197,  
 4271 ["AElig"] = 198,  
 4272 ["Ccedil"] = 199,  
 4273 ["Egrave"] = 200,  
 4274 ["Eacute"] = 201,  
 4275 ["Ecirc"] = 202,  
 4276 ["Euml"] = 203,  
 4277 ["Igrave"] = 204,  
 4278 ["Iacute"] = 205,  
 4279 ["Icirc"] = 206,  
 4280 ["Iuml"] = 207,  
 4281 ["ETH"] = 208,  
 4282 ["Ntilde"] = 209,  
 4283 ["Ograve"] = 210,  
 4284 ["Oacute"] = 211,  
 4285 ["Ocirc"] = 212,  
 4286 ["Otilde"] = 213,  
 4287 ["Ouml"] = 214,  
 4288 ["times"] = 215,  
 4289 ["Oslash"] = 216,  
 4290 ["Ugrave"] = 217,  
 4291 ["Uacute"] = 218,  
 4292 ["Ucirc"] = 219,  
 4293 ["Uuml"] = 220,  
 4294 ["Yacute"] = 221,  
 4295 ["THORN"] = 222,  
 4296 ["szlig"] = 223,  
 4297 ["agrave"] = 224,  
 4298 ["aacute"] = 225,  
 4299 ["acirc"] = 226,  
 4300 ["atilde"] = 227,

```

4301 ["auml"] = 228,
4302 ["aring"] = 229,
4303 ["aelig"] = 230,
4304 ["ccedil"] = 231,
4305 ["egrave"] = 232,
4306 ["eacute"] = 233,
4307 ["ecirc"] = 234,
4308 ["euml"] = 235,
4309 ["igrave"] = 236,
4310 ["iacute"] = 237,
4311 ["icirc"] = 238,
4312 ["iuml"] = 239,
4313 ["eth"] = 240,
4314 ["ntilde"] = 241,
4315 ["ograve"] = 242,
4316 ["oacute"] = 243,
4317 ["ocirc"] = 244,
4318 ["otilde"] = 245,
4319 ["ouml"] = 246,
4320 ["div"] = 247,
4321 ["divide"] = 247,
4322 ["oslash"] = 248,
4323 ["ugrave"] = 249,
4324 ["uacute"] = 250,
4325 ["ucirc"] = 251,
4326 ["uuml"] = 252,
4327 ["yacute"] = 253,
4328 ["thorn"] = 254,
4329 ["yuml"] = 255,
4330 ["Amacr"] = 256,
4331 ["amacr"] = 257,
4332 ["Abreve"] = 258,
4333 ["abreve"] = 259,
4334 ["Aogon"] = 260,
4335 ["aogon"] = 261,
4336 ["Cacute"] = 262,
4337 ["cacute"] = 263,
4338 ["Ccirc"] = 264,
4339 ["ccirc"] = 265,
4340 ["Cdot"] = 266,
4341 ["cdot"] = 267,
4342 ["Ccaron"] = 268,
4343 ["ccaron"] = 269,
4344 ["Dcaron"] = 270,
4345 ["dcaron"] = 271,
4346 ["Dstrok"] = 272,
4347 ["dstrok"] = 273,

```



```

4348 ["Emacr"] = 274,
4349 ["emacr"] = 275,
4350 ["Edot"] = 278,
4351 ["edot"] = 279,
4352 ["Eogon"] = 280,
4353 ["eogon"] = 281,
4354 ["Ecaron"] = 282,
4355 ["ecaron"] = 283,
4356 ["Gcirc"] = 284,
4357 ["gcirc"] = 285,
4358 ["Gbreve"] = 286,
4359 ["gbreve"] = 287,
4360 ["Gdot"] = 288,
4361 ["gdot"] = 289,
4362 ["Gcedil"] = 290,
4363 ["Hcirc"] = 292,
4364 ["hcirc"] = 293,
4365 ["Hstrokr"] = 294,
4366 ["hstrokr"] = 295,
4367 ["Itilde"] = 296,
4368 ["itilde"] = 297,
4369 ["Imacr"] = 298,
4370 ["imacr"] = 299,
4371 ["Iogon"] = 302,
4372 ["iogon"] = 303,
4373 ["Idot"] = 304,
4374 ["imath"] = 305,
4375 ["inodot"] = 305,
4376 ["IJlig"] = 306,
4377 ["ijlig"] = 307,
4378 ["Jcirc"] = 308,
4379 ["jcirc"] = 309,
4380 ["Kcedil"] = 310,
4381 ["kcedil"] = 311,
4382 ["kgreen"] = 312,
4383 ["Lacute"] = 313,
4384 ["lacute"] = 314,
4385 ["Lcedil"] = 315,
4386 ["lcedil"] = 316,
4387 ["Lcaron"] = 317,
4388 ["lcaron"] = 318,
4389 ["Lmidot"] = 319,
4390 ["lmidot"] = 320,
4391 ["Lstrokr"] = 321,
4392 ["lstrokr"] = 322,
4393 ["Nacute"] = 323,
4394 ["nacute"] = 324,

```

4395 ["Ncedil"] = 325,  
 4396 ["ncedil"] = 326,  
 4397 ["Ncaron"] = 327,  
 4398 ["ncaron"] = 328,  
 4399 ["napos"] = 329,  
 4400 ["ENG"] = 330,  
 4401 ["eng"] = 331,  
 4402 ["Omacr"] = 332,  
 4403 ["omacr"] = 333,  
 4404 ["Odblac"] = 336,  
 4405 ["odblac"] = 337,  
 4406 ["OElig"] = 338,  
 4407 ["oelig"] = 339,  
 4408 ["Racute"] = 340,  
 4409 ["racute"] = 341,  
 4410 ["Rcedil"] = 342,  
 4411 ["rcedil"] = 343,  
 4412 ["Rcaron"] = 344,  
 4413 ["rcaron"] = 345,  
 4414 ["Sacute"] = 346,  
 4415 ["sacute"] = 347,  
 4416 ["Scirc"] = 348,  
 4417 ["scirc"] = 349,  
 4418 ["Scedil"] = 350,  
 4419 ["scedil"] = 351,  
 4420 ["Scaron"] = 352,  
 4421 ["scaron"] = 353,  
 4422 ["Tcedil"] = 354,  
 4423 ["tcedil"] = 355,  
 4424 ["Tcaron"] = 356,  
 4425 ["tcaron"] = 357,  
 4426 ["Tstrok"] = 358,  
 4427 ["tstrok"] = 359,  
 4428 ["Utilde"] = 360,  
 4429 ["utilde"] = 361,  
 4430 ["Umacr"] = 362,  
 4431 ["umacr"] = 363,  
 4432 ["Ubreve"] = 364,  
 4433 ["ubreve"] = 365,  
 4434 ["Uring"] = 366,  
 4435 ["uring"] = 367,  
 4436 ["Udblac"] = 368,  
 4437 ["udblac"] = 369,  
 4438 ["Uogon"] = 370,  
 4439 ["uogon"] = 371,  
 4440 ["Wcirc"] = 372,  
 4441 ["wcirc"] = 373,

```

4442 ["Ycirc"] = 374,
4443 ["ycirc"] = 375,
4444 ["Yuml"] = 376,
4445 ["Zacute"] = 377,
4446 ["zacute"] = 378,
4447 ["Zdot"] = 379,
4448 ["zdot"] = 380,
4449 ["Zcaron"] = 381,
4450 ["zcaron"] = 382,
4451 ["fnof"] = 402,
4452 ["imped"] = 437,
4453 ["gacute"] = 501,
4454 ["jmath"] = 567,
4455 ["circ"] = 710,
4456 ["Hacek"] = 711,
4457 ["caron"] = 711,
4458 ["Breve"] = 728,
4459 ["breve"] = 728,
4460 ["DiacriticalDot"] = 729,
4461 ["dot"] = 729,
4462 ["ring"] = 730,
4463 ["ogon"] = 731,
4464 ["DiacriticalTilde"] = 732,
4465 ["tilde"] = 732,
4466 ["DiacriticalDoubleAcute"] = 733,
4467 ["dblac"] = 733,
4468 ["DownBreve"] = 785,
4469 ["Alpha"] = 913,
4470 ["Beta"] = 914,
4471 ["Gamma"] = 915,
4472 ["Delta"] = 916,
4473 ["Epsilon"] = 917,
4474 ["Zeta"] = 918,
4475 ["Eta"] = 919,
4476 ["Theta"] = 920,
4477 ["Iota"] = 921,
4478 ["Kappa"] = 922,
4479 ["Lambda"] = 923,
4480 ["Mu"] = 924,
4481 ["Nu"] = 925,
4482 ["Xi"] = 926,
4483 ["Omicron"] = 927,
4484 ["Pi"] = 928,
4485 ["Rho"] = 929,
4486 ["Sigma"] = 931,
4487 ["Tau"] = 932,
4488 ["Upsilon"] = 933,

```

```

4489 ["Phi"] = 934,
4490 ["Chi"] = 935,
4491 ["Psi"] = 936,
4492 ["Omega"] = 937,
4493 ["ohm"] = 937,
4494 ["alpha"] = 945,
4495 ["beta"] = 946,
4496 ["gamma"] = 947,
4497 ["delta"] = 948,
4498 ["epsi"] = 949,
4499 ["epsilon"] = 949,
4500 ["zeta"] = 950,
4501 ["eta"] = 951,
4502 ["theta"] = 952,
4503 ["iota"] = 953,
4504 ["kappa"] = 954,
4505 ["lambda"] = 955,
4506 ["mu"] = 956,
4507 ["nu"] = 957,
4508 ["xi"] = 958,
4509 ["omicron"] = 959,
4510 ["pi"] = 960,
4511 ["rho"] = 961,
4512 ["sigmaf"] = 962,
4513 ["sigmav"] = 962,
4514 ["varsigma"] = 962,
4515 ["sigma"] = 963,
4516 ["tau"] = 964,
4517 ["upsi"] = 965,
4518 ["upsilon"] = 965,
4519 ["phi"] = 966,
4520 ["chi"] = 967,
4521 ["psi"] = 968,
4522 ["omega"] = 969,
4523 ["thetasym"] = 977,
4524 ["thetav"] = 977,
4525 ["vartheta"] = 977,
4526 ["Upsi"] = 978,
4527 ["upsih"] = 978,
4528 ["phiv"] = 981,
4529 ["straightphi"] = 981,
4530 ["varphi"] = 981,
4531 ["piv"] = 982,
4532 ["varpi"] = 982,
4533 ["Gammad"] = 988,
4534 ["digamma"] = 989,
4535 ["gammad"] = 989,

```

```

4536 ["kappav"] = 1008,
4537 ["varkappa"] = 1008,
4538 ["rhov"] = 1009,
4539 ["varrho"] = 1009,
4540 ["epsiv"] = 1013,
4541 ["straightepsilon"] = 1013,
4542 ["varepsilon"] = 1013,
4543 ["backepsilon"] = 1014,
4544 ["bepsi"] = 1014,
4545 ["IOcy"] = 1025,
4546 ["DJcy"] = 1026,
4547 ["GJcy"] = 1027,
4548 ["Jukcy"] = 1028,
4549 ["DScy"] = 1029,
4550 ["Iukcy"] = 1030,
4551 ["YIcy"] = 1031,
4552 ["Jsercy"] = 1032,
4553 ["LJcy"] = 1033,
4554 ["NJcy"] = 1034,
4555 ["TSHcy"] = 1035,
4556 ["KJcy"] = 1036,
4557 ["Ubrcy"] = 1038,
4558 ["DZcy"] = 1039,
4559 ["Acy"] = 1040,
4560 ["Bcy"] = 1041,
4561 ["Vcy"] = 1042,
4562 ["Gcy"] = 1043,
4563 ["Dcy"] = 1044,
4564 ["IEcy"] = 1045,
4565 ["ZHcy"] = 1046,
4566 ["Zcy"] = 1047,
4567 ["Icy"] = 1048,
4568 ["Jcy"] = 1049,
4569 ["Kcy"] = 1050,
4570 ["Lcy"] = 1051,
4571 ["Mcy"] = 1052,
4572 ["Ncy"] = 1053,
4573 ["Ocy"] = 1054,
4574 ["Pcy"] = 1055,
4575 ["Rcy"] = 1056,
4576 ["Scy"] = 1057,
4577 ["Tcy"] = 1058,
4578 ["Ucy"] = 1059,
4579 ["Fcy"] = 1060,
4580 ["KHcy"] = 1061,
4581 ["TScy"] = 1062,
4582 ["CHcy"] = 1063,

```

```

4583 ["SHcy"] = 1064,
4584 ["SHCHcy"] = 1065,
4585 ["HARDcy"] = 1066,
4586 ["Ycy"] = 1067,
4587 ["SOFTcy"] = 1068,
4588 ["Ecy"] = 1069,
4589 ["YUcy"] = 1070,
4590 ["YAcy"] = 1071,
4591 ["acy"] = 1072,
4592 ["bcy"] = 1073,
4593 ["vcy"] = 1074,
4594 ["gcy"] = 1075,
4595 ["dcy"] = 1076,
4596 ["iecy"] = 1077,
4597 ["zhcy"] = 1078,
4598 ["zcy"] = 1079,
4599 ["icy"] = 1080,
4600 ["jcy"] = 1081,
4601 ["kcy"] = 1082,
4602 ["lcy"] = 1083,
4603 ["mcy"] = 1084,
4604 ["ncy"] = 1085,
4605 ["ocy"] = 1086,
4606 ["pcy"] = 1087,
4607 ["rcy"] = 1088,
4608 ["scy"] = 1089,
4609 ["tcy"] = 1090,
4610 ["ucy"] = 1091,
4611 ["fcy"] = 1092,
4612 ["khcy"] = 1093,
4613 ["tscy"] = 1094,
4614 ["chcy"] = 1095,
4615 ["shcy"] = 1096,
4616 ["shchcy"] = 1097,
4617 ["hardcy"] = 1098,
4618 ["ycy"] = 1099,
4619 ["softcy"] = 1100,
4620 ["ecy"] = 1101,
4621 ["yucy"] = 1102,
4622 ["yacy"] = 1103,
4623 ["iocy"] = 1105,
4624 ["djcy"] = 1106,
4625 ["gjcy"] = 1107,
4626 ["jukcy"] = 1108,
4627 ["dscy"] = 1109,
4628 ["iukcy"] = 1110,
4629 ["yicy"] = 1111,

```

```

4630 ["jsercy"] = 1112,
4631 ["ljcy"] = 1113,
4632 ["njcy"] = 1114,
4633 ["tshcy"] = 1115,
4634 ["kjcy"] = 1116,
4635 ["ubrcy"] = 1118,
4636 ["dzcy"] = 1119,
4637 ["ensp"] = 8194,
4638 ["emsp"] = 8195,
4639 ["emsp13"] = 8196,
4640 ["emsp14"] = 8197,
4641 ["numsp"] = 8199,
4642 ["puncsp"] = 8200,
4643 ["ThinSpace"] = 8201,
4644 ["thinsp"] = 8201,
4645 ["VeryThinSpace"] = 8202,
4646 ["hairsp"] = 8202,
4647 ["NegativeMediumSpace"] = 8203,
4648 ["NegativeThickSpace"] = 8203,
4649 ["NegativeThinSpace"] = 8203,
4650 ["NegativeVeryThinSpace"] = 8203,
4651 ["ZeroWidthSpace"] = 8203,
4652 ["zwnj"] = 8204,
4653 ["zwj"] = 8205,
4654 ["lrm"] = 8206,
4655 ["rlm"] = 8207,
4656 ["dash"] = 8208,
4657 ["hyphen"] = 8208,
4658 ["ndash"] = 8211,
4659 ["mdash"] = 8212,
4660 ["horbar"] = 8213,
4661 ["Verbar"] = 8214,
4662 ["Vert"] = 8214,
4663 ["OpenCurlyQuote"] = 8216,
4664 ["lsquo"] = 8216,
4665 ["CloseCurlyQuote"] = 8217,
4666 ["rsquo"] = 8217,
4667 ["rsquor"] = 8217,
4668 ["lsquor"] = 8218,
4669 ["sbquo"] = 8218,
4670 ["OpenCurlyDoubleQuote"] = 8220,
4671 ["ldquo"] = 8220,
4672 ["CloseCurlyDoubleQuote"] = 8221,
4673 ["rdquo"] = 8221,
4674 ["rdquor"] = 8221,
4675 ["bdquo"] = 8222,
4676 ["ldquor"] = 8222,

```

```

4677 ["dagger"] = 8224,
4678 ["Dagger"] = 8225,
4679 ["ddagger"] = 8225,
4680 ["bull"] = 8226,
4681 ["bullet"] = 8226,
4682 ["nldr"] = 8229,
4683 ["hellip"] = 8230,
4684 ["mldr"] = 8230,
4685 ["permil"] = 8240,
4686 ["pertenk"] = 8241,
4687 ["prime"] = 8242,
4688 ["Prime"] = 8243,
4689 ["tprime"] = 8244,
4690 ["backprime"] = 8245,
4691 ["bprime"] = 8245,
4692 ["lsaquo"] = 8249,
4693 ["rsaquo"] = 8250,
4694 ["OverBar"] = 8254,
4695 ["oline"] = 8254,
4696 ["caret"] = 8257,
4697 ["hybull"] = 8259,
4698 ["frasl"] = 8260,
4699 ["bsemi"] = 8271,
4700 ["qprime"] = 8279,
4701 ["MediumSpace"] = 8287,
4702 ["ThickSpace"] = {8287, 8202},
4703 ["NoBreak"] = 8288,
4704 ["ApplyFunction"] = 8289,
4705 ["af"] = 8289,
4706 ["InvisibleTimes"] = 8290,
4707 ["it"] = 8290,
4708 ["InvisibleComma"] = 8291,
4709 ["ic"] = 8291,
4710 ["euro"] = 8364,
4711 ["TripleDot"] = 8411,
4712 ["tdot"] = 8411,
4713 ["DotDot"] = 8412,
4714 ["Copf"] = 8450,
4715 ["complexes"] = 8450,
4716 ["incare"] = 8453,
4717 ["gscr"] = 8458,
4718 ["HilbertSpace"] = 8459,
4719 ["Hscr"] = 8459,
4720 ["hamilt"] = 8459,
4721 ["Hfr"] = 8460,
4722 ["Poincareplane"] = 8460,
4723 ["Hopf"] = 8461,

```



```

4724 ["quaternions"] = 8461,
4725 ["planckh"] = 8462,
4726 ["hbar"] = 8463,
4727 ["hslash"] = 8463,
4728 ["planck"] = 8463,
4729 ["plankv"] = 8463,
4730 ["Iscr"] = 8464,
4731 ["imagline"] = 8464,
4732 ["Ifr"] = 8465,
4733 ["Im"] = 8465,
4734 ["image"] = 8465,
4735 ["imagpart"] = 8465,
4736 ["Laplacetrfr"] = 8466,
4737 ["Lscr"] = 8466,
4738 ["lagran"] = 8466,
4739 ["ell"] = 8467,
4740 ["Nopf"] = 8469,
4741 ["naturals"] = 8469,
4742 ["numero"] = 8470,
4743 ["copysr"] = 8471,
4744 ["weierp"] = 8472,
4745 ["wp"] = 8472,
4746 ["Popf"] = 8473,
4747 ["primes"] = 8473,
4748 ["Qopf"] = 8474,
4749 ["rationals"] = 8474,
4750 ["Rscr"] = 8475,
4751 ["realine"] = 8475,
4752 ["Re"] = 8476,
4753 ["Rfr"] = 8476,
4754 ["real"] = 8476,
4755 ["realpart"] = 8476,
4756 ["Ropf"] = 8477,
4757 ["reals"] = 8477,
4758 ["rx"] = 8478,
4759 ["TRADE"] = 8482,
4760 ["trade"] = 8482,
4761 ["Zopf"] = 8484,
4762 ["integers"] = 8484,
4763 ["mho"] = 8487,
4764 ["Zfr"] = 8488,
4765 ["zeetrf"] = 8488,
4766 ["iiota"] = 8489,
4767 ["Bernoullis"] = 8492,
4768 ["Bscr"] = 8492,
4769 ["bernou"] = 8492,
4770 ["Cayleys"] = 8493,

```

```

4771 ["Cfr"] = 8493,
4772 ["escr"] = 8495,
4773 ["Escr"] = 8496,
4774 ["expectation"] = 8496,
4775 ["Fouriertrf"] = 8497,
4776 ["Fscr"] = 8497,
4777 ["Mellintrf"] = 8499,
4778 ["Mscr"] = 8499,
4779 ["phmmat"] = 8499,
4780 ["order"] = 8500,
4781 ["orderof"] = 8500,
4782 ["oscr"] = 8500,
4783 ["alefsym"] = 8501,
4784 ["aleph"] = 8501,
4785 ["beth"] = 8502,
4786 ["gimel"] = 8503,
4787 ["daleth"] = 8504,
4788 ["CapitalDifferentialD"] = 8517,
4789 ["DD"] = 8517,
4790 ["DifferentialD"] = 8518,
4791 ["dd"] = 8518,
4792 ["ExponentialE"] = 8519,
4793 ["ee"] = 8519,
4794 ["exponentiale"] = 8519,
4795 ["ImaginaryI"] = 8520,
4796 ["ii"] = 8520,
4797 ["frac13"] = 8531,
4798 ["frac23"] = 8532,
4799 ["frac15"] = 8533,
4800 ["frac25"] = 8534,
4801 ["frac35"] = 8535,
4802 ["frac45"] = 8536,
4803 ["frac16"] = 8537,
4804 ["frac56"] = 8538,
4805 ["frac18"] = 8539,
4806 ["frac38"] = 8540,
4807 ["frac58"] = 8541,
4808 ["frac78"] = 8542,
4809 ["LeftArrow"] = 8592,
4810 ["ShortLeftArrow"] = 8592,
4811 ["larr"] = 8592,
4812 ["leftarrow"] = 8592,
4813 ["slarr"] = 8592,
4814 ["ShortUpArrow"] = 8593,
4815 ["UpArrow"] = 8593,
4816 ["uarr"] = 8593,
4817 ["uparrow"] = 8593,

```

```

4818 ["RightArrow"] = 8594,
4819 ["ShortRightArrow"] = 8594,
4820 ["rarr"] = 8594,
4821 ["rightarrow"] = 8594,
4822 ["srarr"] = 8594,
4823 ["DownArrow"] = 8595,
4824 ["ShortDownArrow"] = 8595,
4825 ["darr"] = 8595,
4826 ["downarrow"] = 8595,
4827 ["LeftRightArrow"] = 8596,
4828 ["harr"] = 8596,
4829 ["leftrightarrow"] = 8596,
4830 ["UpDownArrow"] = 8597,
4831 ["updownarrow"] = 8597,
4832 ["varr"] = 8597,
4833 ["UpperLeftArrow"] = 8598,
4834 ["nwarr"] = 8598,
4835 ["nwarrow"] = 8598,
4836 ["UpperRightArrow"] = 8599,
4837 ["nearr"] = 8599,
4838 ["nearrow"] = 8599,
4839 ["LowerRightArrow"] = 8600,
4840 ["searr"] = 8600,
4841 ["searrow"] = 8600,
4842 ["LowerLeftArrow"] = 8601,
4843 ["swarr"] = 8601,
4844 ["swarrow"] = 8601,
4845 ["nlarr"] = 8602,
4846 ["nleftarrow"] = 8602,
4847 ["nrarr"] = 8603,
4848 ["nrightarrow"] = 8603,
4849 ["nrarrw"] = {8605, 824},
4850 ["rarrw"] = 8605,
4851 ["rightsquigarrow"] = 8605,
4852 ["Larr"] = 8606,
4853 ["twoheadleftarrow"] = 8606,
4854 ["Uarr"] = 8607,
4855 ["Rarr"] = 8608,
4856 ["twoheadrightarrow"] = 8608,
4857 ["Darr"] = 8609,
4858 ["larrtl"] = 8610,
4859 ["leftarrowtail"] = 8610,
4860 ["rarrtl"] = 8611,
4861 ["rightarrowtail"] = 8611,
4862 ["LeftTeeArrow"] = 8612,
4863 ["mapstoleft"] = 8612,
4864 ["UpTeeArrow"] = 8613,

```

```

4865 ["mapstoup"] = 8613,
4866 ["RightTeeArrow"] = 8614,
4867 ["map"] = 8614,
4868 ["mapsto"] = 8614,
4869 ["DownTeeArrow"] = 8615,
4870 ["mapstodown"] = 8615,
4871 ["hookleftarrow"] = 8617,
4872 ["larrhk"] = 8617,
4873 ["hookrightarrow"] = 8618,
4874 ["rarrhk"] = 8618,
4875 ["larrlp"] = 8619,
4876 ["looparrowleft"] = 8619,
4877 ["looparrowright"] = 8620,
4878 ["rarrlp"] = 8620,
4879 ["harrw"] = 8621,
4880 ["leftrightsquigarrow"] = 8621,
4881 ["nharr"] = 8622,
4882 ["nleftrightarrow"] = 8622,
4883 ["Lsh"] = 8624,
4884 ["lsh"] = 8624,
4885 ["Rsh"] = 8625,
4886 ["rsh"] = 8625,
4887 ["ldsh"] = 8626,
4888 ["rdsh"] = 8627,
4889 ["crarr"] = 8629,
4890 ["cularr"] = 8630,
4891 ["curvearrowleft"] = 8630,
4892 ["curarr"] = 8631,
4893 ["curvearrowright"] = 8631,
4894 ["circlearrowleft"] = 8634,
4895 ["olarr"] = 8634,
4896 ["circlearrowright"] = 8635,
4897 ["orarr"] = 8635,
4898 ["LeftVector"] = 8636,
4899 ["leftharpoonup"] = 8636,
4900 ["lharu"] = 8636,
4901 ["DownLeftVector"] = 8637,
4902 ["leftharpoondown"] = 8637,
4903 ["lhard"] = 8637,
4904 ["RightUpVector"] = 8638,
4905 ["uharr"] = 8638,
4906 ["upharpoonright"] = 8638,
4907 ["LeftUpVector"] = 8639,
4908 ["uharl"] = 8639,
4909 ["upharpoonleft"] = 8639,
4910 ["RightVector"] = 8640,
4911 ["rharu"] = 8640,

```

```

4912 ["rightharpoonup"] = 8640,
4913 ["DownRightVector"] = 8641,
4914 ["rhard"] = 8641,
4915 ["rightharpoondown"] = 8641,
4916 ["RightDownVector"] = 8642,
4917 ["dharr"] = 8642,
4918 ["downharpoonright"] = 8642,
4919 ["LeftDownVector"] = 8643,
4920 ["dharl"] = 8643,
4921 ["downharpoonleft"] = 8643,
4922 ["RightArrowLeftArrow"] = 8644,
4923 ["rightleftarrows"] = 8644,
4924 ["rlarr"] = 8644,
4925 ["UpArrowDownArrow"] = 8645,
4926 ["udarr"] = 8645,
4927 ["LeftArrowRightArrow"] = 8646,
4928 ["leftrightarrows"] = 8646,
4929 ["lrarr"] = 8646,
4930 ["leftleftarrows"] = 8647,
4931 ["llarr"] = 8647,
4932 ["upuparrows"] = 8648,
4933 ["uuarr"] = 8648,
4934 ["rightrightarrows"] = 8649,
4935 ["rrarr"] = 8649,
4936 ["ddarr"] = 8650,
4937 ["downdownarrows"] = 8650,
4938 ["ReverseEquilibrium"] = 8651,
4939 ["leftrightharpoons"] = 8651,
4940 ["lrhar"] = 8651,
4941 ["Equilibrium"] = 8652,
4942 ["rightleftharpoons"] = 8652,
4943 ["rlhar"] = 8652,
4944 ["nLeftarrow"] = 8653,
4945 ["nLArr"] = 8653,
4946 ["nLeftrightarrow"] = 8654,
4947 ["nhArr"] = 8654,
4948 ["nRightarrow"] = 8655,
4949 ["nrArr"] = 8655,
4950 ["DoubleLeftArrow"] = 8656,
4951 ["Leftarrow"] = 8656,
4952 ["lArr"] = 8656,
4953 ["DoubleUpArrow"] = 8657,
4954 ["Uparrow"] = 8657,
4955 ["uArr"] = 8657,
4956 ["DoubleRightArrow"] = 8658,
4957 ["Implies"] = 8658,
4958 ["Rightarrow"] = 8658,

```

```

4959 ["rArr"] = 8658,
4960 ["DoubleDownArrow"] = 8659,
4961 ["Downarrow"] = 8659,
4962 ["dArr"] = 8659,
4963 ["DoubleLeftRightArrow"] = 8660,
4964 ["Leftrightarrow"] = 8660,
4965 ["hArr"] = 8660,
4966 ["iff"] = 8660,
4967 ["DoubleUpDownArrow"] = 8661,
4968 ["Updownarrow"] = 8661,
4969 ["vArr"] = 8661,
4970 ["nwArr"] = 8662,
4971 ["neArr"] = 8663,
4972 ["seArr"] = 8664,
4973 ["swArr"] = 8665,
4974 ["Lleftarrow"] = 8666,
4975 ["lAarr"] = 8666,
4976 ["Rrightarrow"] = 8667,
4977 ["rAarr"] = 8667,
4978 ["zigrarr"] = 8669,
4979 ["LeftArrowBar"] = 8676,
4980 ["larrb"] = 8676,
4981 ["RightArrowBar"] = 8677,
4982 ["rarrb"] = 8677,
4983 ["DownArrowUpArrow"] = 8693,
4984 ["duarr"] = 8693,
4985 ["loarr"] = 8701,
4986 ["roarr"] = 8702,
4987 ["hoarr"] = 8703,
4988 ["ForAll"] = 8704,
4989 ["forall"] = 8704,
4990 ["comp"] = 8705,
4991 ["complement"] = 8705,
4992 ["PartialD"] = 8706,
4993 ["npart"] = {8706, 824},
4994 ["part"] = 8706,
4995 ["Exists"] = 8707,
4996 ["exist"] = 8707,
4997 ["NotExists"] = 8708,
4998 ["nexist"] = 8708,
4999 ["nexists"] = 8708,
5000 ["empty"] = 8709,
5001 ["emptyset"] = 8709,
5002 ["emptyv"] = 8709,
5003 ["varnothing"] = 8709,
5004 ["Del"] = 8711,
5005 ["nabla"] = 8711,

```

```

5006 ["Element"] = 8712,
5007 ["in"] = 8712,
5008 ["isin"] = 8712,
5009 ["isinv"] = 8712,
5010 ["NotElement"] = 8713,
5011 ["notin"] = 8713,
5012 ["notinva"] = 8713,
5013 ["ReverseElement"] = 8715,
5014 ["SuchThat"] = 8715,
5015 ["ni"] = 8715,
5016 ["niv"] = 8715,
5017 ["NotReverseElement"] = 8716,
5018 ["notni"] = 8716,
5019 ["notniva"] = 8716,
5020 ["Product"] = 8719,
5021 ["prod"] = 8719,
5022 ["Coproduct"] = 8720,
5023 ["coprod"] = 8720,
5024 ["Sum"] = 8721,
5025 ["sum"] = 8721,
5026 ["minus"] = 8722,
5027 ["MinusPlus"] = 8723,
5028 ["mnplus"] = 8723,
5029 ["mp"] = 8723,
5030 ["dotplus"] = 8724,
5031 ["plusdo"] = 8724,
5032 ["Backslash"] = 8726,
5033 ["setminus"] = 8726,
5034 ["setmn"] = 8726,
5035 ["smallsetminus"] = 8726,
5036 ["ssetmn"] = 8726,
5037 ["lowast"] = 8727,
5038 ["SmallCircle"] = 8728,
5039 ["compfn"] = 8728,
5040 ["Sqrt"] = 8730,
5041 ["radic"] = 8730,
5042 ["Proportional"] = 8733,
5043 ["prop"] = 8733,
5044 ["propto"] = 8733,
5045 ["varpropto"] = 8733,
5046 ["vprop"] = 8733,
5047 ["infin"] = 8734,
5048 ["angrt"] = 8735,
5049 ["ang"] = 8736,
5050 ["angle"] = 8736,
5051 ["nang"] = {8736, 8402},
5052 ["angmsd"] = 8737,

```

```

5053 ["measuredangle"] = 8737,
5054 ["angsph"] = 8738,
5055 ["VerticalBar"] = 8739,
5056 ["mid"] = 8739,
5057 ["shortmid"] = 8739,
5058 ["smid"] = 8739,
5059 ["NotVerticalBar"] = 8740,
5060 ["nmid"] = 8740,
5061 ["nshortmid"] = 8740,
5062 ["nsmid"] = 8740,
5063 ["DoubleVerticalBar"] = 8741,
5064 ["par"] = 8741,
5065 ["parallel"] = 8741,
5066 ["shortparallel"] = 8741,
5067 ["spar"] = 8741,
5068 ["NotDoubleVerticalBar"] = 8742,
5069 ["npar"] = 8742,
5070 ["nparallel"] = 8742,
5071 ["nshortparallel"] = 8742,
5072 ["nspar"] = 8742,
5073 ["and"] = 8743,
5074 ["wedge"] = 8743,
5075 ["or"] = 8744,
5076 ["vee"] = 8744,
5077 ["cap"] = 8745,
5078 ["caps"] = {8745, 65024},
5079 ["cup"] = 8746,
5080 ["cups"] = {8746, 65024},
5081 ["Integral"] = 8747,
5082 ["int"] = 8747,
5083 ["Int"] = 8748,
5084 ["iiint"] = 8749,
5085 ["tint"] = 8749,
5086 ["ContourIntegral"] = 8750,
5087 ["conint"] = 8750,
5088 ["oint"] = 8750,
5089 ["Conint"] = 8751,
5090 ["DoubleContourIntegral"] = 8751,
5091 ["Cconint"] = 8752,
5092 ["cwint"] = 8753,
5093 ["ClockwiseContourIntegral"] = 8754,
5094 ["cwconint"] = 8754,
5095 ["CounterClockwiseContourIntegral"] = 8755,
5096 ["awconint"] = 8755,
5097 ["Therefore"] = 8756,
5098 ["there4"] = 8756,
5099 ["therefore"] = 8756,

```



```

5100 ["Because"] = 8757,
5101 ["becaus"] = 8757,
5102 ["because"] = 8757,
5103 ["ratio"] = 8758,
5104 ["Colon"] = 8759,
5105 ["Proportion"] = 8759,
5106 ["dotminus"] = 8760,
5107 ["minusd"] = 8760,
5108 ["mDDot"] = 8762,
5109 ["homtht"] = 8763,
5110 ["Tilde"] = 8764,
5111 ["nvsim"] = {8764, 8402},
5112 ["sim"] = 8764,
5113 ["thicksim"] = 8764,
5114 ["thksim"] = 8764,
5115 ["backsim"] = 8765,
5116 ["bsim"] = 8765,
5117 ["race"] = {8765, 817},
5118 ["ac"] = 8766,
5119 ["acE"] = {8766, 819},
5120 ["mstpos"] = 8766,
5121 ["acd"] = 8767,
5122 ["VerticalTilde"] = 8768,
5123 ["wr"] = 8768,
5124 ["wreath"] = 8768,
5125 ["NotTilde"] = 8769,
5126 ["nsim"] = 8769,
5127 ["EqualTilde"] = 8770,
5128 ["NotEqualTilde"] = {8770, 824},
5129 ["eqsim"] = 8770,
5130 ["esim"] = 8770,
5131 ["nesim"] = {8770, 824},
5132 ["TildeEqual"] = 8771,
5133 ["sime"] = 8771,
5134 ["simeq"] = 8771,
5135 ["NotTildeEqual"] = 8772,
5136 ["nsime"] = 8772,
5137 ["nsimeq"] = 8772,
5138 ["TildeFullEqual"] = 8773,
5139 ["cong"] = 8773,
5140 ["simne"] = 8774,
5141 ["NotTildeFullEqual"] = 8775,
5142 ["ncong"] = 8775,
5143 ["TildeTilde"] = 8776,
5144 ["ap"] = 8776,
5145 ["approx"] = 8776,
5146 ["asymp"] = 8776,

```

```

5147 ["thickapprox"] = 8776,
5148 ["thkap"] = 8776,
5149 ["NotTildeTilde"] = 8777,
5150 ["nap"] = 8777,
5151 ["napprox"] = 8777,
5152 ["ape"] = 8778,
5153 ["approxeq"] = 8778,
5154 ["apid"] = 8779,
5155 ["napid"] = {8779, 824},
5156 ["backcong"] = 8780,
5157 ["bcong"] = 8780,
5158 ["CupCap"] = 8781,
5159 ["asympeq"] = 8781,
5160 ["nvap"] = {8781, 8402},
5161 ["Bumpeq"] = 8782,
5162 ["HumpDownHump"] = 8782,
5163 ["NotHumpDownHump"] = {8782, 824},
5164 ["bump"] = 8782,
5165 ["nbump"] = {8782, 824},
5166 ["HumpEqual"] = 8783,
5167 ["NotHumpEqual"] = {8783, 824},
5168 ["bumpe"] = 8783,
5169 ["bumpeq"] = 8783,
5170 ["nbumpe"] = {8783, 824},
5171 ["DotEqual"] = 8784,
5172 ["doteq"] = 8784,
5173 ["esdot"] = 8784,
5174 ["nedot"] = {8784, 824},
5175 ["doteqdot"] = 8785,
5176 ["eDot"] = 8785,
5177 ["efDot"] = 8786,
5178 ["fallingdotseq"] = 8786,
5179 ["erDot"] = 8787,
5180 ["risingdotseq"] = 8787,
5181 ["Assign"] = 8788,
5182 ["colone"] = 8788,
5183 ["coloneq"] = 8788,
5184 ["ecolon"] = 8789,
5185 ["eqcolon"] = 8789,
5186 ["ecir"] = 8790,
5187 ["eqcirc"] = 8790,
5188 ["circeq"] = 8791,
5189 ["cire"] = 8791,
5190 ["wedgeq"] = 8793,
5191 ["veeeq"] = 8794,
5192 ["triangleq"] = 8796,
5193 ["trie"] = 8796,

```

```

5194 ["equest"] = 8799,
5195 ["questeq"] = 8799,
5196 ["NotEqual"] = 8800,
5197 ["ne"] = 8800,
5198 ["Congruent"] = 8801,
5199 ["bnequiv"] = {8801, 8421},
5200 ["equiv"] = 8801,
5201 ["NotCongruent"] = 8802,
5202 ["nequiv"] = 8802,
5203 ["le"] = 8804,
5204 ["leq"] = 8804,
5205 ["nvle"] = {8804, 8402},
5206 ["GreaterEqual"] = 8805,
5207 ["ge"] = 8805,
5208 ["geq"] = 8805,
5209 ["nvge"] = {8805, 8402},
5210 ["LessFullEqual"] = 8806,
5211 ["lE"] = 8806,
5212 ["leqq"] = 8806,
5213 ["nlE"] = {8806, 824},
5214 ["nleqq"] = {8806, 824},
5215 ["GreaterFullEqual"] = 8807,
5216 ["NotGreaterFullEqual"] = {8807, 824},
5217 ["gE"] = 8807,
5218 ["geqq"] = 8807,
5219 ["ngE"] = {8807, 824},
5220 ["ngeqq"] = {8807, 824},
5221 ["lnE"] = 8808,
5222 ["lneqq"] = 8808,
5223 ["lvertneqq"] = {8808, 65024},
5224 ["lvnE"] = {8808, 65024},
5225 ["gnE"] = 8809,
5226 ["gneqq"] = 8809,
5227 ["gvertneqq"] = {8809, 65024},
5228 ["gvnE"] = {8809, 65024},
5229 ["Lt"] = 8810,
5230 ["NestedLessLess"] = 8810,
5231 ["NotLessLess"] = {8810, 824},
5232 ["ll"] = 8810,
5233 ["nLt"] = {8810, 8402},
5234 ["nLtv"] = {8810, 824},
5235 ["Gt"] = 8811,
5236 ["NestedGreaterGreater"] = 8811,
5237 ["NotGreaterGreater"] = {8811, 824},
5238 ["gg"] = 8811,
5239 ["nGt"] = {8811, 8402},
5240 ["nGtv"] = {8811, 824},

```

```

5241 ["between"] = 8812,
5242 ["twixt"] = 8812,
5243 ["NotCupCap"] = 8813,
5244 ["NotLess"] = 8814,
5245 ["nless"] = 8814,
5246 ["nlt"] = 8814,
5247 ["NotGreater"] = 8815,
5248 ["ngt"] = 8815,
5249 ["ngtr"] = 8815,
5250 ["NotLessEqual"] = 8816,
5251 ["nle"] = 8816,
5252 ["nleq"] = 8816,
5253 ["NotGreaterEqual"] = 8817,
5254 ["nge"] = 8817,
5255 ["ngeq"] = 8817,
5256 ["LessTilde"] = 8818,
5257 ["lesssim"] = 8818,
5258 ["lsim"] = 8818,
5259 ["GreaterTilde"] = 8819,
5260 ["gsim"] = 8819,
5261 ["gtrsim"] = 8819,
5262 ["NotLessTilde"] = 8820,
5263 ["nlsim"] = 8820,
5264 ["NotGreaterTilde"] = 8821,
5265 ["ngsim"] = 8821,
5266 ["LessGreater"] = 8822,
5267 ["lessgtr"] = 8822,
5268 ["lg"] = 8822,
5269 ["GreaterLess"] = 8823,
5270 ["gl"] = 8823,
5271 ["gtrless"] = 8823,
5272 ["NotLessGreater"] = 8824,
5273 ["ntlg"] = 8824,
5274 ["NotGreaterLess"] = 8825,
5275 ["ntgl"] = 8825,
5276 ["Precedes"] = 8826,
5277 ["pr"] = 8826,
5278 ["prec"] = 8826,
5279 ["Succeeds"] = 8827,
5280 ["sc"] = 8827,
5281 ["succ"] = 8827,
5282 ["PrecedesSlantEqual"] = 8828,
5283 ["prcue"] = 8828,
5284 ["preccurlyeq"] = 8828,
5285 ["SucceedsSlantEqual"] = 8829,
5286 ["sccue"] = 8829,
5287 ["succcurlyeq"] = 8829,

```

```

5288 ["PrecedesTilde"] = 8830,
5289 ["precsim"] = 8830,
5290 ["prsim"] = 8830,
5291 ["NotSucceedsTilde"] = {8831, 824},
5292 ["SucceedsTilde"] = 8831,
5293 ["scsim"] = 8831,
5294 ["succsim"] = 8831,
5295 ["NotPrecedes"] = 8832,
5296 ["npr"] = 8832,
5297 ["nprec"] = 8832,
5298 ["NotSucceeds"] = 8833,
5299 ["nsc"] = 8833,
5300 ["nsucc"] = 8833,
5301 ["NotSubset"] = {8834, 8402},
5302 ["nsubset"] = {8834, 8402},
5303 ["sub"] = 8834,
5304 ["subset"] = 8834,
5305 ["vnsup"] = {8834, 8402},
5306 ["NotSuperset"] = {8835, 8402},
5307 ["Superset"] = 8835,
5308 ["nsupset"] = {8835, 8402},
5309 ["sup"] = 8835,
5310 ["supset"] = 8835,
5311 ["vnsup"] = {8835, 8402},
5312 ["nsub"] = 8836,
5313 ["nsup"] = 8837,
5314 ["SubsetEqual"] = 8838,
5315 ["sube"] = 8838,
5316 ["subseteq"] = 8838,
5317 ["SupersetEqual"] = 8839,
5318 ["supe"] = 8839,
5319 ["supseteq"] = 8839,
5320 ["NotSubsetEqual"] = 8840,
5321 ["nsube"] = 8840,
5322 ["nsubseteq"] = 8840,
5323 ["NotSupersetEqual"] = 8841,
5324 ["nsupe"] = 8841,
5325 ["nsupseteq"] = 8841,
5326 ["subne"] = 8842,
5327 ["subsetneq"] = 8842,
5328 ["varsubsetneq"] = {8842, 65024},
5329 ["vsubne"] = {8842, 65024},
5330 ["supne"] = 8843,
5331 ["supsetneq"] = 8843,
5332 ["varsupsetneq"] = {8843, 65024},
5333 ["vsupne"] = {8843, 65024},
5334 ["cupdot"] = 8845,

```

5335 ["UnionPlus"] = 8846,  
 5336 ["uplus"] = 8846,  
 5337 ["NotSquareSubset"] = {8847, 824},  
 5338 ["SquareSubset"] = 8847,  
 5339 ["sqsub"] = 8847,  
 5340 ["sqsubset"] = 8847,  
 5341 ["NotSquareSuperset"] = {8848, 824},  
 5342 ["SquareSuperset"] = 8848,  
 5343 ["sqsup"] = 8848,  
 5344 ["sqsupset"] = 8848,  
 5345 ["SquareSubsetEqual"] = 8849,  
 5346 ["sqsube"] = 8849,  
 5347 ["sqsubseteq"] = 8849,  
 5348 ["SquareSupersetEqual"] = 8850,  
 5349 ["sqsupe"] = 8850,  
 5350 ["sqsupseteq"] = 8850,  
 5351 ["SquareIntersection"] = 8851,  
 5352 ["sqcap"] = 8851,  
 5353 ["sqcaps"] = {8851, 65024},  
 5354 ["SquareUnion"] = 8852,  
 5355 ["sqcup"] = 8852,  
 5356 ["sqcups"] = {8852, 65024},  
 5357 ["CirclePlus"] = 8853,  
 5358 ["oplus"] = 8853,  
 5359 ["CircleMinus"] = 8854,  
 5360 ["ominus"] = 8854,  
 5361 ["CircleTimes"] = 8855,  
 5362 ["otimes"] = 8855,  
 5363 ["osol"] = 8856,  
 5364 ["CircleDot"] = 8857,  
 5365 ["odot"] = 8857,  
 5366 ["circledcirc"] = 8858,  
 5367 ["ocir"] = 8858,  
 5368 ["circledast"] = 8859,  
 5369 ["oast"] = 8859,  
 5370 ["circleddash"] = 8861,  
 5371 ["odash"] = 8861,  
 5372 ["boxplus"] = 8862,  
 5373 ["plusb"] = 8862,  
 5374 ["boxminus"] = 8863,  
 5375 ["minusb"] = 8863,  
 5376 ["boxtimes"] = 8864,  
 5377 ["timesb"] = 8864,  
 5378 ["dotsquare"] = 8865,  
 5379 ["sdotb"] = 8865,  
 5380 ["RightTee"] = 8866,  
 5381 ["vdash"] = 8866,

```

5382 ["LeftTee"] = 8867,
5383 ["dashv"] = 8867,
5384 ["DownTee"] = 8868,
5385 ["top"] = 8868,
5386 ["UpTee"] = 8869,
5387 ["bot"] = 8869,
5388 ["bottom"] = 8869,
5389 ["perp"] = 8869,
5390 ["models"] = 8871,
5391 ["DoubleRightTee"] = 8872,
5392 ["vDash"] = 8872,
5393 ["Vdash"] = 8873,
5394 ["Vvdash"] = 8874,
5395 ["VDash"] = 8875,
5396 ["nvdash"] = 8876,
5397 ["nvDash"] = 8877,
5398 ["nVdash"] = 8878,
5399 ["nVDash"] = 8879,
5400 ["prurel"] = 8880,
5401 ["LeftTriangle"] = 8882,
5402 ["vartriangleleft"] = 8882,
5403 ["vltri"] = 8882,
5404 ["RightTriangle"] = 8883,
5405 ["vartriangleright"] = 8883,
5406 ["vrtri"] = 8883,
5407 ["LeftTriangleEqual"] = 8884,
5408 ["ltrie"] = 8884,
5409 ["nvltrie"] = {8884, 8402},
5410 ["trianglelefteq"] = 8884,
5411 ["RightTriangleEqual"] = 8885,
5412 ["nvrtrie"] = {8885, 8402},
5413 ["rtrie"] = 8885,
5414 ["trianglerighteq"] = 8885,
5415 ["origof"] = 8886,
5416 ["imof"] = 8887,
5417 ["multimap"] = 8888,
5418 ["mumap"] = 8888,
5419 ["hercon"] = 8889,
5420 ["intcal"] = 8890,
5421 ["intercal"] = 8890,
5422 ["veebar"] = 8891,
5423 ["barvee"] = 8893,
5424 ["angrtvb"] = 8894,
5425 ["lrtri"] = 8895,
5426 ["Wedge"] = 8896,
5427 ["bigwedge"] = 8896,
5428 ["xwedge"] = 8896,

```

```

5429 ["Vee"] = 8897,
5430 ["bigvee"] = 8897,
5431 ["xvee"] = 8897,
5432 ["Intersection"] = 8898,
5433 ["bigcap"] = 8898,
5434 ["xcap"] = 8898,
5435 ["Union"] = 8899,
5436 ["bigcup"] = 8899,
5437 ["xcup"] = 8899,
5438 ["Diamond"] = 8900,
5439 ["diam"] = 8900,
5440 ["diamond"] = 8900,
5441 ["sdot"] = 8901,
5442 ["Star"] = 8902,
5443 ["sstarf"] = 8902,
5444 ["divideontimes"] = 8903,
5445 ["divonx"] = 8903,
5446 ["bowtie"] = 8904,
5447 ["ltimes"] = 8905,
5448 ["rtimes"] = 8906,
5449 ["leftthreetimes"] = 8907,
5450 ["lthree"] = 8907,
5451 ["rightthreetimes"] = 8908,
5452 ["rthree"] = 8908,
5453 ["backsimeq"] = 8909,
5454 ["bsime"] = 8909,
5455 ["curlyvee"] = 8910,
5456 ["cuvee"] = 8910,
5457 ["curlywedge"] = 8911,
5458 ["cuwed"] = 8911,
5459 ["Sub"] = 8912,
5460 ["Subset"] = 8912,
5461 ["Sup"] = 8913,
5462 ["Supset"] = 8913,
5463 ["Cap"] = 8914,
5464 ["Cup"] = 8915,
5465 ["fork"] = 8916,
5466 ["pitchfork"] = 8916,
5467 ["epar"] = 8917,
5468 ["lessdot"] = 8918,
5469 ["ltdot"] = 8918,
5470 ["gtdot"] = 8919,
5471 ["gtrdot"] = 8919,
5472 ["Ll"] = 8920,
5473 ["nLl"] = {8920, 824},
5474 ["Gg"] = 8921,
5475 ["ggg"] = 8921,

```



```

5476 ["nGg"] = {8921, 824},
5477 ["LessEqualGreater"] = 8922,
5478 ["leg"] = 8922,
5479 ["lesg"] = {8922, 65024},
5480 ["lesseqgtr"] = 8922,
5481 ["GreaterEqualLess"] = 8923,
5482 ["gel"] = 8923,
5483 ["gesl"] = {8923, 65024},
5484 ["gtreqless"] = 8923,
5485 ["cuepr"] = 8926,
5486 ["curlyeqprec"] = 8926,
5487 ["cuesc"] = 8927,
5488 ["curlyeqsucc"] = 8927,
5489 ["NotPrecedesSlantEqual"] = 8928,
5490 ["nprcue"] = 8928,
5491 ["NotSucceedsSlantEqual"] = 8929,
5492 ["nsccue"] = 8929,
5493 ["NotSquareSubsetEqual"] = 8930,
5494 ["nsqsube"] = 8930,
5495 ["NotSquareSupersetEqual"] = 8931,
5496 ["nsqsupe"] = 8931,
5497 ["lnsim"] = 8934,
5498 ["gnsim"] = 8935,
5499 ["precnsim"] = 8936,
5500 ["prnsim"] = 8936,
5501 ["scnsim"] = 8937,
5502 ["succnsim"] = 8937,
5503 ["NotLeftTriangle"] = 8938,
5504 ["nltri"] = 8938,
5505 ["ntriangleleft"] = 8938,
5506 ["NotRightTriangle"] = 8939,
5507 ["nrtri"] = 8939,
5508 ["ntriangleright"] = 8939,
5509 ["NotLeftTriangleEqual"] = 8940,
5510 ["nltrie"] = 8940,
5511 ["ntrianglelefteq"] = 8940,
5512 ["NotRightTriangleEqual"] = 8941,
5513 ["nrtrie"] = 8941,
5514 ["ntrianglerighteq"] = 8941,
5515 ["vellip"] = 8942,
5516 ["ctdot"] = 8943,
5517 ["utdot"] = 8944,
5518 ["dtdot"] = 8945,
5519 ["disin"] = 8946,
5520 ["isinsv"] = 8947,
5521 ["isins"] = 8948,
5522 ["isindot"] = 8949,

```

```

5523 ["notindot"] = {8949, 824},
5524 ["notinvc"] = 8950,
5525 ["notinvb"] = 8951,
5526 ["isinE"] = 8953,
5527 ["notinE"] = {8953, 824},
5528 ["nisd"] = 8954,
5529 ["xnis"] = 8955,
5530 ["nis"] = 8956,
5531 ["notnivc"] = 8957,
5532 ["notnivb"] = 8958,
5533 ["barwed"] = 8965,
5534 ["barwedge"] = 8965,
5535 ["Barwed"] = 8966,
5536 ["doublebarwedge"] = 8966,
5537 ["LeftCeiling"] = 8968,
5538 ["lceil"] = 8968,
5539 ["RightCeiling"] = 8969,
5540 ["rceil"] = 8969,
5541 ["LeftFloor"] = 8970,
5542 ["lfloor"] = 8970,
5543 ["RightFloor"] = 8971,
5544 ["rfloor"] = 8971,
5545 ["drcrop"] = 8972,
5546 ["dlcrop"] = 8973,
5547 ["urcrop"] = 8974,
5548 ["ulcrop"] = 8975,
5549 ["bnot"] = 8976,
5550 ["proflin"] = 8978,
5551 ["profsurf"] = 8979,
5552 ["telrec"] = 8981,
5553 ["target"] = 8982,
5554 ["ulcorn"] = 8988,
5555 ["ulcorner"] = 8988,
5556 ["urcorn"] = 8989,
5557 ["urcorner"] = 8989,
5558 ["dlcorn"] = 8990,
5559 ["llcorner"] = 8990,
5560 ["drcorn"] = 8991,
5561 ["lrcorn"] = 8991,
5562 ["frown"] = 8994,
5563 ["sfrown"] = 8994,
5564 ["smile"] = 8995,
5565 ["ssmile"] = 8995,
5566 ["cylcty"] = 9005,
5567 ["profalar"] = 9006,
5568 ["topbot"] = 9014,
5569 ["ovbar"] = 9021,

```

```

5570 ["solbar"] = 9023,
5571 ["angzarr"] = 9084,
5572 ["lmoust"] = 9136,
5573 ["lmoustache"] = 9136,
5574 ["rmoust"] = 9137,
5575 ["rmoustache"] = 9137,
5576 ["OverBracket"] = 9140,
5577 ["tbrk"] = 9140,
5578 ["UnderBracket"] = 9141,
5579 ["bbrk"] = 9141,
5580 ["bbrktbrk"] = 9142,
5581 ["OverParenthesis"] = 9180,
5582 ["UnderParenthesis"] = 9181,
5583 ["OverBrace"] = 9182,
5584 ["UnderBrace"] = 9183,
5585 ["trpezium"] = 9186,
5586 ["elinters"] = 9191,
5587 ["blank"] = 9251,
5588 ["circledS"] = 9416,
5589 ["oS"] = 9416,
5590 ["HorizontalLine"] = 9472,
5591 ["boxh"] = 9472,
5592 ["boxv"] = 9474,
5593 ["boxdr"] = 9484,
5594 ["boxdl"] = 9488,
5595 ["boxur"] = 9492,
5596 ["boxul"] = 9496,
5597 ["boxvr"] = 9500,
5598 ["boxvl"] = 9508,
5599 ["boxhd"] = 9516,
5600 ["boxhu"] = 9524,
5601 ["boxvh"] = 9532,
5602 ["boxH"] = 9552,
5603 ["boxV"] = 9553,
5604 ["boxdR"] = 9554,
5605 ["boxDr"] = 9555,
5606 ["boxDR"] = 9556,
5607 ["boxdL"] = 9557,
5608 ["boxDL"] = 9558,
5609 ["boxDL"] = 9559,
5610 ["boxuR"] = 9560,
5611 ["boxUr"] = 9561,
5612 ["boxUR"] = 9562,
5613 ["boxuL"] = 9563,
5614 ["boxUL"] = 9564,
5615 ["boxUL"] = 9565,
5616 ["boxvR"] = 9566,

```

```

5617 ["boxVr"] = 9567,
5618 ["boxVR"] = 9568,
5619 ["boxvL"] = 9569,
5620 ["boxVl"] = 9570,
5621 ["boxVL"] = 9571,
5622 ["boxHd"] = 9572,
5623 ["boxhD"] = 9573,
5624 ["boxHD"] = 9574,
5625 ["boxHu"] = 9575,
5626 ["boxhU"] = 9576,
5627 ["boxHU"] = 9577,
5628 ["boxvH"] = 9578,
5629 ["boxVh"] = 9579,
5630 ["boxVH"] = 9580,
5631 ["uhblk"] = 9600,
5632 ["lhblk"] = 9604,
5633 ["block"] = 9608,
5634 ["blk14"] = 9617,
5635 ["blk12"] = 9618,
5636 ["blk34"] = 9619,
5637 ["Square"] = 9633,
5638 ["squ"] = 9633,
5639 ["square"] = 9633,
5640 ["FilledVerySmallSquare"] = 9642,
5641 ["blacksquare"] = 9642,
5642 ["squarf"] = 9642,
5643 ["squf"] = 9642,
5644 ["EmptyVerySmallSquare"] = 9643,
5645 ["rect"] = 9645,
5646 ["marker"] = 9646,
5647 ["fltns"] = 9649,
5648 ["bigtriangleup"] = 9651,
5649 ["xutri"] = 9651,
5650 ["blacktriangle"] = 9652,
5651 ["utrif"] = 9652,
5652 ["triangle"] = 9653,
5653 ["utri"] = 9653,
5654 ["blacktriangleright"] = 9656,
5655 ["rtrif"] = 9656,
5656 ["rtri"] = 9657,
5657 ["triangleright"] = 9657,
5658 ["bigtriangledown"] = 9661,
5659 ["xdtri"] = 9661,
5660 ["blacktriangledown"] = 9662,
5661 ["dtrif"] = 9662,
5662 ["dtri"] = 9663,
5663 ["triangledown"] = 9663,

```

```

5664 ["blacktriangleleft"] = 9666,
5665 ["ltrif"] = 9666,
5666 ["ltri"] = 9667,
5667 ["triangleleft"] = 9667,
5668 ["loz"] = 9674,
5669 ["lozenge"] = 9674,
5670 ["cir"] = 9675,
5671 ["tridot"] = 9708,
5672 ["bigcirc"] = 9711,
5673 ["xcirc"] = 9711,
5674 ["ultri"] = 9720,
5675 ["urtri"] = 9721,
5676 ["lltri"] = 9722,
5677 ["EmptySmallSquare"] = 9723,
5678 ["FilledSmallSquare"] = 9724,
5679 ["bigstar"] = 9733,
5680 ["starf"] = 9733,
5681 ["star"] = 9734,
5682 ["phone"] = 9742,
5683 ["female"] = 9792,
5684 ["male"] = 9794,
5685 ["spades"] = 9824,
5686 ["spadesuit"] = 9824,
5687 ["clubs"] = 9827,
5688 ["clubsuit"] = 9827,
5689 ["hearts"] = 9829,
5690 ["heartsuit"] = 9829,
5691 ["diamondsuit"] = 9830,
5692 ["diams"] = 9830,
5693 ["sung"] = 9834,
5694 ["flat"] = 9837,
5695 ["natur"] = 9838,
5696 ["natural"] = 9838,
5697 ["sharp"] = 9839,
5698 ["check"] = 10003,
5699 ["checkmark"] = 10003,
5700 ["cross"] = 10007,
5701 ["malt"] = 10016,
5702 ["maltese"] = 10016,
5703 ["sext"] = 10038,
5704 ["VerticalSeparator"] = 10072,
5705 ["lbbbrk"] = 10098,
5706 ["rbbbrk"] = 10099,
5707 ["bsolhsub"] = 10184,
5708 ["suphsol"] = 10185,
5709 ["LeftDoubleBracket"] = 10214,
5710 ["lobrk"] = 10214,

```

5711 ["RightDoubleBracket"] = 10215,  
 5712 ["robrk"] = 10215,  
 5713 ["LeftAngleBracket"] = 10216,  
 5714 ["lang"] = 10216,  
 5715 ["langle"] = 10216,  
 5716 ["RightAngleBracket"] = 10217,  
 5717 ["rang"] = 10217,  
 5718 ["rangle"] = 10217,  
 5719 ["Lang"] = 10218,  
 5720 ["Rang"] = 10219,  
 5721 ["loang"] = 10220,  
 5722 ["roang"] = 10221,  
 5723 ["LongLeftArrow"] = 10229,  
 5724 ["longleftarrow"] = 10229,  
 5725 ["xlarr"] = 10229,  
 5726 ["LongRightArrow"] = 10230,  
 5727 ["longrightarrow"] = 10230,  
 5728 ["xrarr"] = 10230,  
 5729 ["LongLeftRightArrow"] = 10231,  
 5730 ["longleftrightarrow"] = 10231,  
 5731 ["xharr"] = 10231,  
 5732 ["DoubleLongLeftArrow"] = 10232,  
 5733 ["Longleftarrow"] = 10232,  
 5734 ["xlArr"] = 10232,  
 5735 ["DoubleLongRightArrow"] = 10233,  
 5736 ["Longrightarrow"] = 10233,  
 5737 ["xrArr"] = 10233,  
 5738 ["DoubleLongLeftRightArrow"] = 10234,  
 5739 ["Longleftrightarrow"] = 10234,  
 5740 ["xhArr"] = 10234,  
 5741 ["longmapsto"] = 10236,  
 5742 ["xmap"] = 10236,  
 5743 ["dzigrarr"] = 10239,  
 5744 ["nvlArr"] = 10498,  
 5745 ["nvrArr"] = 10499,  
 5746 ["nvHarr"] = 10500,  
 5747 ["Map"] = 10501,  
 5748 ["lbarr"] = 10508,  
 5749 ["bkarow"] = 10509,  
 5750 ["rbarr"] = 10509,  
 5751 ["lBarr"] = 10510,  
 5752 ["dbkarow"] = 10511,  
 5753 ["rBarr"] = 10511,  
 5754 ["RBarr"] = 10512,  
 5755 ["drbkarow"] = 10512,  
 5756 ["DDottrahd"] = 10513,  
 5757 ["UpArrowBar"] = 10514,

```

5758 ["DownArrowBar"] = 10515,
5759 ["Rarrtl"] = 10518,
5760 ["latail"] = 10521,
5761 ["ratail"] = 10522,
5762 ["lAtail"] = 10523,
5763 ["rAtail"] = 10524,
5764 ["larrfs"] = 10525,
5765 ["rarrfs"] = 10526,
5766 ["larrbfs"] = 10527,
5767 ["rarrbfs"] = 10528,
5768 ["nwarhk"] = 10531,
5769 ["nearhk"] = 10532,
5770 ["hksearow"] = 10533,
5771 ["searhk"] = 10533,
5772 ["hkswarow"] = 10534,
5773 ["swarhk"] = 10534,
5774 ["nwnear"] = 10535,
5775 ["nesear"] = 10536,
5776 ["toea"] = 10536,
5777 ["seswar"] = 10537,
5778 ["tosa"] = 10537,
5779 ["swnwar"] = 10538,
5780 ["nrarrc"] = {10547, 824},
5781 ["rarrc"] = 10547,
5782 ["cudarrr"] = 10549,
5783 ["ldca"] = 10550,
5784 ["rdca"] = 10551,
5785 ["cudarrrl"] = 10552,
5786 ["larrpl"] = 10553,
5787 ["curarrm"] = 10556,
5788 ["cularrp"] = 10557,
5789 ["rarrpl"] = 10565,
5790 ["harrcir"] = 10568,
5791 ["Uarrocir"] = 10569,
5792 ["lurdshar"] = 10570,
5793 ["ldrushar"] = 10571,
5794 ["LeftRightVector"] = 10574,
5795 ["RightUpDownVector"] = 10575,
5796 ["DownLeftRightVector"] = 10576,
5797 ["LeftUpDownVector"] = 10577,
5798 ["LeftVectorBar"] = 10578,
5799 ["RightVectorBar"] = 10579,
5800 ["RightUpVectorBar"] = 10580,
5801 ["RightDownVectorBar"] = 10581,
5802 ["DownLeftVectorBar"] = 10582,
5803 ["DownRightVectorBar"] = 10583,
5804 ["LeftUpVectorBar"] = 10584,

```

```

5805 ["LeftDownVectorBar"] = 10585,
5806 ["LeftTeeVector"] = 10586,
5807 ["RightTeeVector"] = 10587,
5808 ["RightUpTeeVector"] = 10588,
5809 ["RightDownTeeVector"] = 10589,
5810 ["DownLeftTeeVector"] = 10590,
5811 ["DownRightTeeVector"] = 10591,
5812 ["LeftUpTeeVector"] = 10592,
5813 ["LeftDownTeeVector"] = 10593,
5814 ["lHar"] = 10594,
5815 ["uHar"] = 10595,
5816 ["rHar"] = 10596,
5817 ["dHar"] = 10597,
5818 ["luruhar"] = 10598,
5819 ["ldrdhar"] = 10599,
5820 ["ruluhar"] = 10600,
5821 ["rdldhar"] = 10601,
5822 ["lharul"] = 10602,
5823 ["llhard"] = 10603,
5824 ["rharul"] = 10604,
5825 ["lrhard"] = 10605,
5826 ["UpEquilibrium"] = 10606,
5827 ["udhar"] = 10606,
5828 ["ReverseUpEquilibrium"] = 10607,
5829 ["duhar"] = 10607,
5830 ["RoundImplies"] = 10608,
5831 ["erarr"] = 10609,
5832 ["simrarr"] = 10610,
5833 ["larrsim"] = 10611,
5834 ["rarrsim"] = 10612,
5835 ["rarrap"] = 10613,
5836 ["ltlarr"] = 10614,
5837 ["gtrarr"] = 10616,
5838 ["subrarr"] = 10617,
5839 ["suplarr"] = 10619,
5840 ["lfisht"] = 10620,
5841 ["rfisht"] = 10621,
5842 ["ufisht"] = 10622,
5843 ["dfisht"] = 10623,
5844 ["lopar"] = 10629,
5845 ["ropar"] = 10630,
5846 ["lbrke"] = 10635,
5847 ["rbrke"] = 10636,
5848 ["lbrkslu"] = 10637,
5849 ["rbrksld"] = 10638,
5850 ["lbrksld"] = 10639,
5851 ["rbrkslu"] = 10640,

```



```

5852 ["langd"] = 10641,
5853 ["rangd"] = 10642,
5854 ["lparlt"] = 10643,
5855 ["rpargt"] = 10644,
5856 ["gtlPar"] = 10645,
5857 ["ltrPar"] = 10646,
5858 ["vzigzag"] = 10650,
5859 ["vangrt"] = 10652,
5860 ["angrtvbd"] = 10653,
5861 ["ange"] = 10660,
5862 ["range"] = 10661,
5863 ["dwangle"] = 10662,
5864 ["uwangle"] = 10663,
5865 ["angmsdaa"] = 10664,
5866 ["angmsdab"] = 10665,
5867 ["angmsdac"] = 10666,
5868 ["angmsdad"] = 10667,
5869 ["angmsdae"] = 10668,
5870 ["angmsdaf"] = 10669,
5871 ["angmsdag"] = 10670,
5872 ["angmsdah"] = 10671,
5873 ["bemptyv"] = 10672,
5874 ["demptyv"] = 10673,
5875 ["cemptyv"] = 10674,
5876 ["raemptyv"] = 10675,
5877 ["laemptyv"] = 10676,
5878 ["ohbar"] = 10677,
5879 ["omid"] = 10678,
5880 ["opar"] = 10679,
5881 ["operp"] = 10681,
5882 ["olcross"] = 10683,
5883 ["odsold"] = 10684,
5884 ["olcir"] = 10686,
5885 ["ofcir"] = 10687,
5886 ["olt"] = 10688,
5887 ["ogt"] = 10689,
5888 ["cirscir"] = 10690,
5889 ["cirE"] = 10691,
5890 ["solb"] = 10692,
5891 ["bsolb"] = 10693,
5892 ["boxbox"] = 10697,
5893 ["trish"] = 10701,
5894 ["rtriltri"] = 10702,
5895 ["LeftTriangleBar"] = 10703,
5896 ["NotLeftTriangleBar"] = {10703, 824},
5897 ["NotRightTriangleBar"] = {10704, 824},
5898 ["RightTriangleBar"] = 10704,

```

```

5899 ["i infin"] = 10716,
5900 ["infintie"] = 10717,
5901 ["nvinfin"] = 10718,
5902 ["eparsl"] = 10723,
5903 ["smeparsl"] = 10724,
5904 ["eqvparsl"] = 10725,
5905 ["blacklozenge"] = 10731,
5906 ["lozf"] = 10731,
5907 ["RuleDelayed"] = 10740,
5908 ["dsol"] = 10742,
5909 ["bigodot"] = 10752,
5910 ["xodot"] = 10752,
5911 ["bigoplus"] = 10753,
5912 ["xoplus"] = 10753,
5913 ["bigotimes"] = 10754,
5914 ["xotime"] = 10754,
5915 ["biguplus"] = 10756,
5916 ["xuplus"] = 10756,
5917 ["bigsqcup"] = 10758,
5918 ["xsqcup"] = 10758,
5919 ["iiiint"] = 10764,
5920 ["qint"] = 10764,
5921 ["fpartint"] = 10765,
5922 ["cirfnint"] = 10768,
5923 ["awint"] = 10769,
5924 ["rppolint"] = 10770,
5925 ["scpolint"] = 10771,
5926 ["npolint"] = 10772,
5927 ["pointint"] = 10773,
5928 ["quatint"] = 10774,
5929 ["intlarhk"] = 10775,
5930 ["pluscir"] = 10786,
5931 ["plusacir"] = 10787,
5932 ["simplus"] = 10788,
5933 ["plusdu"] = 10789,
5934 ["plussim"] = 10790,
5935 ["plustwo"] = 10791,
5936 ["mcomma"] = 10793,
5937 ["minusdu"] = 10794,
5938 ["loplus"] = 10797,
5939 ["roplus"] = 10798,
5940 ["Cross"] = 10799,
5941 ["timesd"] = 10800,
5942 ["timesbar"] = 10801,
5943 ["smashp"] = 10803,
5944 ["lotimes"] = 10804,
5945 ["rotimes"] = 10805,

```

```

5946 ["otimesas"] = 10806,
5947 ["Otimes"] = 10807,
5948 ["odiv"] = 10808,
5949 ["triplus"] = 10809,
5950 ["triminus"] = 10810,
5951 ["tritime"] = 10811,
5952 ["intprod"] = 10812,
5953 ["iprod"] = 10812,
5954 ["amalg"] = 10815,
5955 ["capdot"] = 10816,
5956 ["ncup"] = 10818,
5957 ["ncap"] = 10819,
5958 ["capand"] = 10820,
5959 ["cupor"] = 10821,
5960 ["cupcap"] = 10822,
5961 ["capcup"] = 10823,
5962 ["cupbrcap"] = 10824,
5963 ["capbrcup"] = 10825,
5964 ["cupcup"] = 10826,
5965 ["capcap"] = 10827,
5966 ["ccups"] = 10828,
5967 ["ccaps"] = 10829,
5968 ["ccupssm"] = 10832,
5969 ["And"] = 10835,
5970 ["Or"] = 10836,
5971 ["andand"] = 10837,
5972 ["oror"] = 10838,
5973 ["orslope"] = 10839,
5974 ["andslope"] = 10840,
5975 ["andv"] = 10842,
5976 ["orv"] = 10843,
5977 ["andd"] = 10844,
5978 ["ord"] = 10845,
5979 ["wedbar"] = 10847,
5980 ["sdote"] = 10854,
5981 ["simdot"] = 10858,
5982 ["congdote"] = 10861,
5983 ["ncongdote"] = {10861, 824},
5984 ["easter"] = 10862,
5985 ["apacir"] = 10863,
5986 ["apE"] = 10864,
5987 ["napE"] = {10864, 824},
5988 ["eplus"] = 10865,
5989 ["pluse"] = 10866,
5990 ["Esim"] = 10867,
5991 ["Colone"] = 10868,
5992 ["Equal"] = 10869,

```

```

5993 ["ddotseq"] = 10871,
5994 ["eDDot"] = 10871,
5995 ["equivDD"] = 10872,
5996 ["ltcir"] = 10873,
5997 ["gtcir"] = 10874,
5998 ["ltquest"] = 10875,
5999 ["gtquest"] = 10876,
6000 ["LessSlantEqual"] = 10877,
6001 ["NotLessSlantEqual"] = {10877, 824},
6002 ["leqslant"] = 10877,
6003 ["les"] = 10877,
6004 ["nleqslant"] = {10877, 824},
6005 ["nles"] = {10877, 824},
6006 ["GreaterSlantEqual"] = 10878,
6007 ["NotGreaterSlantEqual"] = {10878, 824},
6008 ["geqslant"] = 10878,
6009 ["ges"] = 10878,
6010 ["ngeqslant"] = {10878, 824},
6011 ["nges"] = {10878, 824},
6012 ["lesdot"] = 10879,
6013 ["gesdot"] = 10880,
6014 ["lesdoto"] = 10881,
6015 ["gesdoto"] = 10882,
6016 ["lesdotor"] = 10883,
6017 ["gesdoto1"] = 10884,
6018 ["lap"] = 10885,
6019 ["lessapprox"] = 10885,
6020 ["gap"] = 10886,
6021 ["gtrapprox"] = 10886,
6022 ["lne"] = 10887,
6023 ["lneq"] = 10887,
6024 ["gne"] = 10888,
6025 ["gneq"] = 10888,
6026 ["lnap"] = 10889,
6027 ["lnapprox"] = 10889,
6028 ["gnap"] = 10890,
6029 ["gnapprox"] = 10890,
6030 ["lEg"] = 10891,
6031 ["lesseqqgtr"] = 10891,
6032 ["gEl"] = 10892,
6033 ["gtreqqless"] = 10892,
6034 ["lsime"] = 10893,
6035 ["gsime"] = 10894,
6036 ["lsimg"] = 10895,
6037 ["gsiml"] = 10896,
6038 ["lgE"] = 10897,
6039 ["glE"] = 10898,

```

```

6040 ["lesges"] = 10899,
6041 ["gesles"] = 10900,
6042 ["els"] = 10901,
6043 ["eqslantless"] = 10901,
6044 ["egs"] = 10902,
6045 ["eqslantgtr"] = 10902,
6046 ["elsdot"] = 10903,
6047 ["egsdot"] = 10904,
6048 ["el"] = 10905,
6049 ["eg"] = 10906,
6050 ["siml"] = 10909,
6051 ["simg"] = 10910,
6052 ["simlE"] = 10911,
6053 ["simgE"] = 10912,
6054 ["LessLess"] = 10913,
6055 ["NotNestedLessLess"] = {10913, 824},
6056 ["GreaterGreater"] = 10914,
6057 ["NotNestedGreaterGreater"] = {10914, 824},
6058 ["glj"] = 10916,
6059 ["gla"] = 10917,
6060 ["ltcc"] = 10918,
6061 ["gtcc"] = 10919,
6062 ["lescc"] = 10920,
6063 ["gescc"] = 10921,
6064 ["smt"] = 10922,
6065 ["lat"] = 10923,
6066 ["smte"] = 10924,
6067 ["smtes"] = {10924, 65024},
6068 ["late"] = 10925,
6069 ["lates"] = {10925, 65024},
6070 ["bumpE"] = 10926,
6071 ["NotPrecedesEqual"] = {10927, 824},
6072 ["PrecedesEqual"] = 10927,
6073 ["npre"] = {10927, 824},
6074 ["npreceq"] = {10927, 824},
6075 ["pre"] = 10927,
6076 ["preceq"] = 10927,
6077 ["NotSucceedsEqual"] = {10928, 824},
6078 ["SucceedsEqual"] = 10928,
6079 ["nsce"] = {10928, 824},
6080 ["nsucceq"] = {10928, 824},
6081 ["sce"] = 10928,
6082 ["succeq"] = 10928,
6083 ["prE"] = 10931,
6084 ["scE"] = 10932,
6085 ["precneqq"] = 10933,
6086 ["prnE"] = 10933,

```

```

6087 ["scnE"] = 10934,
6088 ["succneqq"] = 10934,
6089 ["prap"] = 10935,
6090 ["precapprox"] = 10935,
6091 ["scap"] = 10936,
6092 ["succapprox"] = 10936,
6093 ["precnapprox"] = 10937,
6094 ["prnap"] = 10937,
6095 ["scnap"] = 10938,
6096 ["succnapprox"] = 10938,
6097 ["Pr"] = 10939,
6098 ["Sc"] = 10940,
6099 ["subdot"] = 10941,
6100 ["supdot"] = 10942,
6101 ["subplus"] = 10943,
6102 ["supplus"] = 10944,
6103 ["submult"] = 10945,
6104 ["supmult"] = 10946,
6105 ["subedot"] = 10947,
6106 ["supedot"] = 10948,
6107 ["nsubE"] = {10949, 824},
6108 ["nsubseteqq"] = {10949, 824},
6109 ["subE"] = 10949,
6110 ["subseteqq"] = 10949,
6111 ["nsupE"] = {10950, 824},
6112 ["nsupseteqq"] = {10950, 824},
6113 ["supE"] = 10950,
6114 ["supseteqq"] = 10950,
6115 ["subsim"] = 10951,
6116 ["supsim"] = 10952,
6117 ["subnE"] = 10955,
6118 ["subsetneqq"] = 10955,
6119 ["varsubsetneqq"] = {10955, 65024},
6120 ["vsubnE"] = {10955, 65024},
6121 ["supnE"] = 10956,
6122 ["supsetneqq"] = 10956,
6123 ["varsupsetneqq"] = {10956, 65024},
6124 ["vsupnE"] = {10956, 65024},
6125 ["csub"] = 10959,
6126 ["csup"] = 10960,
6127 ["csube"] = 10961,
6128 ["csupe"] = 10962,
6129 ["subsup"] = 10963,
6130 ["supsub"] = 10964,
6131 ["subsub"] = 10965,
6132 ["supsup"] = 10966,
6133 ["suphsub"] = 10967,

```

```

6134 ["supdsub"] = 10968,
6135 ["forkv"] = 10969,
6136 ["topfork"] = 10970,
6137 ["mlcp"] = 10971,
6138 ["Dashv"] = 10980,
6139 ["DoubleLeftTee"] = 10980,
6140 ["Vdashl"] = 10982,
6141 ["Barv"] = 10983,
6142 ["vBar"] = 10984,
6143 ["vBarv"] = 10985,
6144 ["Vbar"] = 10987,
6145 ["Not"] = 10988,
6146 ["bNot"] = 10989,
6147 ["rnmid"] = 10990,
6148 ["cirmid"] = 10991,
6149 ["midcir"] = 10992,
6150 ["topcir"] = 10993,
6151 ["nhpar"] = 10994,
6152 ["parsim"] = 10995,
6153 ["nparsl"] = {11005, 8421},
6154 ["parsl"] = 11005,
6155 ["fflig"] = 64256,
6156 ["filig"] = 64257,
6157 ["fllig"] = 64258,
6158 ["ffilig"] = 64259,
6159 ["ffllig"] = 64260,
6160 ["Ascr"] = 119964,
6161 ["Cscr"] = 119966,
6162 ["Dscr"] = 119967,
6163 ["Gscr"] = 119970,
6164 ["Jscr"] = 119973,
6165 ["Kscr"] = 119974,
6166 ["Nscr"] = 119977,
6167 ["Oscr"] = 119978,
6168 ["Pscr"] = 119979,
6169 ["Qscr"] = 119980,
6170 ["Sscr"] = 119982,
6171 ["Tscr"] = 119983,
6172 ["Uscr"] = 119984,
6173 ["Vscr"] = 119985,
6174 ["Wscr"] = 119986,
6175 ["Xscr"] = 119987,
6176 ["Yscr"] = 119988,
6177 ["Zscr"] = 119989,
6178 ["ascr"] = 119990,
6179 ["bscr"] = 119991,
6180 ["cscr"] = 119992,

```

```

6181 ["dscr"] = 119993,
6182 ["fscr"] = 119995,
6183 ["hscr"] = 119997,
6184 ["iscr"] = 119998,
6185 ["jscr"] = 119999,
6186 ["kscr"] = 120000,
6187 ["lscr"] = 120001,
6188 ["mscr"] = 120002,
6189 ["nscr"] = 120003,
6190 ["pscr"] = 120005,
6191 ["qscr"] = 120006,
6192 ["rscr"] = 120007,
6193 ["sscr"] = 120008,
6194 ["tscr"] = 120009,
6195 ["uscr"] = 120010,
6196 ["vscr"] = 120011,
6197 ["wscr"] = 120012,
6198 ["xscr"] = 120013,
6199 ["yscr"] = 120014,
6200 ["zscr"] = 120015,
6201 ["Afr"] = 120068,
6202 ["Bfr"] = 120069,
6203 ["Dfr"] = 120071,
6204 ["Efr"] = 120072,
6205 ["Ffr"] = 120073,
6206 ["Gfr"] = 120074,
6207 ["Jfr"] = 120077,
6208 ["Kfr"] = 120078,
6209 ["Lfr"] = 120079,
6210 ["Mfr"] = 120080,
6211 ["Nfr"] = 120081,
6212 ["Ofr"] = 120082,
6213 ["Pfr"] = 120083,
6214 ["Qfr"] = 120084,
6215 ["Sfr"] = 120086,
6216 ["Tfr"] = 120087,
6217 ["Ufr"] = 120088,
6218 ["Vfr"] = 120089,
6219 ["Wfr"] = 120090,
6220 ["Xfr"] = 120091,
6221 ["Yfr"] = 120092,
6222 ["afr"] = 120094,
6223 ["bfr"] = 120095,
6224 ["cfr"] = 120096,
6225 ["dfr"] = 120097,
6226 ["efr"] = 120098,
6227 ["ffr"] = 120099,

```



```

6228 ["gfr"] = 120100,
6229 ["hfr"] = 120101,
6230 ["ifr"] = 120102,
6231 ["jfr"] = 120103,
6232 ["kfr"] = 120104,
6233 ["lfr"] = 120105,
6234 ["mfr"] = 120106,
6235 ["nfr"] = 120107,
6236 ["ofr"] = 120108,
6237 ["pfr"] = 120109,
6238 ["qfr"] = 120110,
6239 ["rfr"] = 120111,
6240 ["sfr"] = 120112,
6241 ["tfr"] = 120113,
6242 ["ufr"] = 120114,
6243 ["vfr"] = 120115,
6244 ["wfr"] = 120116,
6245 ["xfr"] = 120117,
6246 ["yfr"] = 120118,
6247 ["zfr"] = 120119,
6248 ["Aopf"] = 120120,
6249 ["Bopf"] = 120121,
6250 ["Dopf"] = 120123,
6251 ["Eopf"] = 120124,
6252 ["Fopf"] = 120125,
6253 ["Gopf"] = 120126,
6254 ["Iopf"] = 120128,
6255 ["Jopf"] = 120129,
6256 ["Kopf"] = 120130,
6257 ["Lopf"] = 120131,
6258 ["Mopf"] = 120132,
6259 ["Oopf"] = 120134,
6260 ["Sopf"] = 120138,
6261 ["Topf"] = 120139,
6262 ["Uopf"] = 120140,
6263 ["Vopf"] = 120141,
6264 ["Wopf"] = 120142,
6265 ["Xopf"] = 120143,
6266 ["Yopf"] = 120144,
6267 ["aopf"] = 120146,
6268 ["bopf"] = 120147,
6269 ["copf"] = 120148,
6270 ["dopf"] = 120149,
6271 ["eopf"] = 120150,
6272 ["fopf"] = 120151,
6273 ["gopf"] = 120152,
6274 ["hopf"] = 120153,

```

```

6275 ["iopf"] = 120154,
6276 ["jopf"] = 120155,
6277 ["kopf"] = 120156,
6278 ["lopf"] = 120157,
6279 ["mopf"] = 120158,
6280 ["nopf"] = 120159,
6281 ["oopf"] = 120160,
6282 ["popf"] = 120161,
6283 ["qopf"] = 120162,
6284 ["ropf"] = 120163,
6285 ["sopf"] = 120164,
6286 ["topf"] = 120165,
6287 ["uopf"] = 120166,
6288 ["vopf"] = 120167,
6289 ["wopf"] = 120168,
6290 ["xopf"] = 120169,
6291 ["yopf"] = 120170,
6292 ["zopf"] = 120171,
6293 }

```

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

```

6294 function entities.dec_entity(s)
6295 local n = tonumber(s)
6296 if n == nil then
6297 return "&#" .. s .. ";" -- fallback for unknown entities
6298 end
6299 return utf8.char(n)
6300 end

```

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

```

6301 function entities.hex_entity(s)
6302 local n = tonumber("0x"..s)
6303 if n == nil then
6304 return "&#x" .. s .. ";" -- fallback for unknown entities
6305 end
6306 return utf8.char(n)
6307 end

```

Given a captured character `x` and a string `s` of hexadecimal digits, the `entities.hex_entity_with_x_char` returns the corresponding UTF8-encoded Unicode codepoint or fallback with the `x` character.

```

6308 function entities.hex_entity_with_x_char(x, s)
6309 local n = tonumber("0x"..s)
6310 if n == nil then
6311 return "&#" .. x .. s .. ";" -- fallback for unknown entities
6312 end

```

```

6313 return utf8.char(n)
6314 end

```

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

```

6315 function entities.char_entity(s)
6316 local code_points = character_entities[s]
6317 if code_points == nil then
6318 return "&" .. s .. ";"
6319 end
6320 if type(code_points) ~= 'table' then
6321 code_points = {code_points}
6322 end
6323 local char_table = {}
6324 for _, code_point in ipairs(code_points) do
6325 table.insert(char_table, utf8.char(code_point))
6326 end
6327 return table.concat(char_table)
6328 end

```

### 3.1.3 Plain T<sub>E</sub>X Writer

This section documents the `writer` object, which implements the routines for producing the T<sub>E</sub>X output. The object is an amalgamate of the generic, T<sub>E</sub>X, L<sup>A</sup>T<sub>E</sub>X writer objects that were located in the `lunamark/writer/generic.lua`, `lunamark/writer/tex.lua`, and `lunamark/writer/latex.lua` files in the Luna-mark Lua module.

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

```

6329 M.writer = {}

```

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

The objects produced by the `writer.new` method expose instance methods and variables of their own. As a convention, I will refer to these *member*s as `writer->member`. All member variables are immutable unless explicitly stated otherwise.

```

6330 local parsers
6331 function M.writer.new(options)
6332 local self = {}

```

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

```

6333 self.options = options

```

Define `writer->flatten_inlines`, which indicates whether or not the writer should produce raw text rather than text in the output format for inline elements. The `writer->flatten_inlines` member variable is mutable.

```
6334 self.flatten_inlines = false
```

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

```
6335 local slice_specifiers = {}
6336 for specifier in options.slice:gmatch("[^%s]+") do
6337 table.insert(slice_specifiers, specifier)
6338 end
6339
6340 if #slice_specifiers == 2 then
6341 self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
6342 local slice_begin_type = self.slice_begin:sub(1, 1)
6343 if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
6344 self.slice_begin = "^" .. self.slice_begin
6345 end
6346 local slice_end_type = self.slice_end:sub(1, 1)
6347 if slice_end_type ~= "^" and slice_end_type ~= "$" then
6348 self.slice_end = "$" .. self.slice_end
6349 end
6350 elseif #slice_specifiers == 1 then
6351 self.slice_begin = "^" .. slice_specifiers[1]
6352 self.slice_end = "$" .. slice_specifiers[1]
6353 end
6354
6355 self.slice_begin_type = self.slice_begin:sub(1, 1)
6356 self.slice_begin_identifier = self.slice_begin:sub(2) or ""
6357 self.slice_end_type = self.slice_end:sub(1, 1)
6358 self.slice_end_identifier = self.slice_end:sub(2) or ""
6359
6360 if self.slice_begin == "^" and self.slice_end ~= "^" then
6361 self.is_writing = true
6362 else
6363 self.is_writing = false
6364 end
```

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

```
6365 self.space = " "
```

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

```
6366 self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
6367 function self.plain(s)
6368 return s
```

```
6369 end
```

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

```
6370 function self.paragraph(s)
6371 if not self.is_writing then return "" end
6372 return s
6373 end
```

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

```
6374 self.interblocksep_text = "\\markdownRendererInterblockSeparator\n{"
6375 function self.interblocksep()
6376 if not self.is_writing then return "" end
6377 return self.interblocksep_text
6378 end
```

Define `writer->paragraphsep` as the output format of a paragraph separator. Users can use more than one blank line to delimit two blocks to indicate the end of a series of blocks that make up a paragraph. This produces a paragraph separator instead of an interblock separator.

```
6379 self.paragraphsep_text = "\\markdownRendererParagraphSeparator\n{"
6380 function self.paragraphsep()
6381 if not self.is_writing then return "" end
6382 return self.paragraphsep_text
6383 end
```

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

```
6384 self.undosep_text = "\\markdownRendererUndoSeparator\n{"
6385 function self.undosep()
6386 if not self.is_writing then return "" end
6387 return self.undosep_text
6388 end
```

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

```
6389 self.soft_line_break = function()
6390 if self.flatten_inlines then return "\n" end
6391 return "\\markdownRendererSoftLineBreak\n{"
6392 end
```

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

```
6393 self.hard_line_break = function()
6394 if self.flatten_inlines then return "\n" end
6395 return "\\markdownRendererHardLineBreak\n{"
6396 end
```

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

```
6397 self.ellipsis = "\\markdownRendererEllipsis{"
```

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

```
6398 function self.thematic_break()
6399 if not self.is_writing then return "" end
6400 return "\\markdownRendererThematicBreak{"
6401 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.

```
6402 self.escaped_uri_chars = {
6403 ["{"] = "\\markdownRendererLeftBrace{"},
6404 ["}"] = "\\markdownRendererRightBrace{"},
6405 ["\\"] = "\\markdownRendererBackslash{"},
6406 ["\r"] = " ",
6407 ["\n"] = " ",
6408 }
6409 self.escaped_minimal_strings = {
6410 ["^"] = "\\markdownRendererCircumflex",
6411 .. "\\markdownRendererCircumflex ",
6412 ["☒"] = "\\markdownRendererTickedBox{"},
6413 ["☐"] = "\\markdownRendererHalfTickedBox{"},
6414 ["□"] = "\\markdownRendererUntickedBox{"},
6415 [entities.hex_entity('FFFD')]
6416 = "\\markdownRendererReplacementCharacter{"},
6417 }
```

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

```
6418 self.escaped_strings = util.table_copy(self.escaped_minimal_strings)
6419 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.

```
6420 self.escaped_chars = {
6421 ["{"] = "\\markdownRendererLeftBrace{"},
6422 ["}"] = "\\markdownRendererRightBrace{"},
6423 ["%"] = "\\markdownRendererPercentSign{"},
6424 ["\\"] = "\\markdownRendererBackslash{"},
6425 ["#"] = "\\markdownRendererHash{"},
6426 ["$"] = "\\markdownRendererDollarSign{"},
6427 ["&"] = "\\markdownRendererAmpersand{"},
6428 ["_"] = "\\markdownRendererUnderscore{"},
6429 ["^"] = "\\markdownRendererCircumflex{"},
6430 ["~"] = "\\markdownRendererTilde{"},
6431 ["|"] = "\\markdownRendererPipe{"},
6432 [entities.hex_entity('0000')]
6433 }
```

```

6433 = "\\markdownRendererReplacementCharacter{}",
6434 }

```

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

```

6435 local function create_escaper(char_escapes, string_escapes)
6436 local escape = util.escaper(char_escapes, string_escapes)
6437 return function(s)
6438 if self.flatten_inlines then return s end
6439 return escape(s)
6440 end
6441 end
6442 local escape_typographic_text = create_escaper(
6443 self.escaped_chars, self.escaped_strings)
6444 local escape_programmatic_text = create_escaper(
6445 self.escaped_uri_chars, self.escaped_minimal_strings)
6446 local escape_minimal = create_escaper(
6447 {}, self.escaped_minimal_strings)

```

Define the following semantic aliases for the escaper functions:

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

```

6448 self.escape = escape_typographic_text
6449 self.math = escape_minimal
6450 if options.hybrid then
6451 self.identifier = escape_minimal
6452 self.string = escape_minimal
6453 self.uri = escape_minimal
6454 self.infostring = escape_minimal
6455 else
6456 self.identifier = escape_programmatic_text
6457 self.string = escape_typographic_text
6458 self.uri = escape_programmatic_text
6459 self.infostring = escape_programmatic_text
6460 end

```

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

```

6461 function self.warning(t, m)
6462 return {"\\markdownRendererWarning{" , self.escape(t), "}{" ,
6463 escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
6464 escape_minimal(m or ""), "}"}
6465 end

```

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

```

6466 function self.error(t, m)
6467 return {"\\markdownRendererError{" , self.escape(t), "}{" ,
6468 escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
6469 escape_minimal(m or ""), "}"}
6470 end

```

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

```

6471 function self.code(s, attributes)
6472 if self.flatten_inlines then return s end
6473 local buf = {}
6474 if attributes ~= nil then
6475 table.insert(buf,
6476 "\\markdownRendererCodeSpanAttributeContextBegin\n")
6477 table.insert(buf, self.attributes(attributes))
6478 end
6479 table.insert(buf,
6480 {"\\markdownRendererCodeSpan{" , self.escape(s), "}"}
6481 if attributes ~= nil then
6482 table.insert(buf,
6483 "\\markdownRendererCodeSpanAttributeContextEnd{")
6484 end
6485 return buf
6486 end

```

Define `writer->link` as a function that will transform an input hyperlink to the output format, where `lab` corresponds to the label, `src` to URI, `tit` to the title of the link, and `attributes` to optional attributes.

```

6487 function self.link(lab, src, tit, attributes)
6488 if self.flatten_inlines then return lab end
6489 local buf = {}
6490 if attributes ~= nil then
6491 table.insert(buf,
6492 "\\markdownRendererLinkAttributeContextBegin\n")
6493 table.insert(buf, self.attributes(attributes))
6494 end
6495 table.insert(buf, {"\\markdownRendererLink{" , lab, "}"} ,
6496 {"", self.escape(src), "}"} ,
6497 {"", self.uri(src), "}"} ,
6498 {"", self.string(tit or ""), "}"})

```



```

6499 if attributes ~= nil then
6500 table.insert(buf,
6501 "\\markdownRendererLinkAttributeContextEnd{ }")
6502 end
6503 return buf
6504 end

```

Define `writer->image` as a function that will transform an input image to the output format, where `lab` corresponds to the label, `src` to the URL, `tit` to the title of the image, and `attributes` to optional attributes.

```

6505 function self.image(lab, src, tit, attributes)
6506 if self.flatten_inlines then return lab end
6507 local buf = {}
6508 if attributes ~= nil then
6509 table.insert(buf,
6510 "\\markdownRendererImageAttributeContextBegin\n")
6511 table.insert(buf, self.attributes(attributes))
6512 end
6513 table.insert(buf, {"\\markdownRendererImage{",lab,"",
6514 "{",self.string(src),"}",
6515 "{",self.uri(src),"}",
6516 "{",self.string(tit or ""),"}"}))
6517 if attributes ~= nil then
6518 table.insert(buf,
6519 "\\markdownRendererImageAttributeContextEnd{ }")
6520 end
6521 return buf
6522 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.

```

6523 function self.bulletlist(items,tight)
6524 if not self.is_writing then return "" end
6525 local buffer = {}
6526 for _,item in ipairs(items) do
6527 if item ~= "" then
6528 buffer[#buffer + 1] = self.bulletitem(item)
6529 end
6530 end
6531 local contents = util.intersperse(buffer,"\n")
6532 if tight and options.tightLists then
6533 return {"\\markdownRendererUlBeginTight\n",contents,
6534 "\n\\markdownRendererUlEndTight "}
6535 else
6536 return {"\\markdownRendererUlBegin\n",contents,
6537 "\n\\markdownRendererUlEnd "}

```

```

6538 end
6539 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.

```

6540 function self.bulletitem(s)
6541 return {"\\markdownRenderUListItem ",s,
6542 "\\markdownRenderUListItemEnd "}
6543 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.

```

6544 function self.orderedlist(items,tight,startnum)
6545 if not self.is_writing then return "" end
6546 local buffer = {}
6547 local num = startnum
6548 for _,item in ipairs(items) do
6549 if item ~= "" then
6550 buffer[#buffer + 1] = self.ordereditem(item,num)
6551 end
6552 if num ~= nil and item ~= "" then
6553 num = num + 1
6554 end
6555 end
6556 local contents = util.intersperse(buffer,"\n")
6557 if tight and options.tightLists then
6558 return {"\\markdownRenderO1BeginTight\n",contents,
6559 "\n\\markdownRenderO1EndTight "}
6560 else
6561 return {"\\markdownRenderO1Begin\n",contents,
6562 "\n\\markdownRenderO1End "}
6563 end
6564 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.

```

6565 function self.ordereditem(s,num)
6566 if num ~= nil then
6567 return {"\\markdownRenderO1ItemWithNumber{",num,"}",s,
6568 "\\markdownRenderO1ItemEnd "}
6569 else
6570 return {"\\markdownRenderO1Item ",s,
6571 "\\markdownRenderO1ItemEnd "}
6572 end
6573 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.

```
6574 function self.inline_html_comment(contents)
6575 if self.flatten_inlines then return contents end
6576 return {"\\markdownRendererInlineHtmlComment{" ,contents,""} }
6577 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.

```
6578 function self.inline_html_tag(contents)
6579 if self.flatten_inlines then return contents end
6580 return {"\\markdownRendererInlineHtmlTag{" ,
6581 self.string(contents),""} }
6582 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.

```
6583 function self.block_html_element(s)
6584 if not self.is_writing then return "" end
6585 local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
6586 return {"\\markdownRendererInputBlockHtmlElement{" ,name,""} }
6587 end
```

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

```
6588 function self.emphasis(s)
6589 if self.flatten_inlines then return s end
6590 return {"\\markdownRendererEmphasis{" ,s,""} }
6591 end
```

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

```
6592 function self.checkbox(f)
6593 if f == 1.0 then
6594 return "☒ "
6595 elseif f == 0.0 then
6596 return "☐ "
6597 else
6598 return "◻ "
6599 end
6600 end
```

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

```
6601 function self.strong(s)
```

```

6602 if self.flatten_inlines then return s end
6603 return {"\\markdownRendererStrongEmphasis{" ,s,""}
6604 end

```

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

```

6605 function self.blockquote(s)
6606 if not self.is_writing then return "" end
6607 return {"\\markdownRendererBlockQuoteBegin\n",s,
6608 "\\markdownRendererBlockQuoteEnd "}
6609 end

```

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

```

6610 function self.verbatim(s)
6611 if not self.is_writing then return "" end
6612 s = s:gsub("\n$", "")
6613 local name = util.cache_verbatim(options.cacheDir, s)
6614 return {"\\markdownRendererInputVerbatim{" ,name,""}
6615 end

```

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

```

6616 function self.document(d)
6617 local buf = {"\\markdownRendererDocumentBegin\n"}
6618
6619 -- warn against the `hybrid` option
6620 if options.hybrid then
6621 local text = "The `hybrid` option has been soft-deprecated."
6622 local more = "Consider using one of the following better options "
6623 .. "for mixing TeX and markdown: `contentBlocks`, "
6624 .. "`rawAttribute`, `texComments`, `texMathDollars`, "
6625 .. "`texMathSingleBackslash`, and "
6626 .. "`texMathDoubleBackslash`. "
6627 .. "For more information, see the user manual at "
6628 .. "<https://witiko.github.io/markdown/>."
6629 table.insert(buf, self.warning(text, more))
6630 end
6631
6632 -- insert the text of the document
6633 table.insert(buf, d)
6634
6635 -- pop all attributes
6636 table.insert(buf, self.pop_attributes())
6637
6638 table.insert(buf, "\\markdownRendererDocumentEnd")
6639
6640 return buf

```

```
6641 end
```

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

```
6642 local seen_identifiers = {}
6643 local key_value_regex = "([^\s=]+)%s*=%s*(.*)"
6644 local function normalize_attributes(attributes, auto_identifiers)
6645 -- normalize attributes
6646 local normalized_attributes = {}
6647 local has_explicit_identifiers = false
6648 local key, value
6649 for _, attribute in ipairs(attributes or {}) do
6650 if attribute:sub(1, 1) == "#" then
6651 table.insert(normalized_attributes, attribute)
6652 has_explicit_identifiers = true
6653 seen_identifiers[attribute:sub(2)] = true
6654 elseif attribute:sub(1, 1) == "." then
6655 table.insert(normalized_attributes, attribute)
6656 else
6657 key, value = attribute:match(key_value_regex)
6658 if key:lower() == "id" then
6659 table.insert(normalized_attributes, "#" .. value)
6660 elseif key:lower() == "class" then
6661 local classes = {}
6662 for class in value:gmatch("%S+") do
6663 table.insert(classes, class)
6664 end
6665 table.sort(classes)
6666 for _, class in ipairs(classes) do
6667 table.insert(normalized_attributes, "." .. class)
6668 end
6669 else
6670 table.insert(normalized_attributes, attribute)
6671 end
6672 end
6673 end
6674
6675 -- if no explicit identifiers exist, add auto identifiers
6676 if not has_explicit_identifiers and auto_identifiers ~= nil then
6677 local seen_auto_identifiers = {}
6678 for _, auto_identifier in ipairs(auto_identifiers) do
6679 if seen_auto_identifiers[auto_identifier] == nil then
6680 seen_auto_identifiers[auto_identifier] = true
6681 if seen_identifiers[auto_identifier] == nil then
6682 seen_identifiers[auto_identifier] = true
6683 table.insert(normalized_attributes,
6684 "#" .. auto_identifier)
```

```

6685 else
6686 local auto_identifier_number = 1
6687 while true do
6688 local numbered_auto_identifier = auto_identifier .. "-"
6689 .. auto_identifier_number
6690 if seen_identifiers[numbered_auto_identifier] == nil then
6691 seen_identifiers[numbered_auto_identifier] = true
6692 table.insert(normalized_attributes,
6693 "#" .. numbered_auto_identifier)
6694 break
6695 end
6696 auto_identifier_number = auto_identifier_number + 1
6697 end
6698 end
6699 end
6700 end
6701 end
6702
6703 -- sort and deduplicate normalized attributes
6704 table.sort(normalized_attributes)
6705 local seen_normalized_attributes = {}
6706 local deduplicated_normalized_attributes = {}
6707 for _, attribute in ipairs(normalized_attributes) do
6708 if seen_normalized_attributes[attribute] == nil then
6709 seen_normalized_attributes[attribute] = true
6710 table.insert(deduplicated_normalized_attributes, attribute)
6711 end
6712 end
6713
6714 return deduplicated_normalized_attributes
6715 end
6716
6717 function self.attributes(attributes, should_normalize_attributes)
6718 local normalized_attributes
6719 if should_normalize_attributes == false then
6720 normalized_attributes = attributes
6721 else
6722 normalized_attributes = normalize_attributes(attributes)
6723 end
6724
6725 local buf = {}
6726 local key, value
6727 for _, attribute in ipairs(normalized_attributes) do
6728 if attribute:sub(1, 1) == "#" then
6729 table.insert(buf, {"\\markdownRendererAttributeIdentifier{",
6730 attribute:sub(2), "}"}))
6731 elseif attribute:sub(1, 1) == "." then

```

```

6732 table.insert(buf, {"\\markdownRendererAttributeClassName{",
6733 attribute:sub(2), "}"}))
6734 else
6735 key, value = attribute:match(key_value_regex)
6736 table.insert(buf, {"\\markdownRendererAttributeKeyValue{",
6737 key, "}{" , value, "}"}))
6738 end
6739 end
6740
6741 return buf
6742 end

```

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

```

6743 self.active_attributes = {}

```

Define `writer->attribute_type_levels` as a hash table that maps attribute types to the number of attributes of said type in `writer->active_attributes`.

```

6744 self.attribute_type_levels = {}
6745 setmetatable(self.attribute_type_levels,
6746 { __index = function() return 0 end })

```

Define `writer->push_attributes` and `writer->pop_attributes` as functions that will add a new set of active block-level attributes or remove the most current attributes from `writer->active_attributes`.

```

6747 local function apply_attributes()
6748 local buf = {}
6749 for i = 1, #self.active_attributes do
6750 local start_output = self.active_attributes[i][3]
6751 if start_output ~= nil then
6752 table.insert(buf, start_output)
6753 end
6754 end
6755 return buf
6756 end
6757
6758 local function tear_down_attributes()
6759 local buf = {}
6760 for i = #self.active_attributes, 1, -1 do
6761 local end_output = self.active_attributes[i][4]
6762 if end_output ~= nil then
6763 table.insert(buf, end_output)
6764 end
6765 end
6766 return buf
6767 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.

```

6768 function self.push_attributes(attribute_type, attributes,
6769 start_output, end_output)
6770 local attribute_type_level
6771 = self.attribute_type_levels[attribute_type]
6772 self.attribute_type_levels[attribute_type]
6773 = attribute_type_level + 1
6774
6775 -- index attributes in a hash table for easy lookup
6776 attributes = attributes or {}
6777 for i = 1, #attributes do
6778 attributes[attributes[i]] = true
6779 end
6780
6781 local buf = {}
6782 -- handle slicing
6783 if attributes["#" .. self.slice_end_identifier] ~= nil and
6784 self.slice_end_type == "^" then
6785 if self.is_writing then
6786 table.insert(buf, self.undosep())
6787 table.insert(buf, tear_down_attributes())
6788 end
6789 self.is_writing = false
6790 end
6791 if attributes["#" .. self.slice_begin_identifier] ~= nil and
6792 self.slice_begin_type == "^" then
6793 table.insert(buf, apply_attributes())
6794 self.is_writing = true
6795 end
6796 if self.is_writing and start_output ~= nil then
6797 table.insert(buf, start_output)
6798 end
6799 table.insert(self.active_attributes,
6800 {attribute_type, attributes,
6801 start_output, end_output})
6802 return buf
6803 end
6804

```

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](#)).

```
6805 function self.pop_attributes(attribute_type)
6806 local buf = {}
6807 -- pop attributes until we find attributes of correct type
6808 -- or until no attributes remain
6809 local current_attribute_type = false
6810 while current_attribute_type ~= attribute_type and
6811 #self.active_attributes > 0 do
6812 local attributes, _, end_output
6813 current_attribute_type, attributes, _, end_output = table.unpack(
6814 self.active_attributes[#self.active_attributes])
6815 local attribute_type_level
6816 = self.attribute_type_levels[current_attribute_type]
6817 self.attribute_type_levels[current_attribute_type]
6818 = attribute_type_level - 1
6819 if self.is_writing and end_output ~= nil then
6820 table.insert(buf, end_output)
6821 end
6822 table.remove(self.active_attributes, #self.active_attributes)
6823 -- handle slicing
6824 if attributes["#" .. self.slice_end_identifier] ~= nil
6825 and self.slice_end_type == "$" then
6826 if self.is_writing then
6827 table.insert(buf, self.undosep())
6828 table.insert(buf, tear_down_attributes())
6829 end
6830 self.is_writing = false
6831 end
6832 if attributes["#" .. self.slice_begin_identifier] ~= nil and
6833 self.slice_begin_type == "$" then
6834 self.is_writing = true
6835 table.insert(buf, apply_attributes())
6836 end
6837 end
6838 return buf
6839 end
```

Create an auto identifier string by stripping and converting characters from string [s](#).

```
6840 local function create_auto_identifier(s)
6841 local buffer = {}
6842 local prev_space = false
6843 local letter_found = false
6844 local normalized_s = s
6845 if not options.unicodeNormalization
6846 or options.unicodeNormalizationForm ~= "nfc" then
6847 normalized_s = uni_algos.normalize.NFC(normalized_s)
```

```

6848 end
6849
6850 for _, code in utf8.codes(normalized_s) do
6851 local char = utf8.char(code)
6852
6853 -- Remove everything up to the first letter.
6854 if not letter_found then
6855 local is_letter = lpeg.match(
6856 parsers.unicode.following_alpha,
6857 char
6858)
6859 if is_letter then
6860 letter_found = true
6861 else
6862 goto continue
6863 end
6864 end
6865
6866 -- Remove all non-alphanumeric characters, except underscores,
6867 -- hyphens, and periods.
6868 if not lpeg.match(
6869 (parsers.underscore
6870 + parsers.dash
6871 + parsers.period
6872 + parsers.unicode.following_word
6873 + parsers.unicode.following_whitespace),
6874 char
6875) then
6876 goto continue
6877 end
6878
6879 -- Replace all spaces and newlines with hyphens.
6880 if lpeg.match(
6881 (parsers.newline
6882 + parsers.unicode.following_whitespace),
6883 char
6884) then
6885 char = "-"
6886 if prev_space then
6887 goto continue
6888 else
6889 prev_space = true
6890 end
6891 else
6892 -- Case-fold all alphabetic characters.
6893 char = uni_algos.case.casefold(char)
6894 prev_space = false

```

```

6895 end
6896
6897 table.insert(buffer, char)
6898
6899 ::continue::
6900 end
6901
6902 if prev_space then
6903 table.remove(buffer)
6904 end
6905
6906 local identifier = #buffer == 0 and "section"
6907 or table.concat(buffer, "")
6908 return identifier
6909 end

```

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

```

6910 local function create_gfm_auto_identifier(s)
6911 local buffer = {}
6912 local prev_space = false
6913 local letter_found = false
6914 local normalized_s = s
6915 if not options.unicodeNormalization
6916 or options.unicodeNormalizationForm ~= "nfc" then
6917 normalized_s = uni_algos.normalize.NFC(normalized_s)
6918 end
6919
6920 for _, code in utf8.codes(normalized_s) do
6921 local char = utf8.char(code)
6922
6923 -- Remove everything up to the first non-space.
6924 if not letter_found then
6925 local is_letter = not lpeg.match(
6926 parsers.unicode.following_whitespace,
6927 char
6928)
6929 if is_letter then
6930 letter_found = true
6931 else
6932 goto continue
6933 end
6934 end
6935
6936 -- Remove all non-alphanumeric characters, except underscores
6937 -- and hyphens.
6938 if not lpeg.match(

```

```

6939 (parsers.underscore
6940 + parsers.dash
6941 + parsers.unicode.following_word
6942 + parsers.unicode.following_whitespace),
6943 char
6944) then
6945 prev_space = false
6946 goto continue
6947 end
6948
6949 -- Replace all spaces and newlines with hyphens.
6950 if lpeg.match(
6951 (parsers.newline
6952 + parsers.unicode.following_whitespace),
6953 char
6954) then
6955 char = "-"
6956 if prev_space then
6957 goto continue
6958 else
6959 prev_space = true
6960 end
6961 else
6962 -- Case-fold all alphabetic characters.
6963 char = uni_algos.case.casefold(char)
6964 prev_space = false
6965 end
6966
6967 table.insert(buffer, char)
6968
6969 ::continue::
6970 end
6971
6972 if prev_space then
6973 table.remove(buffer)
6974 end
6975
6976 local identifier = #buffer == 0 and "section"
6977 or table.concat(buffer, "")
6978 return identifier
6979 end

```

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

```

6980 self.secbegin_text = "\\markdownRendererSectionBegin\n"
6981 self.seccend_text = "\n\\markdownRendererSectionEnd "
6982 function self.heading(s, level, attributes)

```

```

6983 local buf = {}
6984 local flat_text, inlines = table.unpack(s)
6985
6986 -- push empty attributes for implied sections
6987 while self.attribute_type_levels["heading"] < level - 1 do
6988 table.insert(buf,
6989 self.push_attributes("heading",
6990 nil,
6991 self.secbegin_text,
6992 self.secend_text))
6993 end
6994
6995 -- pop attributes for sections that have ended
6996 while self.attribute_type_levels["heading"] >= level do
6997 table.insert(buf, self.pop_attributes("heading"))
6998 end
6999
7000 -- construct attributes for the new section
7001 local auto_identifiers = {}
7002 if self.options.autoIdentifiers then
7003 table.insert(auto_identifiers, create_auto_identifier(flat_text))
7004 end
7005 if self.options.gfmAutoIdentifiers then
7006 table.insert(auto_identifiers,
7007 create_gfm_auto_identifier(flat_text))
7008 end
7009 local normalized_attributes = normalize_attributes(attributes,
7010 auto_identifiers)
7011
7012 -- push attributes for the new section
7013 local start_output = {}
7014 local end_output = {}
7015 table.insert(start_output, self.secbegin_text)
7016 table.insert(end_output, self.secend_text)
7017
7018 table.insert(buf, self.push_attributes("heading",
7019 normalized_attributes,
7020 start_output,
7021 end_output))
7022 assert(self.attribute_type_levels["heading"] == level)
7023
7024 -- render the heading and its attributes
7025 if self.is_writing and #normalized_attributes > 0 then
7026 table.insert(buf,
7027 "\\markdownRendererHeaderAttributeContextBegin\\n")
7028 table.insert(buf, self.attributes(normalized_attributes, false))
7029 end

```

```

7030
7031 local cmd
7032 level = level + options.shiftHeadings
7033 if level <= 1 then
7034 cmd = "\\markdownRendererHeadingOne"
7035 elseif level == 2 then
7036 cmd = "\\markdownRendererHeadingTwo"
7037 elseif level == 3 then
7038 cmd = "\\markdownRendererHeadingThree"
7039 elseif level == 4 then
7040 cmd = "\\markdownRendererHeadingFour"
7041 elseif level == 5 then
7042 cmd = "\\markdownRendererHeadingFive"
7043 elseif level >= 6 then
7044 cmd = "\\markdownRendererHeadingSix"
7045 else
7046 cmd = ""
7047 end
7048 if self.is_writing then
7049 table.insert(buf, {cmd, "{", inlines, "}"})
7050 end
7051
7052 if self.is_writing and #normalized_attributes > 0 then
7053 table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd{")
7054 end
7055
7056 return buf
7057 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.

```

7058 function self.get_state()
7059 return {
7060 is_writing=self.is_writing,
7061 flatten_inlines=self.flatten_inlines,
7062 active_attributes={table.unpack(self.active_attributes)},
7063 }
7064 end

```

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

```

7065 function self.set_state(s)
7066 local previous_state = self.get_state()
7067 for key, value in pairs(s) do
7068 self[key] = value
7069 end
7070 return previous_state
7071 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`.

```

7072 function self.defer_call(f)
7073 local previous_state = self.get_state()
7074 return function(...)
7075 local state = self.set_state(previous_state)
7076 local return_value = f(...)
7077 self.set_state(state)
7078 return return_value
7079 end
7080 end
7081
7082 return self
7083 end

```

### 3.1.4 Parsers

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

```

7084 parsers = {}

```

#### 3.1.4.1 Basic Parsers

```

7085 parsers.percent = P("%")
7086 parsers.at = P("@")
7087 parsers.comma = P(",")
7088 parsers.asterisk = P("*")
7089 parsers.dash = P("-")
7090 parsers.plus = P("+")
7091 parsers.underscore = P("_")
7092 parsers.period = P(".")
7093 parsers.hash = P("#")
7094 parsers.dollar = P("$")
7095 parsers.ampersand = P("&")
7096 parsers.backtick = P("`")
7097 parsers.less = P("<")
7098 parsers.more = P(">")
7099 parsers.space = P(" ")
7100 parsers.squote = P("'")
7101 parsers.dquote = P('"')
7102 parsers.lparent = P("(")
7103 parsers.rparent = P(")")
7104 parsers.lbracket = P("[")
7105 parsers.rbracket = P("]")
7106 parsers.lbrace = P("{")

```

```

7107 parsers.rbrace = P("}")
7108 parsers.circumflex = P("^")
7109 parsers.slash = P("/")
7110 parsers.equal = P("=")
7111 parsers.colon = P(":")
7112 parsers.semicolon = P(";")
7113 parsers.exclamation = P("!")
7114 parsers.pipe = P("|")
7115 parsers.tilde = P("~")
7116 parsers.backslash = P("\\")
7117 parsers.tab = P("\t")
7118 parsers.newline = P("\n")
7119
7120 parsers.digit = R("09")
7121 parsers.hexdigit = R("09","af","AF")
7122 parsers.letter = R("AZ","az")
7123 parsers.alphanumeric = R("AZ","az","09")
7124 parsers.keyword = parsers.letter
7125 * (parsers.alphanumeric + parsers.dash)^0
7126
7127 parsers.doubleasterisks = P("**")
7128 parsers.doubleunderscores = P("__")
7129 parsers.doubletildes = P("~")
7130 parsers.fourspace = P(" ")
7131
7132 parsers.any = P(1)
7133 parsers.succeed = P(true)
7134 parsers.fail = P(false)
7135
7136 parsers.internal_punctuation = S(",:;,.?")
7137 parsers.ascii_punctuation = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")
7138

```

### 3.1.5 Unicode data

This section documents different Unicode character categories recognized by the markdown reader. The parsers for the different categories are organized in the table [parsers.unicode\\_data](#) according to the number of bytes occupied after conversion to UTF8.

All code from this section will be executed during the compilation of the Markdown package and the standard output will be stored in a file named [markdown-unicode-data.lua](#) with the precompiled parser of Unicode punctuation.

```

7139 ;(function()
7140 local pathname = assert(kpse.find_file("UnicodeData.txt"),
7141 [[Could not locate file "UnicodeData.txt"]])
7142 local file = assert(io.open(pathname, "r"),

```



```
7143 [[Could not open file "UnicodeData.txt"]])
```

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

```
7144 local categories = {"L", "N", "P", "Pc", "S", "Z"}
7145 local prefix_trees = {}
7146 for _, category in ipairs(categories) do
7147 prefix_trees[category] = {}
7148 for char_length = 1, 4 do
7149 prefix_trees[category][char_length] = {}
7150 end
7151 end
7152 for line in file:lines() do
7153 local codepoint, full_category = line:match("^(%x+);[^\s]*;(%a*)")
7154 assert(#full_category >= 1)
7155 local major_category = full_category:sub(1, 1)
7156 for _, category in ipairs({full_category, major_category}) do
7157 if prefix_trees[category] == nil then
7158 goto continue
7159 end
7160 local code = utf8.char(tonumber(codepoint, 16))
7161 local node = prefix_trees[category][#code]
7162 for i = 1, #code do
7163 local byte = code:sub(i, i)
7164 if i < #code then
7165 if node[byte] == nil then
7166 node[byte] = {}
7167 end
7168 node = node[byte]
7169 else
7170 table.insert(node, byte)
7171 end
7172 end
7173 ::continue::
7174 end
7175 end
7176 assert(file:close())
7177
```

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

```
7178 local function depth_first_search(node, path, visit, leave)
7179 visit(node, path)
7180 for label, child in pairs(node) do
7181 if type(child) == "table" then
7182 depth_first_search(child, path .. label, visit, leave)
7183 else
7184 visit(child, path)
7185 end
7186 end
7187 end
```

```

7185 end
7186 end
7187 leave(node, path)
7188 end
7189
7190 print("M.categories = {}")
7191 print("local P = lpeg.P")
7192 print("local fail = P(false)")
7193 print("-- luacheck: push no max line length")
7194 for _, category in ipairs(categories) do
7195 print("M.categories." .. category .. " = {}")
7196 for length, prefix_tree in pairs(prefix_trees[category]) do
7197 local subparsers = {}
7198 depth_first_search(prefix_tree, "", function(node, path)
7199 if type(node) == "string" then
7200 local suffix
7201 if node == "]" then
7202 suffix = "P('" .. node .. "')"
7203 else
7204 suffix = "P([[" .. node .. "]])"
7205 end
7206 if subparsers[path] ~= nil then
7207 subparsers[path] = subparsers[path] .. " + " .. suffix
7208 else
7209 subparsers[path] = suffix
7210 end
7211 end
7212 end, function(_, path)
7213 if #path > 0 then
7214 local byte = path:sub(#path, #path)
7215 local parent_path = path:sub(1, #path-1)
7216 local prefix
7217 if byte == "]" then
7218 prefix = "P('" .. byte .. "')"
7219 else
7220 prefix = "P([[" .. byte .. "]])"
7221 end
7222 local suffix
7223 if subparsers[path]:find(" %+ ") then
7224 suffix = prefix .. " * (" .. subparsers[path] .. ")"
7225 else
7226 suffix = prefix .. " * " .. subparsers[path]
7227 end
7228 if subparsers[parent_path] ~= nil then
7229 subparsers[parent_path] = subparsers[parent_path]
7230 .. " + " .. suffix
7231 else

```

```

7232 subparsers[parent_path] = suffix
7233 end
7234 else
7235 print(
7236 "M.categories." .. category .. "[" .. length .. "]" = "
7237 .. (subparsers[path] or "fail")
7238)
7239 end
7240 end)
7241 end
7242 end
7243 print("-- luacheck: pop")
7244 end)()
7245 print("return M")

```

Back in the Markdown package, we will load the precompiled parsers of Unicode categories.

```

7246 local unicode_data = require("markdown-unicode-data")
7247 if metadata.version ~= unicode_data.metadata.version then
7248 util.warning(
7249 "markdown.lua " .. metadata.version .. " used with " ..
7250 "markdown-unicode-data.lua " .. unicode_data.metadata.version .. "."
7251)
7252 end

```

Finally, we define high-level parsers for specific types of characters that are interesting for us.

```

7253 parsers.unicode = {}
7254 parsers.unicode.preceding_punctuation = parsers.fail
7255 parsers.unicode.following_punctuation = parsers.fail
7256 parsers.unicode.following_alpha = parsers.fail
7257 parsers.unicode.following_word = parsers.fail
7258 parsers.unicode.preceding_whitespace = parsers.fail
7259 parsers.unicode.following_whitespace = parsers.fail
7260 for n = 1, 4 do

```

For punctuation, accept any characters from Unicode categories P (punctuation) and S (symbol), as mandated by the CommonMark standard<sup>33</sup>.

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

```

7261 local punctuation_of_length_n
7262 = unicode_data.categories.P[n]
7263 + unicode_data.categories.S[n]
7264 parsers.unicode.preceding_punctuation
7265 = parsers.unicode.preceding_punctuation

```

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

```

7266 + B(punctuation_of_length_n)
7267 parsers.unicode.following_punctuation
7268 = parsers.unicode.following_punctuation
7269 + #punctuation_of_length_n

```

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

```

7270 local alpha_of_length_n = unicode_data.categories.L[n]
7271 parsers.unicode.following_alpha
7272 = parsers.unicode.following_alpha
7273 + alpha_of_length_n

```

For word characters, accept any characters from Unicode categories L (letter), N (number), and Pc (connector punctuation), similar to the character class ‘

```

7274 local word_of_length_n
7275 = unicode_data.categories.L[n]
7276 + unicode_data.categories.N[n]
7277 + unicode_data.categories.Pc[n]
7278 parsers.unicode.following_word
7279 = parsers.unicode.following_word
7280 + word_of_length_n

```

For space characters, accept any characters from Unicode category Z (separator), as well as the ASCII control characters 9 (horizontal tab) through 13 (carriage return), similar to the character class ‘Lua library Selene Unicode.

```

7281 local whitespace_of_length_n = unicode_data.categories.Z[n]
7282 if n == 1 then
7283 whitespace_of_length_n
7284 = whitespace_of_length_n
7285 + R("\t\r")
7286 end
7287 parsers.unicode.preceding_whitespace
7288 = parsers.unicode.preceding_whitespace
7289 + B(whitespace_of_length_n)
7290 parsers.unicode.following_whitespace
7291 = parsers.unicode.following_whitespace
7292 + #whitespace_of_length_n
7293 end
7294
7295 parsers.escapable = parsers.ascii_punctuation
7296 parsers.anyescaped = parsers.backslash / ""
7297 * parsers.escapable
7298 + parsers.any
7299
7300 parsers.spacechar = S("\t ")
7301 parsers.spacing = S(" \n\r\t")
7302 parsers.nonspacechar = parsers.any - parsers.spacing
7303 parsers.optionalspace = parsers.spacechar^0

```

```

7304
7305 parsers.normalchar = parsers.any - (V("SpecialChar")
7306 + parsers.spacing)
7307 parsers.eof = -parsers.any
7308 parsers.nonindentspace = parsers.space-3 * - parsers.spacechar
7309 parsers.indent = parsers.space-3 * parsers.tab
7310 + parsers.fourspace / ""
7311 parsers.linechar = P(1 - parsers.newline)
7312
7313 parsers.blankline = parsers.optionalspace
7314 * parsers.newline / "\n"
7315 parsers.blanklines = parsers.blankline0
7316 parsers.skipblanklines = (parsers.optionalspace
7317 * parsers.newline)0
7318 parsers.indentedline = parsers.indent / ""
7319 * C(parsers.linechar1
7320 * parsers.newline-1)
7321 parsers.optionallyindentedline = parsers.indent-1 / ""
7322 * C(parsers.linechar1
7323 * parsers.newline-1)
7324 parsers.sp = parsers.spacing0
7325 parsers.spnl = parsers.optionalspace
7326 * (parsers.newline
7327 * parsers.optionalspace)-1
7328 parsers.line = parsers.linechar0 * parsers.newline
7329 parsers.nonemptyline = parsers.line - parsers.blankline

```

### 3.1.5.1 Parsers Used for Indentation

```

7330
7331 parsers.leader = parsers.space-3
7332

```

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

```

7333 local function has_trail(indent_table)
7334 return indent_table ~= nil and
7335 indent_table.trail ~= nil and
7336 next(indent_table.trail) ~= nil
7337 end
7338

```

Check if indent table `indent_table` has any indents.

```

7339 local function has_indents(indent_table)
7340 return indent_table ~= nil and
7341 indent_table.indents ~= nil and
7342 next(indent_table.indents) ~= nil
7343 end
7344

```

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

```
7345 local function add_trail(indent_table, trail_info)
7346 indent_table.trail = trail_info
7347 return indent_table
7348 end
7349
```

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

```
7350 local function remove_trail(indent_table)
7351 indent_table.trail = nil
7352 return indent_table
7353 end
7354
```

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

```
7355 local function update_indent_table(indent_table, new_indent, add)
7356 indent_table = remove_trail(indent_table)
7357
7358 if not has_indents(indent_table) then
7359 indent_table.indents = {}
7360 end
7361
7362 if add then
7363 indent_table.indents[#indent_table.indents + 1] = new_indent
7364 else
7365 if indent_table.indents[#indent_table.indents].name
7366 == new_indent.name then
7367 indent_table.indents[#indent_table.indents] = nil
7368 end
7369 end
7370 end
7371
7372 return indent_table
7373 end
7374
```

Remove an indent by its name `name`.

```
7375 local function remove_indent(name)
7376 local remove_indent_level =
7377 function(s, i, indent_table) -- luacheck: ignore s i
7378 indent_table = update_indent_table(indent_table, {name=name},
7379 false)
7380 return true, indent_table
7381 end
7382
7383 return Cg(Cmt(Cb("indent_info"), remove_indent_level), "indent_info")
7384 end
7385
```

Process the spacing of a string of spaces and tabs `spacing` with preceding indent width from the start of the line `indent` and strip up to `left_strip_length` spaces. Return the remainder `remainder` and whether there is enough spaces to produce a code `is_code`. Return how many spaces were stripped, as well as if the minimum was met `is_minimum` and what remainder it left `minimum_remainder`.

```

7386 local function process_starter_spacing(indent, spacing,
7387 minimum, left_strip_length)
7388 left_strip_length = left_strip_length or 0
7389
7390 local count = 0
7391 local tab_value = 4 - (indent) % 4
7392
7393 local code_started, minimum_found = false, false
7394 local code_start, minimum_remainder = "", ""
7395
7396 local left_total_stripped = 0
7397 local full_remainder = ""
7398
7399 if spacing ~= nil then
7400 for i = 1, #spacing do
7401 local character = spacing:sub(i, i)
7402
7403 if character == "\t" then
7404 count = count + tab_value
7405 tab_value = 4
7406 elseif character == " " then
7407 count = count + 1
7408 tab_value = 4 - (1 - tab_value) % 4
7409 end
7410
7411 if (left_strip_length ~= 0) then
7412 local possible_to_strip = math.min(count, left_strip_length)
7413 count = count - possible_to_strip
7414 left_strip_length = left_strip_length - possible_to_strip
7415 left_total_stripped = left_total_stripped + possible_to_strip
7416 else
7417 full_remainder = full_remainder .. character
7418 end
7419
7420 if (minimum_found) then
7421 minimum_remainder = minimum_remainder .. character
7422 elseif (count >= minimum) then
7423 minimum_found = true
7424 minimum_remainder = minimum_remainder
7425 .. string.rep(" ", count - minimum)
7426 end

```

```

7427
7428 if (code_started) then
7429 code_start = code_start .. character
7430 elseif (count >= minimum + 4) then
7431 code_started = true
7432 code_start = code_start
7433 .. string.rep(" ", count - (minimum + 4))
7434 end
7435 end
7436 end
7437
7438 local remainder
7439 if (code_started) then
7440 remainder = code_start
7441 else
7442 remainder = string.rep(" ", count - minimum)
7443 end
7444
7445 local is_minimum = count >= minimum
7446 return {
7447 is_code = code_started,
7448 remainder = remainder,
7449 left_total_stripped = left_total_stripped,
7450 is_minimum = is_minimum,
7451 minimum_remainder = minimum_remainder,
7452 total_length = count,
7453 full_remainder = full_remainder
7454 }
7455 end
7456

```

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

```

7457 local function count_indent_tab_level(indent_table)
7458 local count = 0
7459 if not has_indents(indent_table) then
7460 return count
7461 end
7462
7463 for i=1, #indent_table.indents do
7464 count = count + indent_table.indents[i].length
7465 end
7466 return count
7467 end
7468

```

Count the total width of a delimiter `delimiter`.

```

7469 local function total_delimiter_length(delimiter)
7470 local count = 0

```



```

7471 if type(delimiter) == "string" then return #delimiter end
7472 for _, value in pairs(delimiter) do
7473 count = count + total_delimiter_length(value)
7474 end
7475 return count
7476 end
7477

```

Process the container starter `starter` of a type `indent_type`. Adjust the width of the indent if the delimiter is followed only by whitespaces `is_blank`.

```

7478 local function process_starter_indent(_, _, indent_table, starter,
7479 is_blank, indent_type, breakable)
7480 local last_trail = starter[1]
7481 local delimiter = starter[2]
7482 local raw_new_trail = starter[3]
7483
7484 if indent_type == "bq" and not breakable then
7485 indent_table.ignore_blockquote_blank = true
7486 end
7487
7488 if has_trail(indent_table) then
7489 local trail = indent_table.trail
7490 if trail.is_code then
7491 return false
7492 end
7493 last_trail = trail.remainder
7494 else
7495 local sp = process_starter_spacing(0, last_trail, 0, 0)
7496
7497 if sp.is_code then
7498 return false
7499 end
7500 last_trail = sp.remainder
7501 end
7502
7503 local preceding_indentation = count_indent_tab_level(indent_table) % 4
7504 local last_trail_length = #last_trail
7505 local delimiter_length = total_delimiter_length(delimiter)
7506
7507 local total_indent_level = preceding_indentation + last_trail_length
7508 + delimiter_length
7509
7510 local sp = {}
7511 if not is_blank then
7512 sp = process_starter_spacing(total_indent_level, raw_new_trail,
7513 0, 1)
7514 end

```

```

7515
7516 local del_trail_length = sp.left_total_stripped
7517 if is_blank then
7518 del_trail_length = 1
7519 elseif not sp.is_code then
7520 del_trail_length = del_trail_length + #sp.remainder
7521 end
7522
7523 local indent_length = last_trail_length + delimiter_length
7524 + del_trail_length
7525 local new_indent_info = {name=indent_type, length=indent_length}
7526
7527 indent_table = update_indent_table(indent_table, new_indent_info,
7528 true)
7529 indent_table = add_trail(indent_table,
7530 {is_code=sp.is_code,
7531 remainder=sp.remainder,
7532 total_length=sp.total_length,
7533 full_remainder=sp.full_remainder})
7534
7535 return true, indent_table
7536 end
7537

```

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

```

7538 local function decode_pattern(name)
7539 local delimiter = parsers.succeed
7540 if name == "bq" then
7541 delimiter = parsers.more
7542 end
7543
7544 return C(parsers.optionalspace) * C(delimiter)
7545 * C(parsers.optionalspace) * Cp()
7546 end
7547

```

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

```

7548 local function left_blank_starter(indent_table)
7549 local blank_starter_index
7550
7551 if not has_indents(indent_table) then
7552 return
7553 end
7554
7555 for i = #indent_table.indents,1,-1 do
7556 local value = indent_table.indents[i]
7557 if value.name == "li" then

```

```

7558 blank_starter_index = i
7559 else
7560 break
7561 end
7562 end
7563
7564 return blank_starter_index
7565 end
7566

```

Apply the patterns decoded from the indents of the indent table `indent_table` iteratively starting at position `index` of the string `s`. If the `is_optional` mode is selected, match as many patterns as possible, else match all or fail. With the option `is_blank`, the parsing behaves as optional after the position of a blank-only indent has been surpassed.

```

7567 local function traverse_indent(s, i, indent_table, is_optional,
7568 is_blank, current_line_indents)
7569 local new_index = i
7570
7571 local preceding_indentation = 0
7572 local current_trail = {}
7573
7574 local blank_starter = left_blank_starter(indent_table)
7575
7576 if current_line_indents == nil then
7577 current_line_indents = {}
7578 end
7579
7580 for index = 1, #indent_table.indents do
7581 local value = indent_table.indents[index]
7582 local pattern = decode_pattern(value.name)
7583
7584 -- match decoded pattern
7585 local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
7586 if new_indent_info == nil then
7587 local blankline_end = lpeg.match(
7588 Ct(parsers.blankline * Cg(Cp(), "pos")), s, new_index)
7589 if is_optional or not indent_table.ignore_blockquote_blank
7590 or not blankline_end then
7591 return is_optional, new_index, current_trail,
7592 current_line_indents
7593 end
7594
7595 return traverse_indent(s, tonumber(blankline_end.pos),
7596 indent_table, is_optional, is_blank,
7597 current_line_indents)
7598 end
7599 end
7600

```

```

7599
7600 local raw_last_trail = new_indent_info[1]
7601 local delimiter = new_indent_info[2]
7602 local raw_new_trail = new_indent_info[3]
7603 local next_index = new_indent_info[4]
7604
7605 local space_only = delimiter == ""
7606
7607 -- check previous trail
7608 if not space_only and next(current_trail) == nil then
7609 local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
7610 current_trail = {is_code=sp.is_code, remainder=sp.remainder,
7611 total_length=sp.total_length,
7612 full_remainder=sp.full_remainder}
7613 end
7614
7615 if next(current_trail) ~= nil then
7616 if not space_only and current_trail.is_code then
7617 return is_optional, new_index, current_trail,
7618 current_line_indents
7619 end
7620 if current_trail.internal_remainder ~= nil then
7621 raw_last_trail = current_trail.internal_remainder
7622 end
7623 end
7624
7625 local raw_last_trail_length = 0
7626 local delimiter_length = 0
7627
7628 if not space_only then
7629 delimiter_length = #delimiter
7630 raw_last_trail_length = #raw_last_trail
7631 end
7632
7633 local total_indent_level = preceding_indentation
7634 + raw_last_trail_length + delimiter_length
7635
7636 local spacing_to_process
7637 local minimum = 0
7638 local left_strip_length = 0
7639
7640 if not space_only then
7641 spacing_to_process = raw_new_trail
7642 left_strip_length = 1
7643 else
7644 spacing_to_process = raw_last_trail
7645 minimum = value.length

```

```

7646 end
7647
7648 local sp = process_starter_spacing(total_indent_level,
7649 spacing_to_process, minimum,
7650 left_strip_length)
7651
7652 if space_only and not sp.is_minimum then
7653 return is_optional or (is_blank and blank_starter <= index),
7654 new_index, current_trail, current_line_indents
7655 end
7656
7657 local indent_length = raw_last_trail_length + delimiter_length
7658 + sp.left_total_stripped
7659
7660 -- update info for the next pattern
7661 if not space_only then
7662 preceding_indentation = preceding_indentation + indent_length
7663 else
7664 preceding_indentation = preceding_indentation + value.length
7665 end
7666
7667 current_trail = {is_code=sp.is_code, remainder=sp.remainder,
7668 internal_remainder=sp.minimum_remainder,
7669 total_length=sp.total_length,
7670 full_remainder=sp.full_remainder}
7671
7672 current_line_indents[#current_line_indents + 1] = new_indent_info
7673 new_index = next_index
7674 end
7675
7676 return true, new_index, current_trail, current_line_indents
7677 end
7678

```

Check if a code trail is expected.

```

7679 local function check_trail(expect_code, is_code)
7680 return (expect_code and is_code) or (not expect_code and not is_code)
7681 end
7682

```

Check if the current trail of the [indent\\_table](#) would produce code if it is expected [expect\\_code](#) or it would not if it is not. If there is no trail, process and check the current spacing [spacing](#).

```

7683 local check_trail_joined =
7684 function(s, i, indent_table, -- luacheck: ignore s i
7685 spacing, expect_code, omit_remainder)
7686 local is_code
7687 local remainder

```

```

7688
7689 if has_trail(indent_table) then
7690 local trail = indent_table.trail
7691 is_code = trail.is_code
7692 if is_code then
7693 remainder = trail.remainder
7694 else
7695 remainder = trail.full_remainder
7696 end
7697 else
7698 local sp = process_starter_spacing(0, spacing, 0, 0)
7699 is_code = sp.is_code
7700 if is_code then
7701 remainder = sp.remainder
7702 else
7703 remainder = sp.full_remainder
7704 end
7705 end
7706
7707 local result = check_trail(expect_code, is_code)
7708 if omit_remainder then
7709 return result
7710 end
7711 return result, remainder
7712 end
7713

```

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

```

7714 local check_trail_length =
7715 function(s, i, indent_table, -- luacheck: ignore s i
7716 spacing, min, max)
7717 local trail
7718
7719 if has_trail(indent_table) then
7720 trail = indent_table.trail
7721 else
7722 trail = process_starter_spacing(0, spacing, 0, 0)
7723 end
7724
7725 local total_length = trail.total_length
7726 if total_length == nil then
7727 return false
7728 end
7729
7730 return min <= total_length and total_length <= max
7731 end
7732

```

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

```
7733 local function check_continuation_indentation(s, i, indent_table,
7734 is_optional, is_blank)
7735 if not has_indents(indent_table) then
7736 return true
7737 end
7738
7739 local passes, new_index, current_trail, current_line_indents =
7740 traverse_indent(s, i, indent_table, is_optional, is_blank)
7741
7742 if passes then
7743 indent_table.current_line_indents = current_line_indents
7744 indent_table = add_trail(indent_table, current_trail)
7745 return new_index, indent_table
7746 end
7747 return false
7748 end
7749
```

Get name of the last indent from the `indent_table`.

```
7750 local function get_last_indent_name(indent_table)
7751 if has_indents(indent_table) then
7752 return indent_table.indents[#indent_table.indents].name
7753 end
7754 end
7755
```

Remove the remainder altogether if the last indent from the `indent_table` is blank-only.

```
7756 local function remove_remainder_if_blank(indent_table, remainder)
7757 if get_last_indent_name(indent_table) == "li" then
7758 return ""
7759 end
7760 return remainder
7761 end
7762
```

Take the trail `trail` or create a new one from `spacing` and compare it with the expected `trail_type`. On success return the index `i` and the remainder of the trail.

```
7763 local check_trail_type =
7764 function(s, i, -- luacheck: ignore s i
7765 trail, spacing, trail_type)
7766 if trail == nil then
7767 trail = process_starter_spacing(0, spacing, 0, 0)
7768 end
7769
7770 if trail_type == "non-code" then
```

```

7771 return check_trail(false, trail.is_code)
7772 end
7773 if trail_type == "code" then
7774 return check_trail(true, trail.is_code)
7775 end
7776 if trail_type == "full-code" then
7777 if (trail.is_code) then
7778 return i, trail.remainder
7779 end
7780 return i, ""
7781 end
7782 if trail_type == "full-any" then
7783 return i, trail.internal_remainder
7784 end
7785 end
7786

```

Stores or restores an `is_freezing` trail from indent table `indent_table`.

```

7787 local trail_freezing =
7788 function(s, i, -- luacheck: ignore s i
7789 indent_table, is_freezing)
7790 if is_freezing then
7791 if indent_table.is_trail_frozen then
7792 indent_table.trail = indent_table.frozen_trail
7793 else
7794 indent_table.frozen_trail = indent_table.trail
7795 indent_table.is_trail_frozen = true
7796 end
7797 else
7798 indent_table.frozen_trail = nil
7799 indent_table.is_trail_frozen = false
7800 end
7801 return true, indent_table
7802 end
7803

```

Check the indentation of the continuation line, optionally with the mode `is_optional` selected. Check blank line specifically with `is_blank`. Additionally, also directly check the new trail with a type `trail_type`.

```

7804 local check_continuation_indentation_and_trail =
7805 function(s, i, indent_table, is_optional, is_blank, trail_type,
7806 reset_rem, omit_remainder)
7807 if not has_indents(indent_table) then
7808 local spacing, new_index = lpeg.match(C(parsers.spacechar~0)
7809 * Cp(), s, i)
7810 local result, remainder = check_trail_type(s, i,
7811 indent_table.trail, spacing, trail_type)
7812 if remainder == nil then

```



```

7813 if result then
7814 return new_index
7815 end
7816 return false
7817 end
7818 if result then
7819 return new_index, remainder
7820 end
7821 return false
7822 end
7823
7824 local passes, new_index, current_trail = traverse_indent(s, i,
7825 indent_table, is_optional, is_blank)
7826
7827 if passes then
7828 local spacing
7829 if current_trail == nil then
7830 local newer_spacing, newer_index = lpeg.match(
7831 C(parsers.spacechar^0) * Cp(), s, i)
7832 current_trail = process_starter_spacing(0, newer_spacing, 0, 0)
7833 new_index = newer_index
7834 spacing = newer_spacing
7835 else
7836 spacing = current_trail.remainder
7837 end
7838 local result, remainder = check_trail_type(s, new_index,
7839 current_trail, spacing, trail_type)
7840 if remainder == nil or omit_remainder then
7841 if result then
7842 return new_index
7843 end
7844 return false
7845 end
7846
7847 if is_blank and reset_rem then
7848 remainder = remove_remainder_if_blank(indent_table, remainder)
7849 end
7850 if result then
7851 return new_index, remainder
7852 end
7853 return false
7854 end
7855 return false
7856 end
7857

```

The following patterns check whitespace indentation at the start of a block.

```

7858 parsers.check_trail = Cmt(Cb("indent_info") * C(parsers.spacechar^0)

```

```

7859 * Cc(false), check_trail_joined)
7860
7861 parsers.check_trail_no_rem = Cmt(Cb("indent_info")
7862 * C(parsers.spacechar^0) * Cc(false)
7863 * Cc(true), check_trail_joined)
7864
7865 parsers.check_code_trail = Cmt(Cb("indent_info")
7866 * C(parsers.spacechar^0)
7867 * Cc(true), check_trail_joined)
7868
7869 parsers.check_trail_length_range = function(min, max)
7870 return Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(min)
7871 * Cc(max), check_trail_length)
7872 end
7873
7874 parsers.check_trail_length = function(n)
7875 return parsers.check_trail_length_range(n, n)
7876 end
7877

```

The following patterns handle trail backup, to prevent a failing pattern to modify it before passing it to the next.

```

7878 parsers.freeze_trail = Cg(Cmt(Cb("indent_info")
7879 * Cc(true), trail_freezing), "indent_info")
7880
7881 parsers.unfreeze_trail = Cg(Cmt(Cb("indent_info") * Cc(false),
7882 trail_freezing), "indent_info")
7883

```

The following patterns check indentation in continuation lines as defined by the container start.

```

7884 parsers.check_minimal_indent = Cmt(Cb("indent_info") * Cc(false),
7885 check_continuation_indentation)
7886
7887 parsers.check_optional_indent = Cmt(Cb("indent_info") * Cc(true),
7888 check_continuation_indentation)
7889
7890 parsers.check_minimal_blank_indent
7891 = Cmt(Cb("indent_info") * Cc(false)
7892 * Cc(true)
7893 , check_continuation_indentation)
7894

```

The following patterns check indentation in continuation lines as defined by the container start. Additionally the subsequent trail is also directly checked.

```

7895
7896 parsers.check_minimal_indent_and_trail =
7897 Cmt(Cb("indent_info")

```

```

7898 * Cc(false) * Cc(false) * Cc("non-code") * Cc(true)
7899 , check_continuation_indentation_and_trail)
7900
7901 parsers.check_minimal_indent_and_code_trail =
7902 Cmt(Cb("indent_info")
7903 * Cc(false) * Cc(false) * Cc("code") * Cc(false)
7904 , check_continuation_indentation_and_trail)
7905
7906 parsers.check_minimal_blank_indent_and_full_code_trail =
7907 Cmt(Cb("indent_info")
7908 * Cc(false) * Cc(true) * Cc("full-code") * Cc(true)
7909 , check_continuation_indentation_and_trail)
7910
7911 parsers.check_minimal_indent_and_any_trail =
7912 Cmt(Cb("indent_info")
7913 * Cc(false) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
7914 , check_continuation_indentation_and_trail)
7915
7916 parsers.check_minimal_blank_indent_and_any_trail =
7917 Cmt(Cb("indent_info")
7918 * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
7919 , check_continuation_indentation_and_trail)
7920
7921 parsers.check_minimal_blank_indent_and_any_trail_no_rem =
7922 Cmt(Cb("indent_info")
7923 * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(true)
7924 , check_continuation_indentation_and_trail)
7925
7926 parsers.check_optional_indent_and_any_trail =
7927 Cmt(Cb("indent_info")
7928 * Cc(true) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
7929 , check_continuation_indentation_and_trail)
7930
7931 parsers.check_optional_blank_indent_and_any_trail =
7932 Cmt(Cb("indent_info")
7933 * Cc(true) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
7934 , check_continuation_indentation_and_trail)
7935

```

The following patterns specify behaviour around newlines.

```

7936
7937 parsers.spnlc_noexc = parsers.optionalspace
7938 * (parsers.newline
7939 * parsers.check_minimal_indent_and_any_trail)^-1
7940
7941 parsers.spnlc = parsers.optionalspace
7942 * (V("EndlineNoSub"))^-1
7943

```

```

7944 parsers.spnlc_sep = parsers.optionalspace * V("EndlineNoSub")
7945 + parsers.spacechar^1
7946
7947 parsers.only_blank = parsers.spacechar^0
7948 * (parsers.newline + parsers.eof)
7949

```

The `parsers.commented_line^1` parser recognizes the regular language of T<sub>E</sub>X comments, see an equivalent finite automaton in Figure 8.

```

7950 parsers.commented_line_letter = parsers.linechar
7951 + parsers.newline
7952 - parsers.backslash
7953 - parsers.percent
7954 parsers.commented_line = Cg(Cc(""), "backslashes")
7955 * ((#(parsers.commented_line_letter
7956 - parsers.newline)
7957 * Cb("backslashes")
7958 * Cs(parsers.commented_line_letter
7959 - parsers.newline)^1 -- initial
7960 * Cg(Cc(""), "backslashes"))
7961 + #(parsers.backslash
7962 * (parsers.backslash + parsers.newline))
7963 * Cg((parsers.backslash -- even backslash
7964 * (parsers.backslash
7965 + #parsers.newline))^1, "backslashes")
7966 + (parsers.backslash
7967 * (#parsers.percent
7968 * Cb("backslashes")
7969 / function(backslashes)
7970 return string.rep("\\", #backslashes / 2)
7971 end
7972 * C(parsers.percent)
7973 + #parsers.commented_line_letter
7974 * Cb("backslashes")
7975 * Cc("\\")
7976 * C(parsers.commented_line_letter))
7977 * Cg(Cc(""), "backslashes"))^0
7978 * (#parsers.percent
7979 * Cb("backslashes")
7980 / function(backslashes)
7981 return string.rep("\\", #backslashes / 2)
7982 end
7983 * ((parsers.percent -- comment
7984 * parsers.line
7985 * #parsers.blankline) -- blank line
7986 / "\\n"
7987 + parsers.percent -- comment

```



```

7988 * parsers.line
7989 * parsers.optionalspace) -- leading spaces
7990 + #(parsers.newline)
7991 * Cb("backslashes")
7992 * C(parsers.newline))
7993
7994 parsers.chunk = parsers.line * (parsers.optionallyindentedline
7995 - parsers.blankline)^0
7996
7997 parsers.attribute_key_char = parsers.alphanumeric + S("-_:.")
7998 parsers.attribute_raw_char = parsers.alphanumeric + S("-_")
7999 parsers.attribute_key = (parsers.attribute_key_char
8000 - parsers.dash - parsers.digit)
8001 * parsers.attribute_key_char^0
8002 parsers.attribute_value = ((parsers.dquote / "\"")
8003 * (parsers.anyescaped - parsers.dquote)^0
8004 * (parsers.dquote / "\""))
8005 + ((parsers.squote / "\"")
8006 * (parsers.anyescaped - parsers.squote)^0
8007 * (parsers.squote / "\""))
8008 + (parsers.anyescaped
8009 - parsers.dquote
8010 - parsers.rbrace
8011 - parsers.space)^0
8012 parsers.attribute_identfier = parsers.attribute_key_char^1
8013 parsers.attribute_classname = parsers.letter
8014 * parsers.attribute_key_char^0
8015 parsers.attribute_raw = parsers.attribute_raw_char^1
8016
8017 parsers.attribute = (parsers.dash * Cc(".unnumbered"))
8018 + C(parsers.hash
8019 * parsers.attribute_identfier)
8020 + C(parsers.period
8021 * parsers.attribute_classname)
8022 + Cs(parsers.attribute_key
8023 * parsers.optionalspace
8024 * parsers.equal
8025 * parsers.optionalspace
8026 * parsers.attribute_value)
8027 parsers.attributes = parsers.lbrace
8028 * parsers.optionalspace
8029 * parsers.attribute
8030 * (parsers.spacechar^1
8031 * parsers.attribute)^0
8032 * parsers.optionalspace
8033 * parsers.rbrace
8034

```

```

8035 parsers.raw_attribute = parsers.lbrace
8036 * parsers.optionalspace
8037 * parsers.equal
8038 * C(parsers.attribute_raw)
8039 * parsers.optionalspace
8040 * parsers.rbrace
8041
8042 -- block followed by 0 or more optionally
8043 -- indented blocks with first line indented.
8044 parsers.indented_blocks = function(bl)
8045 return Cs(bl
8046 * (parsers.blankline^1
8047 * parsers.indent
8048 * -parsers.blankline
8049 * bl)^0
8050 * (parsers.blankline^1 + parsers.eof))
8051 end

```

### 3.1.5.2 Parsers Used for HTML Entities

```

8052 local function repeat_between(pattern, min, max)
8053 return -pattern^(max + 1) * pattern^min
8054 end
8055
8056 parsers.hexentity = parsers.ampersand * parsers.hash * C(S("Xx"))
8057 * C(repeat_between(parsers.hexdigit, 1, 6))
8058 * parsers.semicolon
8059 parsers.decentity = parsers.ampersand * parsers.hash
8060 * C(repeat_between(parsers.digit, 1, 7))
8061 * parsers.semicolon
8062 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
8063 * parsers.semicolon
8064
8065 parsers.html_entities
8066 = parsers.hexentity / entities.hex_entity_with_x_char
8067 + parsers.decentity / entities.dec_entity
8068 + parsers.tagentity / entities.char_entity

```

### 3.1.5.3 Parsers Used for Markdown Lists

```

8069 parsers.bullet = function(bullet_char, interrupting)
8070 local allowed_end
8071 if interrupting then
8072 allowed_end = C(parsers.spacechar^1) * #parsers.linechar
8073 else
8074 allowed_end = C(parsers.spacechar^1)
8075 + #(parsers.newline + parsers.eof)
8076 end

```

```

8077 return parsers.check_trail
8078 * Ct(C(bullet_char) * Cc(""))
8079 * allowed_end
8080 end
8081
8082 local function tickbox(interior)
8083 return parsers.optionalspace * parsers.lbracket
8084 * interior * parsers.rbracket * parsers.spacechar^1
8085 end
8086
8087 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
8088 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
8089 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
8090

```

#### 3.1.5.4 Parsers Used for Markdown Code Spans

```

8091 parsers.openticks = Cg(parsers.backtick^1, "ticks")
8092
8093 local function captures_equal_length(_,i,a,b)
8094 return #a == #b and i
8095 end
8096
8097 parsers.closeticks = Cmt(C(parsers.backtick^1)
8098 * Cb("ticks"), captures_equal_length)
8099
8100 parsers.intickschar = (parsers.any - S("\n\r`"))
8101 + V("NoSoftLineBreakEndline")
8102 + (parsers.backtick^1 - parsers.closeticks)
8103
8104 local function process_inticks(s)
8105 s = s:gsub("\n", " ")
8106 s = s:gsub("^ (.*) $", "%1")
8107 return s
8108 end
8109
8110 parsers.inticks = parsers.openticks
8111 * C(parsers.space^0)
8112 * parsers.closeticks
8113 + parsers.openticks
8114 * Cs(Cs(parsers.intickschar^0) / process_inticks)
8115 * parsers.closeticks
8116

```

#### 3.1.5.5 Parsers Used for HTML

```

8117 -- case-insensitive match (we assume s is lowercase)
8118 -- must be single byte encoding

```



```

8119 parsers.keyword_exact = function(s)
8120 local parser = P(0)
8121 for i=1,#s do
8122 local c = s:sub(i,i)
8123 local m = c .. upper(c)
8124 parser = parser * S(m)
8125 end
8126 return parser
8127 end
8128
8129 parsers.special_block_keyword =
8130 parsers.keyword_exact("pre") +
8131 parsers.keyword_exact("script") +
8132 parsers.keyword_exact("style") +
8133 parsers.keyword_exact("textarea")
8134
8135 parsers.block_keyword =
8136 parsers.keyword_exact("address") +
8137 parsers.keyword_exact("article") +
8138 parsers.keyword_exact("aside") +
8139 parsers.keyword_exact("base") +
8140 parsers.keyword_exact("basefont") +
8141 parsers.keyword_exact("blockquote") +
8142 parsers.keyword_exact("body") +
8143 parsers.keyword_exact("caption") +
8144 parsers.keyword_exact("center") +
8145 parsers.keyword_exact("col") +
8146 parsers.keyword_exact("colgroup") +
8147 parsers.keyword_exact("dd") +
8148 parsers.keyword_exact("details") +
8149 parsers.keyword_exact("dialog") +
8150 parsers.keyword_exact("dir") +
8151 parsers.keyword_exact("div") +
8152 parsers.keyword_exact("dl") +
8153 parsers.keyword_exact("dt") +
8154 parsers.keyword_exact("fieldset") +
8155 parsers.keyword_exact("figcaption") +
8156 parsers.keyword_exact("figure") +
8157 parsers.keyword_exact("footer") +
8158 parsers.keyword_exact("form") +
8159 parsers.keyword_exact("frame") +
8160 parsers.keyword_exact("frameset") +
8161 parsers.keyword_exact("h1") +
8162 parsers.keyword_exact("h2") +
8163 parsers.keyword_exact("h3") +
8164 parsers.keyword_exact("h4") +
8165 parsers.keyword_exact("h5") +

```

```

8166 parsers.keyword_exact("h6") +
8167 parsers.keyword_exact("head") +
8168 parsers.keyword_exact("header") +
8169 parsers.keyword_exact("hr") +
8170 parsers.keyword_exact("html") +
8171 parsers.keyword_exact("iframe") +
8172 parsers.keyword_exact("legend") +
8173 parsers.keyword_exact("li") +
8174 parsers.keyword_exact("link") +
8175 parsers.keyword_exact("main") +
8176 parsers.keyword_exact("menu") +
8177 parsers.keyword_exact("menuitem") +
8178 parsers.keyword_exact("nav") +
8179 parsers.keyword_exact("noframes") +
8180 parsers.keyword_exact("ol") +
8181 parsers.keyword_exact("optgroup") +
8182 parsers.keyword_exact("option") +
8183 parsers.keyword_exact("p") +
8184 parsers.keyword_exact("param") +
8185 parsers.keyword_exact("section") +
8186 parsers.keyword_exact("source") +
8187 parsers.keyword_exact("summary") +
8188 parsers.keyword_exact("table") +
8189 parsers.keyword_exact("tbody") +
8190 parsers.keyword_exact("td") +
8191 parsers.keyword_exact("tfoot") +
8192 parsers.keyword_exact("th") +
8193 parsers.keyword_exact("thead") +
8194 parsers.keyword_exact("title") +
8195 parsers.keyword_exact("tr") +
8196 parsers.keyword_exact("track") +
8197 parsers.keyword_exact("ul")
8198
8199 -- end conditions
8200 parsers.html_blankline_end_condition
8201 = parsers.linechar^0
8202 * (parsers.newline
8203 * (parsers.check_minimal_blank_indent_and_any_trail
8204 * #parsers.blankline
8205 + parsers.check_minimal_indent_and_any_trail)
8206 * parsers.linechar^1)^0
8207 * (parsers.newline^-1 / "")
8208
8209 local function remove_trailing_blank_lines(s)
8210 return s:gsub("[\n\r]+%s*$", "")
8211 end
8212

```

```

8213 parsers.html_until_end = function(end_marker)
8214 return Cs(Cs((parsers.newline
8215 * (parsers.check_minimal_blank_indent_and_any_trail
8216 * #parsers.blankline
8217 + parsers.check_minimal_indent_and_any_trail)
8218 + parsers.linechar - end_marker)^0
8219 * parsers.linechar^0 * parsers.newline~1)
8220 / remove_trailing_blank_lines)
8221 end
8222
8223 -- attributes
8224 parsers.html_attribute_spacing = parsers.optionalspace
8225 * V("NoSoftLineBreakEndline")
8226 * parsers.optionalspace
8227 + parsers.spacechar^1
8228
8229 parsers.html_attribute_name = (parsers.letter
8230 + parsers.colon
8231 + parsers.underscore)
8232 * (parsers.alphanumeric
8233 + parsers.colon
8234 + parsers.underscore
8235 + parsers.period
8236 + parsers.dash)^0
8237
8238 parsers.html_attribute_value = parsers.squote
8239 * (parsers.linechar - parsers.squote)^0
8240 * parsers.squote
8241 + parsers.dquote
8242 * (parsers.linechar - parsers.dquote)^0
8243 * parsers.dquote
8244 + (parsers.any
8245 - parsers.spacechar
8246 - parsers.newline
8247 - parsers.dquote
8248 - parsers.squote
8249 - parsers.backtick
8250 - parsers.equal
8251 - parsers.less
8252 - parsers.more)^1
8253
8254 parsers.html_inline_attribute_value = parsers.squote
8255 * (V("NoSoftLineBreakEndline")
8256 + parsers.any
8257 - parsers.blankline^2
8258 - parsers.squote)^0
8259 * parsers.squote

```

```

8260 + parsers.dquote
8261 * (V("NoSoftLineBreakEndline")
8262 + parsers.any
8263 - parsers.blankline^2
8264 - parsers.dquote)^0
8265 * parsers.dquote
8266 + (parsers.any
8267 - parsers.spacechar
8268 - parsers.newline
8269 - parsers.dquote
8270 - parsers.squote
8271 - parsers.backtick
8272 - parsers.equal
8273 - parsers.less
8274 - parsers.more)^1
8275
8276 parsers.html_attribute_value_specification
8277 = parsers.optionalspace
8278 * parsers.equal
8279 * parsers.optionalspace
8280 * parsers.html_attribute_value
8281
8282 parsers.html_spnl = parsers.optionalspace
8283 * (V("NoSoftLineBreakEndline")
8284 * parsers.optionalspace)^-1
8285
8286 parsers.html_inline_attribute_value_specification
8287 = parsers.html_spnl
8288 * parsers.equal
8289 * parsers.html_spnl
8290 * parsers.html_inline_attribute_value
8291
8292 parsers.html_attribute
8293 = parsers.html_attribute_spacing
8294 * parsers.html_attribute_name
8295 * parsers.html_inline_attribute_value_specification^-1
8296
8297 parsers.html_non_newline_attribute
8298 = parsers.spacechar^1
8299 * parsers.html_attribute_name
8300 * parsers.html_attribute_value_specification^-1
8301
8302 parsers.nested_breaking_blank = parsers.newline
8303 * parsers.check_minimal_blank_indent
8304 * parsers.blankline
8305
8306 parsers.html_comment_start = P("<!--")

```

```

8307
8308 parsers.html_comment_end = P("-->")
8309
8310 parsers.html_comment
8311 = Cs(parsers.html_comment_start
8312 * parsers.html_until_end(parsers.html_comment_end))
8313
8314 parsers.html_inline_comment = (parsers.html_comment_start / "")
8315 * -P(">") * -P("->")
8316 * Cs((V("NoSoftLineBreakEndline")
8317 + parsers.any
8318 - parsers.nested_breaking_blank
8319 - parsers.html_comment_end)^0)
8320 * (parsers.html_comment_end / "")
8321
8322 parsers.html_cdatasection_start = P("<![CDATA[")
8323
8324 parsers.html_cdatasection_end = P("]]>")
8325
8326 parsers.html_cdatasection
8327 = Cs(parsers.html_cdatasection_start
8328 * parsers.html_until_end(parsers.html_cdatasection_end))
8329
8330 parsers.html_inline_cdatasection
8331 = parsers.html_cdatasection_start
8332 * Cs(V("NoSoftLineBreakEndline") + parsers.any
8333 - parsers.nested_breaking_blank - parsers.html_cdatasection_end)^0
8334 * parsers.html_cdatasection_end
8335
8336 parsers.html_declaration_start = P("<!") * parsers.letter
8337
8338 parsers.html_declaration_end = P(">")
8339
8340 parsers.html_declaration
8341 = Cs(parsers.html_declaration_start
8342 * parsers.html_until_end(parsers.html_declaration_end))
8343
8344 parsers.html_inline_declaration
8345 = parsers.html_declaration_start
8346 * Cs(V("NoSoftLineBreakEndline") + parsers.any
8347 - parsers.nested_breaking_blank - parsers.html_declaration_end)^0
8348 * parsers.html_declaration_end
8349
8350 parsers.html_instruction_start = P("<?")
8351
8352 parsers.html_instruction_end = P("?>")
8353

```

```

8354 parsers.html_instruction
8355 = Cs(parsers.html_instruction_start
8356 * parsers.html_until_end(parsers.html_instruction_end))
8357
8358 parsers.html_inline_instruction = parsers.html_instruction_start
8359 * Cs(V("NoSoftLineBreakEndline")
8360 + parsers.any
8361 - parsers.nested_breaking_blank
8362 - parsers.html_instruction_end)^0
8363 * parsers.html_instruction_end
8364
8365 parsers.html_blankline = parsers.newline
8366 * parsers.optionalspace
8367 * parsers.newline
8368
8369 parsers.html_tag_start = parsers.less
8370
8371 parsers.html_tag_closing_start = parsers.less
8372 * parsers.slash
8373
8374 parsers.html_tag_end = parsers.html_spnl
8375 * parsers.more
8376
8377 parsers.html_empty_tag_end = parsers.html_spnl
8378 * parsers.slash
8379 * parsers.more
8380
8381 -- opening tags
8382 parsers.html_any_open_inline_tag = parsers.html_tag_start
8383 * parsers.keyword
8384 * parsers.html_attribute^0
8385 * parsers.html_tag_end
8386
8387 parsers.html_any_open_tag = parsers.html_tag_start
8388 * parsers.keyword
8389 * parsers.html_non_newline_attribute^0
8390 * parsers.html_tag_end
8391
8392 parsers.html_open_tag = parsers.html_tag_start
8393 * parsers.block_keyword
8394 * parsers.html_attribute^0
8395 * parsers.html_tag_end
8396
8397 parsers.html_open_special_tag = parsers.html_tag_start
8398 * parsers.special_block_keyword
8399 * parsers.html_attribute^0
8400 * parsers.html_tag_end

```

```

8401
8402 -- incomplete tags
8403 parsers.incomplete_tag_following = parsers.spacechar
8404 + parsers.more
8405 + parsers.slash * parsers.more
8406 + #(parsers.newline + parsers.eof)
8407
8408 parsers.incomplete_special_tag_following = parsers.spacechar
8409 + parsers.more
8410 + #(parsers.newline
8411 + parsers.eof)
8412
8413 parsers.html_incomplete_open_tag = parsers.html_tag_start
8414 * parsers.block_keyword
8415 * parsers.incomplete_tag_following
8416
8417 parsers.html_incomplete_open_special_tag
8418 = parsers.html_tag_start
8419 * parsers.special_block_keyword
8420 * parsers.incomplete_special_tag_following
8421
8422 parsers.html_incomplete_close_tag = parsers.html_tag_closing_start
8423 * parsers.block_keyword
8424 * parsers.incomplete_tag_following
8425
8426 parsers.html_incomplete_close_special_tag
8427 = parsers.html_tag_closing_start
8428 * parsers.special_block_keyword
8429 * parsers.incomplete_tag_following
8430
8431 -- closing tags
8432 parsers.html_close_tag = parsers.html_tag_closing_start
8433 * parsers.block_keyword
8434 * parsers.html_tag_end
8435
8436 parsers.html_any_close_tag = parsers.html_tag_closing_start
8437 * parsers.keyword
8438 * parsers.html_tag_end
8439
8440 parsers.html_close_special_tag = parsers.html_tag_closing_start
8441 * parsers.special_block_keyword
8442 * parsers.html_tag_end
8443
8444 -- empty tags
8445 parsers.html_any_empty_inline_tag = parsers.html_tag_start
8446 * parsers.keyword
8447 * parsers.html_attribute^0

```

```

8448 * parsers.html_empty_tag_end
8449
8450 parsers.html_any_empty_tag = parsers.html_tag_start
8451 * parsers.keyword
8452 * parsers.html_non_newline_attribute~0
8453 * parsers.optionalspace
8454 * parsers.slash
8455 * parsers.more
8456
8457 parsers.html_empty_tag = parsers.html_tag_start
8458 * parsers.block_keyword
8459 * parsers.html_attribute~0
8460 * parsers.html_empty_tag_end
8461
8462 parsers.html_empty_special_tag = parsers.html_tag_start
8463 * parsers.special_block_keyword
8464 * parsers.html_attribute~0
8465 * parsers.html_empty_tag_end
8466
8467 parsers.html_incomplete_blocks
8468 = parsers.html_incomplete_open_tag
8469 + parsers.html_incomplete_open_special_tag
8470 + parsers.html_incomplete_close_tag
8471
8472 -- parse special html blocks
8473 parsers.html_blankline_ending_special_block_opening
8474 = (parsers.html_close_special_tag
8475 + parsers.html_empty_special_tag)
8476 * #(parsers.optionalspace
8477 * (parsers.newline + parsers.eof))
8478
8479 parsers.html_blankline_ending_special_block
8480 = parsers.html_blankline_ending_special_block_opening
8481 * parsers.html_blankline_end_condition
8482
8483 parsers.html_special_block_opening
8484 = parsers.html_incomplete_open_special_tag
8485 - parsers.html_empty_special_tag
8486
8487 parsers.html_closing_special_block
8488 = parsers.html_special_block_opening
8489 * parsers.html_until_end(parsers.html_close_special_tag)
8490
8491 parsers.html_special_block
8492 = parsers.html_blankline_ending_special_block
8493 + parsers.html_closing_special_block
8494

```



```

8495 -- parse html blocks
8496 parsers.html_block_opening = parsers.html_incomplete_open_tag
8497 + parsers.html_incomplete_close_tag
8498
8499 parsers.html_block = parsers.html_block_opening
8500 * parsers.html_blankline_end_condition
8501
8502 -- parse any html blocks
8503 parsers.html_any_block_opening
8504 = (parsers.html_any_open_tag
8505 + parsers.html_any_close_tag
8506 + parsers.html_any_empty_tag)
8507 * #(parsers.optionalspace * (parsers.newline + parsers.eof))
8508
8509 parsers.html_any_block = parsers.html_any_block_opening
8510 * parsers.html_blankline_end_condition
8511
8512 parsers.html_inline_comment_full = parsers.html_comment_start
8513 * -P(">") * -P("->")
8514 * Cs((V("NoSoftLineBreakEndline")
8515 + parsers.any - P("--")
8516 - parsers.nested_breaking_blank
8517 - parsers.html_comment_end)^0)
8518 * parsers.html_comment_end
8519
8520 parsers.html_inline_tags = parsers.html_inline_comment_full
8521 + parsers.html_any_empty_inline_tag
8522 + parsers.html_inline_instruction
8523 + parsers.html_inline_cdatasection
8524 + parsers.html_inline_declaration
8525 + parsers.html_any_open_inline_tag
8526 + parsers.html_any_close_tag
8527

```

### 3.1.5.6 Parsers Used for Markdown Tags and Links

```

8528 parsers.urlchar = parsers.anyescaped
8529 - parsers.newline
8530 - parsers.more
8531
8532 parsers.auto_link_scheme_part = parsers.alphanumeric
8533 + parsers.plus
8534 + parsers.period
8535 + parsers.dash
8536
8537 parsers.auto_link_scheme = parsers.letter
8538 * parsers.auto_link_scheme_part

```

```

8539 * parsers.auto_link_scheme_part^-30
8540
8541 parsers.absolute_uri = parsers.auto_link_scheme * parsers.colon
8542 * (parsers.any - parsers.spacing
8543 - parsers.less - parsers.more)^0
8544
8545 parsers.printable_characters = S("!.#%&'*/+/?^_`{|}~-")
8546
8547 parsers.email_address_local_part_char = parsers.alphanumeric
8548 + parsers.printable_characters
8549
8550 parsers.email_address_local_part
8551 = parsers.email_address_local_part_char^1
8552
8553 parsers.email_address_dns_label = parsers.alphanumeric
8554 * (parsers.alphanumeric
8555 + parsers.dash)^-62
8556 * B(parsers.alphanumeric)
8557
8558 parsers.email_address_domain = parsers.email_address_dns_label
8559 * (parsers.period
8560 * parsers.email_address_dns_label)^0
8561
8562 parsers.email_address = parsers.email_address_local_part
8563 * parsers.at
8564 * parsers.email_address_domain
8565
8566 parsers.auto_link_url = parsers.less
8567 * C(parsers.absolute_uri)
8568 * parsers.more
8569
8570 parsers.auto_link_email = parsers.less
8571 * C(parsers.email_address)
8572 * parsers.more
8573
8574 parsers.auto_link_relative_reference = parsers.less
8575 * C(parsers.urlchar^1)
8576 * parsers.more
8577
8578 parsers.autolink = parsers.auto_link_url
8579 + parsers.auto_link_email
8580
8581 -- content in balanced brackets, parentheses, or quotes:
8582 parsers.bracketed = P{ parsers.lbracket
8583 * ((parsers.backslash / '"' * parsers.rbracket
8584 + parsers.any - (parsers.lbracket
8585 + parsers.rbracket

```

```

8586 + parsers.blankline^2)
8587) + V(1))^0
8588 * parsers.rbracket }
8589
8590 parsers.inparens = P{ parsers.lparent
8591 * ((parsers.anyescaped - (parsers.lparent
8592 + parsers.rparent
8593 + parsers.blankline^2)
8594) + V(1))^0
8595 * parsers.rparent }
8596
8597 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
8598 * ((parsers.anyescaped - (parsers.squote
8599 + parsers.blankline^2)
8600) + V(1))^0
8601 * parsers.squote }
8602
8603 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
8604 * ((parsers.anyescaped - (parsers.dquote
8605 + parsers.blankline^2)
8606) + V(1))^0
8607 * parsers.dquote }
8608
8609 parsers.link_text = parsers.lbracket
8610 * Cs((parsers.alphanumeric^1
8611 + parsers.bracketed
8612 + parsers.inticks
8613 + parsers.autolink
8614 + V("InlineHtml")
8615 + (parsers.backslash * parsers.backslash)
8616 + (parsers.backslash
8617 * (parsers.lbracket
8618 + parsers.rbracket)
8619 + V("NoSoftLineBreakSpace")
8620 + V("NoSoftLineBreakEndline")
8621 + (parsers.any
8622 - (parsers.newline
8623 + parsers.lbracket
8624 + parsers.rbracket
8625 + parsers.blankline^2))))^0)
8626 * parsers.rbracket
8627
8628 parsers.link_label_body = -#(parsers.sp * parsers.rbracket)
8629 * #((parsers.any
8630 - parsers.rbracket)^-999
8631 * parsers.rbracket)
8632 * Cs((parsers.alphanumeric^1

```

```

8633 + parsers.inticks
8634 + parsers.autolink
8635 + V("InlineHtml")
8636 + (parsers.backslash * parsers.backslash)
8637 + (parsers.backslash
8638 * (parsers.lbracket
8639 + parsers.rbracket)
8640 + V("NoSoftLineBreakSpace")
8641 + V("NoSoftLineBreakEndline")
8642 + (parsers.any
8643 - (parsers.newline
8644 + parsers.lbracket
8645 + parsers.rbracket
8646 + parsers.blankline^2))))^1)
8647
8648 parsers.link_label = parsers.lbracket
8649 * parsers.link_label_body
8650 * parsers.rbracket
8651
8652 parsers.inparens_url = P{ parsers.lparent
8653 * ((parsers.anyescaped - (parsers.lparent
8654 + parsers.rparent
8655 + parsers.spacing)
8656) + V(1))^0
8657 * parsers.rparent }
8658
8659 -- url for markdown links, allowing nested brackets:
8660 parsers.url = parsers.less * Cs((parsers.anyescaped
8661 - parsers.newline
8662 - parsers.less
8663 - parsers.more)^0)
8664 * parsers.more
8665 + -parsers.less
8666 * Cs((parsers.inparens_url + (parsers.anyescaped
8667 - parsers.spacing
8668 - parsers.lparent
8669 - parsers.rparent))^1)
8670
8671 -- quoted text:
8672 parsers.title_s = parsers.squote
8673 * Cs((parsers.html_entities
8674 + V("NoSoftLineBreakSpace")
8675 + V("NoSoftLineBreakEndline")
8676 + (parsers.anyescaped
8677 - parsers.newline
8678 - parsers.squote
8679 - parsers.blankline^2))^0)

```

```

8680 * parsers.squote
8681
8682 parsers.title_d = parsers.dquote
8683 * Cs((parsers.html_entities
8684 + V("NoSoftLineBreakSpace")
8685 + V("NoSoftLineBreakEndline")
8686 + (parsers.anyescaped
8687 - parsers.newline
8688 - parsers.dquote
8689 - parsers.blankline^2))^0)
8690 * parsers.dquote
8691
8692 parsers.title_p = parsers.lparent
8693 * Cs((parsers.html_entities
8694 + V("NoSoftLineBreakSpace")
8695 + V("NoSoftLineBreakEndline")
8696 + (parsers.anyescaped
8697 - parsers.newline
8698 - parsers.lparent
8699 - parsers.rparent
8700 - parsers.blankline^2))^0)
8701 * parsers.rparent
8702
8703 parsers.title
8704 = parsers.title_d + parsers.title_s + parsers.title_p
8705
8706 parsers.optionaltitle
8707 = parsers.spnlc * parsers.title * parsers.spacechar^0 + Cc("")
8708

```

### 3.1.5.7 Helpers for Links and Link Reference Definitions

```

8709 -- parse a reference definition: [foo]: /bar "title"
8710 parsers.define_reference_parser = (parsers.check_trail / "")
8711 * parsers.link_label * parsers.colon
8712 * parsers.spnlc * parsers.url
8713 * (parsers.spnlc_sep * parsers.title
8714 * parsers.only_blank
8715 + Cc("") * parsers.only_blank)

```

### 3.1.5.8 Inline Elements

```

8716 parsers.Inline = V("Inline")
8717
8718 -- parse many p between starter and ender
8719 parsers.between = function(p, starter, ender)
8720 local ender2 = B(parsers.nonspacechar) * ender
8721 return (starter

```

```

8722 * #parsers.nonspacechar
8723 * Ct(p * (p - ender2)^0)
8724 * ender2)
8725 end
8726

```

### 3.1.5.9 Block Elements

```

8727 parsers.lineof = function(c)
8728 return (parsers.check_trail_no_rem
8729 * (P(c) * parsers.optionalspace)^3
8730 * (parsers.newline + parsers.eof))
8731 end
8732
8733 parsers.thematic_break_lines = parsers.lineof(parsers.asterisk)
8734 + parsers.lineof(parsers.dash)
8735 + parsers.lineof(parsers.underscore)

```

### 3.1.5.10 Headings

```

8736 -- parse Atx heading start and return level
8737 parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
8738 * -parsers.hash / length
8739
8740 -- parse setext header ending and return level
8741 parsers.heading_level
8742 = parsers.nonindentSPACE * parsers.equal^1
8743 * parsers.optionalspace * #parsers.newline * Cc(1)
8744 + parsers.nonindentSPACE * parsers.dash^1
8745 * parsers.optionalspace * #parsers.newline * Cc(2)
8746
8747 local function strip_atx_end(s)
8748 return s:gsub("%s+#+%s*\n$", "")
8749 end
8750
8751 parsers.atx_heading = parsers.check_trail_no_rem
8752 * Cg(parsers.heading_start, "level")
8753 * (C(parsers.optionalspace
8754 * parsers.hash^0
8755 * parsers.optionalspace
8756 * parsers.newline)
8757 + parsers.spacechar^1
8758 * C(parsers.line))

```

### 3.1.6 Markdown Reader

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

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

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

```
8759 M.reader = {}
8760 function M.reader.new(writer, options)
8761 local self = {}
```

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

```
8762 self.writer = writer
8763 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.

```
8764 self.parsers = {}
8765 (function(parsers)
8766 setmetatable(self.parsers, {
8767 __index = function (_, key)
8768 return parsers[key]
8769 end
8770 })
8771 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.

```
8772 local parsers = self.parsers
```

#### 3.1.6.1 Top-Level Helper Functions

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

```
8773 function self.normalize_tag(tag)
8774 tag = util.rope_to_string(tag)
8775 tag = tag:gsub("[\\n\\r\\t]+", " ")
8776 tag = tag:gsub("^ ", ""):gsub(" $", "")
8777 tag = uni_algos.case.casefold(tag, true, false)
8778 return tag
```

```
8779 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.

```
8780 local function iterlines(s, f)
8781 local rope = lpeg.match(Ct((parsers.line / f)^1), s)
8782 return util.rope_to_string(rope)
8783 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.

```
8784 if options.preserveTabs then
8785 self.expandtabs = function(s) return s end
8786 else
8787 self.expandtabs = function(s)
8788 if s:find("\t") then
8789 return iterlines(s, util.expand_tabs_in_line)
8790 else
8791 return s
8792 end
8793 end
8794 end
```

### 3.1.6.2 High-Level Parser Functions

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

```
8795 self.parser_functions = {}
8796 self.create_parser = function(name, grammar, toplevel)
8797 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.

```
8798 if toplevel and options.stripIndent then
8799 local min_prefix_length, min_prefix = nil, ''
8800 str = iterlines(str, function(line)
8801 if lpeg.match(parsers.nonemptyline, line) == nil then
8802 return line
8803 end
8804 line = util.expand_tabs_in_line(line)
8805 local prefix = lpeg.match(C(parsers.optionalspace), line)
```



```

8806 local prefix_length = #prefix
8807 local is_shorter = min_prefix_length == nil
8808 if not is_shorter then
8809 is_shorter = prefix_length < min_prefix_length
8810 end
8811 if is_shorter then
8812 min_prefix_length, min_prefix = prefix_length, prefix
8813 end
8814 return line
8815 end)
8816 str = str:gsub('^' .. min_prefix, '')
8817 end

```

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

```

8818 if toplevel and (options.texComments or options.hybrid) then
8819 str = lpeg.match(Ct(parsers.commented_line^1), str)
8820 str = util.ropetostring(str)
8821 end
8822 local res = lpeg.match(grammar(), str)
8823 if res == nil then
8824 return writer.error(
8825 format("Parser `%s` failed to process the input text.", name),
8826 format("Here are the first 20 characters of the remaining "
8827 .. "unprocessed text: `%s`.", str:sub(1,20))
8828)
8829 else
8830 return res
8831 end
8832 end
8833 end
8834
8835 self.create_parser("parse_blocks",
8836 function()
8837 return parsers.blocks
8838 end, true)
8839
8840 self.create_parser("parse_blocks_nested",
8841 function()
8842 return parsers.blocks_nested
8843 end, false)
8844
8845 self.create_parser("parse_inlines",
8846 function()
8847 return parsers.inlines
8848 end, false)

```

```

8849
8850 self.create_parser("parse_inlines_no_inline_note",
8851 function()
8852 return parsers.inlines_no_inline_note
8853 end, false)
8854
8855 self.create_parser("parse_inlines_no_html",
8856 function()
8857 return parsers.inlines_no_html
8858 end, false)
8859
8860 self.create_parser("parse_inlines_nbsp",
8861 function()
8862 return parsers.inlines_nbsp
8863 end, false)
8864 self.create_parser("parse_inlines_no_link_or_emphasis",
8865 function()
8866 return parsers.inlines_no_link_or_emphasis
8867 end, false)

```

### 3.1.6.3 Parsers Used for Indentation (local)

The following patterns represent basic building blocks of indented content.

```

8868 parsers.minimally_indented_blankline
8869 = parsers.check_minimal_indent * (parsers.blankline / "")
8870
8871 parsers.minimally_indented_block
8872 = parsers.check_minimal_indent * V("Block")
8873
8874 parsers.minimally_indented_block_or_paragraph
8875 = parsers.check_minimal_indent * V("BlockOrParagraph")
8876
8877 parsers.minimally_indented_paragraph
8878 = parsers.check_minimal_indent * V("Paragraph")
8879
8880 parsers.minimally_indented_plain
8881 = parsers.check_minimal_indent * V("Plain")
8882
8883 parsers.minimally_indented_par_or_plain
8884 = parsers.minimally_indented_paragraph
8885 + parsers.minimally_indented_plain
8886
8887 parsers.minimally_indented_par_or_plain_no_blank
8888 = parsers.minimally_indented_par_or_plain
8889 - parsers.minimally_indented_blankline
8890
8891 parsers.minimally_indented_ref

```

```

8892 = parsers.check_minimal_indent * V("Reference")
8893
8894 parsers.minimally_indented_blank
8895 = parsers.check_minimal_indent * V("Blank")
8896
8897 parsers.conditionally_indented_blankline
8898 = parsers.check_minimal_blank_indent * (parsers.blankline / "")
8899
8900 parsers.minimally_indented_ref_or_block
8901 = parsers.minimally_indented_ref
8902 + parsers.minimally_indented_block
8903 - parsers.minimally_indented_blankline
8904
8905 parsers.minimally_indented_ref_or_block_or_par
8906 = parsers.minimally_indented_ref
8907 + parsers.minimally_indented_block_or_paragraph
8908 - parsers.minimally_indented_blankline
8909

```

The following pattern parses the properly indented content that follows the initial container start.

```

8910
8911 function parsers.separator_loop(separated_block, paragraph,
8912 block_separator, paragraph_separator)
8913 return separated_block
8914 + block_separator
8915 * paragraph
8916 * separated_block
8917 + paragraph_separator
8918 * paragraph
8919 end
8920
8921 function parsers.create_loop_body_pair(separated_block, paragraph,
8922 block_separator,
8923 paragraph_separator)
8924 return {
8925 block = parsers.separator_loop(separated_block, paragraph,
8926 block_separator, block_separator),
8927 par = parsers.separator_loop(separated_block, paragraph,
8928 block_separator, paragraph_separator)
8929 }
8930 end
8931
8932 parsers.block_sep_group = function(blank)
8933 return blank^0 * parsers.eof
8934 + (blank^2 / writer.paragraphsep
8935 + blank^0 / writer.interblocksep

```

```

8936)
8937 end
8938
8939 parsers.par_sep_group = function(blank)
8940 return blank^0 * parsers.eof
8941 + blank^0 / writer.paragraphsep
8942 end
8943
8944 parsers.sep_group_no_output = function(blank)
8945 return blank^0 * parsers.eof
8946 + blank^0
8947 end
8948
8949 parsers.content_blank = parsers.minimally_indented_blankline
8950
8951 parsers.ref_or_block_separated
8952 = parsers.sep_group_no_output(parsers.content_blank)
8953 * (parsers.minimally_indented_ref
8954 - parsers.content_blank)
8955 + parsers.block_sep_group(parsers.content_blank)
8956 * (parsers.minimally_indented_block
8957 - parsers.content_blank)
8958
8959 parsers.loop_body_pair =
8960 parsers.create_loop_body_pair(
8961 parsers.ref_or_block_separated,
8962 parsers.minimally_indented_par_or_plain_no_blank,
8963 parsers.block_sep_group(parsers.content_blank),
8964 parsers.par_sep_group(parsers.content_blank))
8965
8966 parsers.content_loop = (V("Block")
8967 * parsers.loop_body_pair.block^0
8968 + (V("Paragraph") + V("Plain")))
8969 * parsers.ref_or_block_separated
8970 * parsers.loop_body_pair.block^0
8971 + (V("Paragraph") + V("Plain")))
8972 * parsers.loop_body_pair.par^0)
8973 * parsers.content_blank^0
8974
8975 parsers.indented_content = function()
8976 return Ct((V("Reference") + (parsers.blankline / ""))
8977 * parsers.content_blank^0
8978 * parsers.check_minimal_indent
8979 * parsers.content_loop
8980 + (V("Reference") + (parsers.blankline / ""))
8981 * parsers.content_blank^0
8982 + parsers.content_loop)

```

```

8983 end
8984
8985 parsers.add_indent = function(pattern, name, breakable)
8986 return Cg(Cmt(Cb("indent_info")
8987 * Ct(pattern)
8988 * (#parsers.linechar -- check if starter is blank
8989 * Cc(false) + Cc(true))
8990 * Cc(name)
8991 * Cc(breakable),
8992 process_starter_indent), "indent_info")
8993 end
8994

```

#### 3.1.6.4 Parsers Used for Markdown Lists (local)

```

8995 if options.hashEnumerators then
8996 parsers.dig = parsers.digit + parsers.hash
8997 else
8998 parsers.dig = parsers.digit
8999 end
9000
9001 parsers.enumerator = function(delimiter_type, interrupting)
9002 local delimiter_range
9003 local allowed_end
9004 if interrupting then
9005 delimiter_range = P("1")
9006 allowed_end = C(parsers.spacechar^1) * #parsers.linechar
9007 else
9008 delimiter_range = parsers.dig * parsers.dig^-8
9009 allowed_end = C(parsers.spacechar^1)
9010 + #(parsers.newline + parsers.eof)
9011 end
9012
9013 return parsers.check_trail
9014 * Ct(C(delimiter_range) * C(delimiter_type))
9015 * allowed_end
9016 end
9017
9018 parsers.starter = parsers.bullet(parsers.dash)
9019 + parsers.bullet(parsers.asterisk)
9020 + parsers.bullet(parsers.plus)
9021 + parsers.enumerator(parsers.period)
9022 + parsers.enumerator(parsers.rparent)
9023

```

#### 3.1.6.5 Parsers Used for Blockquotes (local)

```

9024 parsers.blockquote_start

```

```

9025 = parsers.check_trail
9026 * C(parsers.more)
9027 * C(parsers.spacechar^0)
9028
9029 parsers.blockquote_body
9030 = parsers.add_indent(parsers.blockquote_start, "bq", true)
9031 * parsers.indented_content()
9032 * remove_indent("bq")
9033
9034 if not options.breakableBlockquotes then
9035 parsers.blockquote_body
9036 = parsers.add_indent(parsers.blockquote_start, "bq", false)
9037 * parsers.indented_content()
9038 * remove_indent("bq")
9039 end

```

### 3.1.6.6 Helpers for Emphasis and Strong Emphasis (local)

Parse the content of a table `content_part` with links, images and emphasis disabled.

```

9040 local function parse_content_part(content_part)
9041 local rope = util.rope_to_string(content_part)
9042 local parsed
9043 = self.parser_functions.parse_inlines_no_link_or_emphasis(rope)
9044 parsed.indent_info = nil
9045 return parsed
9046 end
9047

```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

9048 local collect_emphasis_content =
9049 function(t, opening_index, closing_index)
9050 local content = {}
9051
9052 local content_part = {}
9053 for i = opening_index, closing_index do
9054 local value = t[i]
9055
9056 if value.rendered ~= nil then
9057 content[#content + 1] = parse_content_part(content_part)
9058 content_part = {}
9059 content[#content + 1] = value.rendered
9060 value.rendered = nil
9061 else
9062 if value.type == "delimiter"
9063 and value.element == "emphasis" then
9064 if value.is_active then

```

```

9065 content_part[#content_part + 1]
9066 = string.rep(value.character, value.current_count)
9067 end
9068 else
9069 content_part[#content_part + 1] = value.content
9070 end
9071 value.content = ''
9072 value.is_active = false
9073 end
9074 end
9075
9076 if next(content_part) ~= nil then
9077 content[#content + 1] = parse_content_part(content_part)
9078 end
9079
9080 return content
9081 end
9082

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as emphasis.

```

9083 local function fill_emph(t, opening_index, closing_index)
9084 local content
9085 = collect_emphasis_content(t, opening_index + 1,
9086 closing_index - 1)
9087 t[opening_index + 1].is_active = true
9088 t[opening_index + 1].rendered = writer.emphasis(content)
9089 end
9090

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as strong emphasis.

```

9091 local function fill_strong(t, opening_index, closing_index)
9092 local content
9093 = collect_emphasis_content(t, opening_index + 1,
9094 closing_index - 1)
9095 t[opening_index + 1].is_active = true
9096 t[opening_index + 1].rendered = writer.strong(content)
9097 end
9098

```

Check whether the opening delimiter `opening_delimiter` and closing delimiter `closing_delimiter` break rule three together.

```

9099 local function breaks_three_rule(opening_delimiter, closing_delimiter)
9100 return (opening_delimiter.is_closing
9101 or closing_delimiter.is_opening)
9102 and ((opening_delimiter.original_count
9103 + closing_delimiter.original_count) % 3 == 0)

```

```

9104 and (opening_delimiter.original_count % 3 ~= 0
9105 or closing_delimiter.original_count % 3 ~= 0)
9106 end
9107

```

Look for the first potential emphasis opener in the delimiter table `t` in the range from `bottom_index` to `latest_index` that has the same character `character` as the closing delimiter `closing_delimiter`.

```

9108 local find_emphasis_opener = function(t, bottom_index, latest_index,
9109 character, closing_delimiter)
9110 for i = latest_index, bottom_index, -1 do
9111 local value = t[i]
9112 if value.is_active and
9113 value.is_opening and
9114 value.type == "delimiter" and
9115 value.element == "emphasis" and
9116 (value.character == character) and
9117 (value.current_count > 0) then
9118 if not breaks_three_rule(value, closing_delimiter) then
9119 return i
9120 end
9121 end
9122 end
9123 end
9124

```

Iterate over the delimiters in the delimiter table `t`, producing emphasis or strong emphasis macros.

```

9125 local function process_emphasis(t, opening_index, closing_index)
9126 for i = opening_index, closing_index do
9127 local value = t[i]
9128 if value.type == "delimiter" and value.element == "emphasis" then
9129 local delimiter_length = string.len(value.content)
9130 value.character = string.sub(value.content, 1, 1)
9131 value.current_count = delimiter_length
9132 value.original_count = delimiter_length
9133 end
9134 end
9135
9136 local openers_bottom = {
9137 ['*'] = {
9138 [true] = {opening_index, opening_index, opening_index},
9139 [false] = {opening_index, opening_index, opening_index}
9140 },
9141 ['_'] = {
9142 [true] = {opening_index, opening_index, opening_index},
9143 [false] = {opening_index, opening_index, opening_index}
9144 }
9145 }
9146

```



```

9145 }
9146
9147 local current_position = opening_index
9148 local max_position = closing_index
9149
9150 while current_position <= max_position do
9151 local value = t[current_position]
9152
9153 if value.type ~= "delimiter" or
9154 value.element ~= "emphasis" or
9155 not value.is_active or
9156 not value.is_closing or
9157 (value.current_count <= 0) then
9158 current_position = current_position + 1
9159 goto continue
9160 end
9161
9162 local character = value.character
9163 local is_opening = value.is_opening
9164 local closing_length_modulo_three = value.original_count % 3
9165
9166 local current_openers_bottom
9167 = openers_bottom[character][is_opening]
9168 [closing_length_modulo_three + 1]
9169
9170 local opener_position
9171 = find_emphasis_opener(t, current_openers_bottom,
9172 current_position - 1, character, value)
9173
9174 if (opener_position == nil) then
9175 openers_bottom[character][is_opening]
9176 [closing_length_modulo_three + 1]
9177 = current_position
9178 current_position = current_position + 1
9179 goto continue
9180 end
9181
9182 local opening_delimiter = t[opener_position]
9183
9184 local current_opening_count = opening_delimiter.current_count
9185 local current_closing_count = t[current_position].current_count
9186
9187 if (current_opening_count >= 2)
9188 and (current_closing_count >= 2) then
9189 opening_delimiter.current_count = current_opening_count - 2
9190 t[current_position].current_count = current_closing_count - 2
9191 fill_strong(t, opener_position, current_position)

```

```

9192 else
9193 opening_delimiter.current_count = current_opening_count - 1
9194 t[current_position].current_count = current_closing_count - 1
9195 fill_emph(t, opener_position, current_position)
9196 end
9197
9198 ::continue::
9199 end
9200 end
9201
9202 parsers.delimiter_run = function(character)
9203 return (B(parsers.backslash * character) + -B(character))
9204 * character^1
9205 * -#character
9206 end
9207
9208 parsers.left_flanking_delimiter_run = function(character)
9209 return (B(parsers.any)
9210 * (parsers.unicode.preceding_punctuation
9211 + parsers.unicode.preceding_whitespace)
9212 + -B(parsers.any))
9213 * parsers.delimiter_run(character)
9214 * parsers.unicode.following_punctuation
9215 + parsers.delimiter_run(character)
9216 * -(parsers.unicode.following_punctuation
9217 + parsers.unicode.following_whitespace
9218 + parsers.eof)
9219 end
9220
9221 parsers.right_flanking_delimiter_run = function(character)
9222 return parsers.unicode.preceding_punctuation
9223 * parsers.delimiter_run(character)
9224 * (parsers.unicode.following_punctuation
9225 + parsers.unicode.following_whitespace
9226 + parsers.eof)
9227 + (B(parsers.any)
9228 * -(parsers.unicode.preceding_punctuation
9229 + parsers.unicode.preceding_whitespace))
9230 * parsers.delimiter_run(character)
9231 end
9232
9233 if options.underscores then
9234 parsers.emph_start
9235 = parsers.left_flanking_delimiter_run(parsers.asterisk)
9236 + (-#parsers.right_flanking_delimiter_run(parsers.underscore)
9237 + (parsers.unicode.preceding_punctuation
9238 * #parsers.right_flanking_delimiter_run(parsers.underscore)))

```

```

9239 * parsers.left_flanking_delimiter_run(parsers.underscore)
9240
9241 parsers.emph_end
9242 = parsers.right_flanking_delimiter_run(parsers.asterisk)
9243 + (-#parsers.left_flanking_delimiter_run(parsers.underscore)
9244 + #(parsers.left_flanking_delimiter_run(parsers.underscore)
9245 * parsers.unicode.following_punctuation))
9246 * parsers.right_flanking_delimiter_run(parsers.underscore)
9247 else
9248 parsers.emph_start
9249 = parsers.left_flanking_delimiter_run(parsers.asterisk)
9250
9251 parsers.emph_end
9252 = parsers.right_flanking_delimiter_run(parsers.asterisk)
9253 end
9254
9255 parsers.emph_capturing_open_and_close
9256 = #parsers.emph_start * #parsers.emph_end
9257 * Ct(Cg(Cc("delimiter"), "type")
9258 * Cg(Cc("emphasis"), "element")
9259 * Cg(C(parsers.emph_start), "content")
9260 * Cg(Cc(true), "is_opening")
9261 * Cg(Cc(true), "is_closing"))
9262
9263 parsers.emph_capturing_open = Ct(Cg(Cc("delimiter"), "type")
9264 * Cg(Cc("emphasis"), "element")
9265 * Cg(C(parsers.emph_start), "content")
9266 * Cg(Cc(true), "is_opening")
9267 * Cg(Cc(false), "is_closing"))
9268
9269 parsers.emph_capturing_close = Ct(Cg(Cc("delimiter"), "type")
9270 * Cg(Cc("emphasis"), "element")
9271 * Cg(C(parsers.emph_end), "content")
9272 * Cg(Cc(false), "is_opening")
9273 * Cg(Cc(true), "is_closing"))
9274
9275 parsers.emph_open_or_close = parsers.emph_capturing_open_and_close
9276 + parsers.emph_capturing_open
9277 + parsers.emph_capturing_close
9278
9279 parsers.emph_open = parsers.emph_capturing_open_and_close
9280 + parsers.emph_capturing_open
9281
9282 parsers.emph_close = parsers.emph_capturing_open_and_close
9283 + parsers.emph_capturing_close
9284

```

### 3.1.6.7 Helpers for Links and Link Reference Definitions (local)

```
9285 -- List of references defined in the document
9286 local references
9287
9288 -- List of note references defined in the document
9289 parsers.rawnotes = {}
9290
```

The `reader->register_link` method registers a link reference, where `tag` is the link label, `url` is the link destination, `title` is the optional link title, and `attributes` are the optional attributes.

```
9291 function self.register_link(_, tag, url, title,
9292 attributes)
9293 local normalized_tag = self.normalize_tag(tag)
9294 if references[normalized_tag] == nil then
9295 references[normalized_tag] = {
9296 url = url,
9297 title = title,
9298 attributes = attributes
9299 }
9300 end
9301 return ""
9302 end
9303
```

The `reader->lookup_reference` method looks up a reference with link label `tag`.

```
9304 function self.lookup_reference(tag)
9305 return references[self.normalize_tag(tag)]
9306 end
9307
```

The `reader->lookup_note_reference` method looks up a note reference with label `tag`.

```
9308 function self.lookup_note_reference(tag)
9309 return parsers.rawnotes[self.normalize_tag(tag)]
9310 end
9311
9312 parsers.title_s_direct_ref = parsers.squote
9313 * Cs((parsers.html_entities
9314 + (parsers.anyescaped
9315 - parsers.squote
9316 - parsers.blankline^2))^0)
9317 * parsers.squote
9318
9319 parsers.title_d_direct_ref = parsers.dquote
9320 * Cs((parsers.html_entities
9321 + (parsers.anyescaped
9322 - parsers.dquote
```

```

9323 - parsers.blankline^2))^0)
9324 * parsers.dquote
9325
9326 parsers.title_p_direct_ref = parsers.lparent
9327 * Cs((parsers.html_entities
9328 + (parsers.anyescaped
9329 - parsers.lparent
9330 - parsers.rparent
9331 - parsers.blankline^2))^0)
9332 * parsers.rparent
9333
9334 parsers.title_direct_ref = parsers.title_s_direct_ref
9335 + parsers.title_d_direct_ref
9336 + parsers.title_p_direct_ref
9337
9338 parsers.inline_direct_ref_inside = parsers.lparent * parsers.spnl
9339 * Cg(parsers.url + Cc(""), "url")
9340 * parsers.spnl
9341 * Cg(parsers.title_direct_ref
9342 + Cc(""), "title")
9343 * parsers.spnl * parsers.rparent
9344
9345 parsers.inline_direct_ref = parsers.lparent * parsers.spnlc
9346 * Cg(parsers.url + Cc(""), "url")
9347 * parsers.spnlc
9348 * Cg(parsers.title + Cc(""), "title")
9349 * parsers.spnlc * parsers.rparent
9350
9351 parsers.empty_link = parsers.lbracket
9352 * parsers.rbracket
9353
9354 parsers.inline_link = parsers.link_text
9355 * parsers.inline_direct_ref
9356
9357 parsers.full_link = parsers.link_text
9358 * parsers.link_label
9359
9360 parsers.shortcut_link = parsers.link_label
9361 * -(parsers.empty_link + parsers.link_label)
9362
9363 parsers.collapsed_link = parsers.link_label
9364 * parsers.empty_link
9365
9366 parsers.image_opening = #(parsers.exclamation * parsers.inline_link)
9367 * Cg(Cc("inline"), "link_type")
9368 + #(parsers.exclamation * parsers.full_link)
9369 * Cg(Cc("full"), "link_type")

```

```

9370 + #(parsers.exclamation
9371 * parsers.collapsed_link)
9372 * Cg(Cc("collapsed"), "link_type")
9373 + #(parsers.exclamation * parsers.shortcut_link)
9374 * Cg(Cc("shortcut"), "link_type")
9375 + #(parsers.exclamation * parsers.empty_link)
9376 * Cg(Cc("empty"), "link_type")
9377
9378 parsers.link_opening = #parsers.inline_link
9379 * Cg(Cc("inline"), "link_type")
9380 + #parsers.full_link
9381 * Cg(Cc("full"), "link_type")
9382 + #parsers.collapsed_link
9383 * Cg(Cc("collapsed"), "link_type")
9384 + #parsers.shortcut_link
9385 * Cg(Cc("shortcut"), "link_type")
9386 + #parsers.empty_link
9387 * Cg(Cc("empty_link"), "link_type")
9388 + #parsers.link_text
9389 * Cg(Cc("link_text"), "link_type")
9390
9391 parsers.note_opening = #(parsers.circumflex * parsers.link_text)
9392 * Cg(Cc("note_inline"), "link_type")
9393
9394 parsers.raw_note_opening = #(parsers.lbracket
9395 * parsers.circumflex
9396 * parsers.link_label_body
9397 * parsers.rbracket)
9398 * Cg(Cc("raw_note"), "link_type")
9399
9400 local inline_note_element = Cg(Cc("note"), "element")
9401 * parsers.note_opening
9402 * Cg(parsers.circumflex
9403 * parsers.lbracket, "content")
9404
9405 local image_element = Cg(Cc("image"), "element")
9406 * parsers.image_opening
9407 * Cg(parsers.exclamation
9408 * parsers.lbracket, "content")
9409
9410 local note_element = Cg(Cc("note"), "element")
9411 * parsers.raw_note_opening
9412 * Cg(parsers.lbracket
9413 * parsers.circumflex, "content")
9414
9415 local link_element = Cg(Cc("link"), "element")
9416 * parsers.link_opening

```

```

9417 * Cg(parsers.lbracket, "content")
9418
9419 local opening_elements = parsers.fail
9420
9421 if options.inlineNotes then
9422 opening_elements = opening_elements + inline_note_element
9423 end
9424
9425 opening_elements = opening_elements + image_element
9426
9427 if options.notes then
9428 opening_elements = opening_elements + note_element
9429 end
9430
9431 opening_elements = opening_elements + link_element
9432
9433 parsers.link_image_opening = Ct(Cg(Cc("delimiter"), "type")
9434 * Cg(Cc(true), "is_opening")
9435 * Cg(Cc(false), "is_closing")
9436 * opening_elements)
9437
9438 parsers.link_image_closing = Ct(Cg(Cc("delimiter"), "type")
9439 * Cg(Cc("link"), "element")
9440 * Cg(Cc(false), "is_opening")
9441 * Cg(Cc(true), "is_closing")
9442 * (Cg(Cc(true), "is_direct")
9443 * Cg(parsers.rbracket
9444 * #parsers.inline_direct_ref,
9445 "content")
9446 + Cg(Cc(false), "is_direct")
9447 * Cg(parsers.rbracket, "content")))
9448
9449 parsers.link_image_open_or_close = parsers.link_image_opening
9450 + parsers.link_image_closing
9451
9452 if options.html then
9453 parsers.link_emph_precedence = parsers.inticks
9454 + parsers.autolink
9455 + parsers.html_inline_tags
9456 else
9457 parsers.link_emph_precedence = parsers.inticks
9458 + parsers.autolink
9459 end
9460
9461 parsers.link_and_emph_endline = parsers.newline
9462 * ((parsers.check_minimal_indent
9463 * -V("EndlineExceptions"))

```

```

9464 + parsers.check_optional_indent
9465 * -V("EndlineExceptions")
9466 * -V("ListStarter")) / "")
9467 * parsers.spacechar^0 / "\n"
9468
9469 parsers.link_and_emph_content
9470 = Ct(Cg(Cc("content"), "type")
9471 * Cg(Cs((parsers.link_emph_precedence
9472 + parsers.backslash * parsers.linechar
9473 + parsers.link_and_emph_endline
9474 + (parsers.linechar
9475 - parsers.blankline^2
9476 - parsers.link_image_open_or_close
9477 - parsers.emph_open_or_close))^0), "content"))
9478
9479 parsers.link_and_emph_table
9480 = (parsers.link_image_opening + parsers.emph_open)
9481 * parsers.link_and_emph_content
9482 * ((parsers.link_image_open_or_close + parsers.emph_open_or_close)
9483 * parsers.link_and_emph_content)^1
9484

```

Collect the content between the [opening\\_index](#) and [closing\\_index](#) in the delimiter table [t](#).

```

9485 local function collect_link_content(t, opening_index, closing_index)
9486 local content = {}
9487 for i = opening_index, closing_index do
9488 content[#content + 1] = t[i].content
9489 end
9490 return util.rope_to_string(content)
9491 end
9492

```

Look for the closest potential link opener in the delimiter table [t](#) in the range from [bottom\\_index](#) to [latest\\_index](#).

```

9493 local function find_link_opener(t, bottom_index, latest_index)
9494 for i = latest_index, bottom_index, -1 do
9495 local value = t[i]
9496 if value.type == "delimiter" and
9497 value.is_opening and
9498 (value.element == "link"
9499 or value.element == "image"
9500 or value.element == "note")
9501 and not value.removed then
9502 if value.is_active then
9503 return i
9504 end
9505 value.removed = true

```



```

9506 return nil
9507 end
9508 end
9509 end
9510

```

Find the position of a delimiter that closes a full link after an index `latest_index` in the delimiter table `t`.

```

9511 local function find_next_link_closing_index(t, latest_index)
9512 for i = latest_index, #t do
9513 local value = t[i]
9514 if value.is_closing and
9515 value.element == "link" and
9516 not value.removed then
9517 return i
9518 end
9519 end
9520 end
9521

```

Disable all preceding opening link delimiters by marking them inactive with the `is_active` property to prevent links within links. Images within links are allowed.

```

9522 local function disable_previous_link_openers(t, opening_index)
9523 if t[opening_index].element == "image" then
9524 return
9525 end
9526
9527 for i = opening_index, 1, -1 do
9528 local value = t[i]
9529 if value.is_active and
9530 value.type == "delimiter" and
9531 value.is_opening and
9532 value.element == "link" then
9533 value.is_active = false
9534 end
9535 end
9536 end
9537

```

Disable the delimiters between the `opening_index` and `closing_index` in the delimiter table `t` by marking them inactive with the `is_active` property.

```

9538 local function disable_range(t, opening_index, closing_index)
9539 for i = opening_index, closing_index do
9540 local value = t[i]
9541 if value.is_active then
9542 value.is_active = false
9543 if value.type == "delimiter" then
9544 value.removed = true

```

```

9545 end
9546 end
9547 end
9548 end
9549

```

Clear the parsed content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

9550 local delete_parsed_content_in_range =
9551 function(t, opening_index, closing_index)
9552 for i = opening_index, closing_index do
9553 t[i].rendered = nil
9554 end
9555 end
9556

```

Clear the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

9557 local function empty_content_in_range(t, opening_index, closing_index)
9558 for i = opening_index, closing_index do
9559 t[i].content = ''
9560 end
9561 end
9562

```

Join the attributes from the link reference definition `reference_attributes` with the link's own attributes `own_attributes`.

```

9563 local function join_attributes(reference_attributes, own_attributes)
9564 local merged_attributes = {}
9565 for _, attribute in ipairs(reference_attributes or {}) do
9566 table.insert(merged_attributes, attribute)
9567 end
9568 for _, attribute in ipairs(own_attributes or {}) do
9569 table.insert(merged_attributes, attribute)
9570 end
9571 if next(merged_attributes) == nil then
9572 merged_attributes = nil
9573 end
9574 return merged_attributes
9575 end
9576

```

Parse content between two delimiters in the delimiter table `t`. Produce the respective link and image macros.

```

9577 local render_link_or_image =
9578 function(t, opening_index, closing_index, content_end_index,
9579 reference)
9580 process_emphasis(t, opening_index, content_end_index)

```

```

9581 local mapped = collect_emphasis_content(t, opening_index + 1,
9582 content_end_index - 1)
9583
9584 local rendered = {}
9585 if (t[opening_index].element == "link") then
9586 rendered = writer.link(mapped, reference.url,
9587 reference.title, reference.attributes)
9588 end
9589
9590 if (t[opening_index].element == "image") then
9591 rendered = writer.image(mapped, reference.url, reference.title,
9592 reference.attributes)
9593 end
9594
9595 if (t[opening_index].element == "note") then
9596 if (t[opening_index].link_type == "note_inline") then
9597 rendered = writer.note(mapped)
9598 end
9599 if (t[opening_index].link_type == "raw_note") then
9600 rendered = writer.note(reference)
9601 end
9602 end
9603
9604 t[opening_index].rendered = rendered
9605 delete_parsed_content_in_range(t, opening_index + 1,
9606 closing_index)
9607 empty_content_in_range(t, opening_index, closing_index)
9608 disable_previous_link_openers(t, opening_index)
9609 disable_range(t, opening_index, closing_index)
9610 end
9611

```

Match the link destination of an inline link at index `closing_index` in table `t` when `match_reference` is true. Additionally, match attributes when the option `linkAttributes` is enabled.

```

9612 local resolve_inline_following_content =
9613 function(t, closing_index, match_reference, match_link_attributes)
9614 local content = ""
9615 for i = closing_index + 1, #t do
9616 content = content .. t[i].content
9617 end
9618
9619 local matching_content = parsers.succeed
9620
9621 if match_reference then
9622 matching_content = matching_content
9623 * parsers.inline_direct_ref_inside

```

```

9624 end
9625
9626 if match_link_attributes then
9627 matching_content = matching_content
9628 * Cg(Ct(parsers.attributes~-1), "attributes")
9629 end
9630
9631 local matched = lpeg.match(Ct(matching_content
9632 * Cg(Cp(), "end_position")), content)
9633
9634 local matched_count = matched.end_position - 1
9635 for i = closing_index + 1, #t do
9636 local value = t[i]
9637
9638 local chars_left = matched_count
9639 matched_count = matched_count - #value.content
9640
9641 if matched_count <= 0 then
9642 value.content = value.content:sub(chars_left + 1)
9643 break
9644 end
9645
9646 value.content = ''
9647 value.is_active = false
9648 end
9649
9650 local attributes = matched.attributes
9651 if attributes == nil or next(attributes) == nil then
9652 attributes = nil
9653 end
9654
9655 return {
9656 url = matched.url or "",
9657 title = matched.title or "",
9658 attributes = attributes
9659 }
9660 end
9661

```

Resolve an inline link `[a](b "c")` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Here, compared to other types of links, no reference definition is needed.

```

9662 local function resolve_inline_link(t, opening_index, closing_index)
9663 local inline_content
9664 = resolve_inline_following_content(t, closing_index, true,
9665 t.match_link_attributes)
9666 render_link_or_image(t, opening_index, closing_index,

```

```

9667 closing_index, inline_content)
9668 end
9669

```

Resolve an inline note `^[a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`.

```

9670 local function resolve_note_inline_link =
9671 function(t, opening_index, closing_index)
9672 local inline_content
9673 = resolve_inline_following_content(t, closing_index,
9674 false, false)
9675 render_link_or_image(t, opening_index, closing_index,
9676 closing_index, inline_content)
9677 end
9678

```

Resolve a shortcut link `[a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `a` is not found in the references.

```

9679 local function resolve_shortcut_link(t, opening_index, closing_index)
9680 local content
9681 = collect_link_content(t, opening_index + 1, closing_index - 1)
9682 local r = self.lookup_reference(content)
9683
9684 if r then
9685 local inline_content
9686 = resolve_inline_following_content(t, closing_index, false,
9687 t.match_link_attributes)
9688 r.attributes
9689 = join_attributes(r.attributes, inline_content.attributes)
9690 render_link_or_image(t, opening_index, closing_index,
9691 closing_index, r)
9692 end
9693 end
9694

```

Resolve a note `[^a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `a` is not found in the rawnotes.

```

9695 local function resolve_raw_note_link(t, opening_index, closing_index)
9696 local content
9697 = collect_link_content(t, opening_index + 1, closing_index - 1)
9698 local r = self.lookup_note_reference(content)
9699
9700 if r then
9701 local parsed_ref = self.parser_functions.parse_blocks_nested(r)
9702 render_link_or_image(t, opening_index, closing_index,
9703 closing_index, parsed_ref)
9704 end

```

```

9705 end
9706

```

Resolve a full link [\[a\]\[b\]](#) from the delimiters at [opening\\_index](#) and [closing\\_index](#) within a delimiter table [t](#). Continue if a tag [b](#) is not found in the references.

```

9707 local function resolve_full_link(t, opening_index, closing_index)
9708 local next_link_closing_index
9709 = find_next_link_closing_index(t, closing_index + 4)
9710 local next_link_content
9711 = collect_link_content(t, closing_index + 3,
9712 next_link_closing_index - 1)
9713 local r = self.lookup_reference(next_link_content)
9714
9715 if r then
9716 local inline_content
9717 = resolve_inline_following_content(t, next_link_closing_index,
9718 false,
9719 t.match_link_attributes)
9720 r.attributes
9721 = join_attributes(r.attributes, inline_content.attributes)
9722 render_link_or_image(t, opening_index, next_link_closing_index,
9723 closing_index, r)
9724 end
9725 end
9726

```

Resolve a collapsed link [\[a\]](#) from the delimiters at [opening\\_index](#) and [closing\\_index](#) within a delimiter table [t](#). Continue if a tag [a](#) is not found in the references.

```

9727 local function resolve_collapsed_link(t, opening_index, closing_index)
9728 local next_link_closing_index
9729 = find_next_link_closing_index(t, closing_index + 4)
9730 local content
9731 = collect_link_content(t, opening_index + 1, closing_index - 1)
9732 local r = self.lookup_reference(content)
9733
9734 if r then
9735 local inline_content
9736 = resolve_inline_following_content(t, closing_index, false,
9737 t.match_link_attributes)
9738 r.attributes
9739 = join_attributes(r.attributes, inline_content.attributes)
9740 render_link_or_image(t, opening_index, next_link_closing_index,
9741 closing_index, r)
9742 end
9743 end
9744

```

Parse a table of link and emphasis delimiters `t`. First, iterate over the link delimiters and produce either link or image macros. Then run `process_emphasis` over the entire delimiter table, resolving emphasis and strong emphasis and parsing any content outside of closed delimiters.

```

9745 local function process_links_and_emphasis(t)
9746 for _,value in ipairs(t) do
9747 value.is_active = true
9748 end
9749
9750 for i,value in ipairs(t) do
9751 if not value.is_closing
9752 or value.type ~= "delimiter"
9753 or not (value.element == "link"
9754 or value.element == "image"
9755 or value.element == "note")
9756 or value.removed then
9757 goto continue
9758 end
9759
9760 local opener_position = find_link_opener(t, 1, i - 1)
9761 if (opener_position == nil) then
9762 goto continue
9763 end
9764
9765 local opening_delimiter = t[opener_position]
9766 opening_delimiter.removed = true
9767
9768 local link_type = opening_delimiter.link_type
9769
9770 if (link_type == "inline") then
9771 resolve_inline_link(t, opener_position, i)
9772 end
9773 if (link_type == "shortcut") then
9774 resolve_shortcut_link(t, opener_position, i)
9775 end
9776 if (link_type == "full") then
9777 resolve_full_link(t, opener_position, i)
9778 end
9779 if (link_type == "collapsed") then
9780 resolve_collapsed_link(t, opener_position, i)
9781 end
9782 if (link_type == "note_inline") then
9783 resolve_note_inline_link(t, opener_position, i)
9784 end
9785 if (link_type == "raw_note") then
9786 resolve_raw_note_link(t, opener_position, i)

```

```

9787 end
9788
9789 ::continue::
9790 end
9791
9792 t[#t].content = t[#t].content:gsub("%s*$","")
9793
9794 process_emphasis(t, 1, #t)
9795 local final_result = collect_emphasis_content(t, 1, #t)
9796 return final_result
9797 end
9798
9799 function self.defer_link_and_emphasis_processing(delimiter_table)
9800 return writer.defer_call(function()
9801 return process_links_and_emphasis(delimiter_table)
9802 end)
9803 end
9804

```

### 3.1.6.8 Inline Elements (local)

```

9805 parsers.Str = (parsers.normalchar
9806 * (parsers.normalchar + parsers.at)^0)
9807 / writer.string
9808
9809 parsers.Symbol = (parsers.backtick^1 + V("SpecialChar"))
9810 / writer.string
9811
9812 parsers.Ellipsis = P("...") / writer.ellipsis
9813
9814 parsers.Smart = parsers.Ellipsis
9815
9816 parsers.Code = parsers.inticks / writer.code
9817
9818 if options.blankBeforeBlockquote then
9819 parsers.bqstart = parsers.fail
9820 else
9821 parsers.bqstart = parsers.blockquote_start
9822 end
9823
9824 if options.blankBeforeHeading then
9825 parsers.headerstart = parsers.fail
9826 else
9827 parsers.headerstart = parsers.atx_heading
9828 end
9829
9830 if options.blankBeforeList then

```



```

9831 parsers.interrupting_bullets = parsers.fail
9832 parsers.interrupting_enumerators = parsers.fail
9833 else
9834 parsers.interrupting_bullets
9835 = parsers.bullet(parsers.dash, true)
9836 + parsers.bullet(parsers.asterisk, true)
9837 + parsers.bullet(parsers.plus, true)
9838
9839 parsers.interrupting_enumerators
9840 = parsers.enumerator(parsers.period, true)
9841 + parsers.enumerator(parsers.rparent, true)
9842 end
9843
9844 if options.html then
9845 parsers.html_interrupting
9846 = parsers.check_trail
9847 * (parsers.html_incomplete_open_tag
9848 + parsers.html_incomplete_close_tag
9849 + parsers.html_incomplete_open_special_tag
9850 + parsers.html_comment_start
9851 + parsers.html_cdatasection_start
9852 + parsers.html_declaration_start
9853 + parsers.html_instruction_start
9854 - parsers.html_close_special_tag
9855 - parsers.html_empty_special_tag)
9856 else
9857 parsers.html_interrupting = parsers.fail
9858 end
9859
9860 parsers.ListStarter = parsers.starter
9861
9862 parsers.EndlineExceptions
9863 = parsers.blankline -- paragraph break
9864 + parsers.eof -- end of document
9865 + parsers.bqstart
9866 + parsers.thematic_break_lines
9867 + parsers.interrupting_bullets
9868 + parsers.interrupting_enumerators
9869 + parsers.headerstart
9870 + parsers.html_interrupting
9871
9872 parsers.NoSoftLineBreakEndlineExceptions = parsers.EndlineExceptions
9873
9874 parsers.endline = parsers.newline
9875 * (parsers.check_minimal_indent
9876 * -V("EndlineExceptions")
9877 + parsers.check_optional_indent

```

```

9878 * -V("EndlineExceptions")
9879 * -V("ListStarter")) / function(_) return end
9880 * parsers.spacechar^0
9881
9882 parsers.Endline = parsers.endline
9883 / writer.soft_line_break
9884
9885 parsers.EndlineNoSub = parsers.endline
9886
9887 parsers.NoSoftLineBreakEndline
9888 = parsers.newline
9889 * (parsers.check_minimal_indent
9890 * -V("NoSoftLineBreakEndlineExceptions")
9891 + parsers.check_optional_indent
9892 * -V("NoSoftLineBreakEndlineExceptions")
9893 * -V("ListStarter"))
9894 * parsers.spacechar^0
9895 / writer.space
9896
9897 parsers.EndlineBreak = parsers.backslash * parsers.endline
9898 / writer.hard_line_break
9899
9900 parsers.OptionalIndent
9901 = parsers.spacechar^1 / writer.space
9902
9903 parsers.Space = parsers.spacechar^2 * parsers.endline
9904 / writer.hard_line_break
9905 + parsers.spacechar^1
9906 * parsers.endline~-1
9907 * parsers.eof / self.expandtabs
9908 + parsers.spacechar^1 * parsers.endline
9909 / writer.soft_line_break
9910 + parsers.spacechar^1
9911 * -parsers.newline / self.expandtabs
9912 + parsers.spacechar^1
9913
9914 parsers.NoSoftLineBreakSpace
9915 = parsers.spacechar^2 * parsers.endline
9916 / writer.hard_line_break
9917 + parsers.spacechar^1
9918 * parsers.endline~-1
9919 * parsers.eof / self.expandtabs
9920 + parsers.spacechar^1 * parsers.endline
9921 / writer.soft_line_break
9922 + parsers.spacechar^1
9923 * -parsers.newline / self.expandtabs
9924 + parsers.spacechar^1

```

```

9925
9926 parsers.NonbreakingEndline
9927 = parsers.endline
9928 / writer.nbsp
9929
9930 parsers.NonbreakingSpace
9931 = parsers.spacechar^2 * parsers.endline
9932 / writer.nbsp
9933 + parsers.spacechar^1
9934 * parsers.endline~-1 * parsers.eof / ""
9935 + parsers.spacechar^1 * parsers.endline
9936 * parsers.optionalspace
9937 / writer.nbsp
9938 + parsers.spacechar^1 * parsers.optionalspace
9939 / writer.nbsp
9940

```

The `reader->auto_link_url` method produces an autolink to a URL or a relative reference in the output format, where `url` is the link destination and `attributes` are the optional attributes.

```

9941 function self.auto_link_url(url, attributes)
9942 return writer.link(writer.escape(url),
9943 url, nil, attributes)
9944 end

```

The `reader->auto_link_email` method produces an autolink to an e-mail in the output format, where `email` is the email address destination and `attributes` are the optional attributes.

```

9945 function self.auto_link_email(email, attributes)
9946 return writer.link(writer.escape(email),
9947 "mailto:".email,
9948 nil, attributes)
9949 end
9950
9951 parsers.AutoLinkUrl = parsers.auto_link_url
9952 / self.auto_link_url
9953
9954 parsers.AutoLinkEmail
9955 = parsers.auto_link_email
9956 / self.auto_link_email
9957
9958 parsers.AutoLinkRelativeReference
9959 = parsers.auto_link_relative_reference
9960 / self.auto_link_url
9961
9962 parsers.LinkAndEmph = Ct(parsers.link_and_emph_table)
9963 / self.defer_link_and_emphasis_processing

```

```

9964
9965 parsers.EscapedChar = parsers.backslash
9966 * C(parsers.escapable) / writer.string
9967
9968 parsers.InlineHtml = Cs(parsers.html_inline_comment)
9969 / writer.inline_html_comment
9970 + Cs(parsers.html_any_empty_inline_tag
9971 + parsers.html_inline_instruction
9972 + parsers.html_inline_cdatasection
9973 + parsers.html_inline_declaration
9974 + parsers.html_any_open_inline_tag
9975 + parsers.html_any_close_tag)
9976 / writer.inline_html_tag
9977
9978 parsers.HtmlEntity = parsers.html_entities / writer.string

```

### 3.1.6.9 Block Elements (local)

```

9979 parsers.DisplayHtml = Cs(parsers.check_trail
9980 * (parsers.html_comment
9981 + parsers.html_special_block
9982 + parsers.html_block
9983 + parsers.html_any_block
9984 + parsers.html_instruction
9985 + parsers.html_cdatasection
9986 + parsers.html_declaration))
9987 / writer.block_html_element
9988
9989 parsers.indented_non_blank_line = parsers.indentedline
9990 - parsers.blankline
9991
9992 parsers.Verbatim
9993 = Cs(parsers.check_code_trail
9994 * (parsers.line - parsers.blankline)
9995 * ((parsers.check_minimal_blank_indent_and_full_code_trail
9996 * parsers.blankline)^0
9997 * ((parsers.check_minimal_indent / "")
9998 * parsers.check_code_trail
9999 * (parsers.line - parsers.blankline))^1)^0)
10000 / self.expandtabs / writer.verbatim
10001
10002 parsers.Blockquote = parsers.blockquote_body
10003 / writer.blockquote
10004
10005 parsers.ThematicBreak = parsers.thematic_break_lines
10006 / writer.thematic_break
10007

```

```

10008 parsers.Reference = parsers.define_reference_parser
10009 / self.register_link
10010
10011 parsers.Paragraph = parsers.freeze_trail
10012 * (Ct((parsers.Inline)^1)
10013 * (parsers.newline + parsers.eof)
10014 * parsers.unfreeze_trail
10015 / writer.paragraph)
10016
10017 parsers.Plain = parsers.nonindent_space * Ct(parsers.Inline^1)
10018 / writer.plain

```

### 3.1.6.10 Lists (local)

```

10019
10020 if options.taskLists then
10021 parsers.tickbox = (parsers.ticked_box
10022 + parsers.halfticked_box
10023 + parsers.unticked_box
10024) / writer.tickbox
10025 else
10026 parsers.tickbox = parsers.fail
10027 end
10028
10029 parsers.list_blank = parsers.conditionally_indented_blankline
10030
10031 parsers.ref_or_block_list_separated
10032 = parsers.sep_group_no_output(parsers.list_blank)
10033 * parsers.minimally_indented_ref
10034 + parsers.block_sep_group(parsers.list_blank)
10035 * parsers.minimally_indented_block
10036
10037 parsers.ref_or_block_non_separated
10038 = parsers.minimally_indented_ref
10039 + (parsers.succeed / writer.interblocksep)
10040 * parsers.minimally_indented_block
10041 - parsers.minimally_indented_blankline
10042
10043 parsers.tight_list_loop_body_pair =
10044 parsers.create_loop_body_pair(
10045 parsers.ref_or_block_non_separated,
10046 parsers.minimally_indented_par_or_plain_no_blank,
10047 (parsers.succeed / writer.interblocksep),
10048 (parsers.succeed / writer.paragraphsep))
10049
10050 parsers.loose_list_loop_body_pair =
10051 parsers.create_loop_body_pair(

```

```

10052 parsers.ref_or_block_list_separated,
10053 parsers.minimally_indented_par_or_plain,
10054 parsers.block_sep_group(parsers.list_blank),
10055 parsers.par_sep_group(parsers.list_blank))
10056
10057 parsers.tight_list_content_loop
10058 = V("Block")
10059 * parsers.tight_list_loop_body_pair.block^0
10060 + (V("Paragraph") + V("Plain"))
10061 * parsers.ref_or_block_non_separated
10062 * parsers.tight_list_loop_body_pair.block^0
10063 + (V("Paragraph") + V("Plain"))
10064 * parsers.tight_list_loop_body_pair.par^0
10065
10066 parsers.loose_list_content_loop
10067 = V("Block")
10068 * parsers.loose_list_loop_body_pair.block^0
10069 + (V("Paragraph") + V("Plain"))
10070 * parsers.ref_or_block_list_separated
10071 * parsers.loose_list_loop_body_pair.block^0
10072 + (V("Paragraph") + V("Plain"))
10073 * parsers.loose_list_loop_body_pair.par^0
10074
10075 parsers.list_item_tightness_condition
10076 = -#(parsers.list_blank^0
10077 * parsers.minimally_indented_ref_or_block_or_par)
10078 * remove_indent("li")
10079 + remove_indent("li")
10080 * parsers.fail
10081
10082 parsers.indented_content_tight
10083 = Ct((parsers.blankline / "")
10084 * #parsers.list_blank
10085 * remove_indent("li")
10086 + ((V("Reference") + (parsers.blankline / ""))
10087 * parsers.check_minimal_indent
10088 * parsers.tight_list_content_loop
10089 + (V("Reference") + (parsers.blankline / ""))
10090 + (parsers.tickbox^1 / writer.escape)
10091 * parsers.tight_list_content_loop
10092)
10093 * parsers.list_item_tightness_condition)
10094
10095 parsers.indented_content_loose
10096 = Ct((parsers.blankline / "")
10097 * #parsers.list_blank
10098 + ((V("Reference") + (parsers.blankline / ""))

```

```

10099 * parsers.check_minimal_indent
10100 * parsers.loose_list_content_loop
10101 + (V("Reference") + (parsers.blankline / ""))
10102 + (parsers.tickbox~-1 / writer.escape)
10103 * parsers.loose_list_content_loop))
10104
10105 parsers.TightListItem = function(starter)
10106 return -parsers.ThematicBreak
10107 * parsers.add_indent(starter, "li")
10108 * parsers.indented_content_tight
10109 end
10110
10111 parsers.LooseListItem = function(starter)
10112 return -parsers.ThematicBreak
10113 * parsers.add_indent(starter, "li")
10114 * parsers.indented_content_loose
10115 * remove_indent("li")
10116 end
10117
10118 parsers.BulletListOfType = function(bullet_type)
10119 local bullet = parsers.bullet(bullet_type)
10120 return (Ct(parsers.TightListItem(bullet)
10121 * ((parsers.check_minimal_indent / "")
10122 * parsers.TightListItem(bullet)
10123)^0
10124)
10125 * Cc(true)
10126 * -#((parsers.list_blank^0 / "")
10127 * parsers.check_minimal_indent
10128 * (bullet - parsers.ThematicBreak)
10129)
10130 + Ct(parsers.LooseListItem(bullet)
10131 * ((parsers.list_blank^0 / "")
10132 * (parsers.check_minimal_indent / "")
10133 * parsers.LooseListItem(bullet)
10134)^0
10135)
10136 * Cc(false)
10137) / writer.bulletlist
10138 end
10139
10140 parsers.BulletList = parsers.BulletListOfType(parsers.dash)
10141 + parsers.BulletListOfType(parsers.asterisk)
10142 + parsers.BulletListOfType(parsers.plus)
10143
10144 local function ordered_list(items,tight,starter)
10145 local startnum = starter[2][1]

```

```

10146 if options.startNumber then
10147 startnum = tonumber(startnum) or 1 -- fallback for '#'
10148 if startnum ~= nil then
10149 startnum = math.floor(startnum)
10150 end
10151 else
10152 startnum = nil
10153 end
10154 return writer.orderedlist(items,tight,startnum)
10155 end
10156
10157 parsers.OrderedListOfType = function(delimiter_type)
10158 local enumerator = parsers.enumerator(delimiter_type)
10159 return Cg(enumerator, "listtype")
10160 * (Ct(parsers.TightListItem(Cb("listtype"))
10161 * ((parsers.check_minimal_indent / "")
10162 * parsers.TightListItem(enumerator))^0)
10163 * Cc(true)
10164 * -#((parsers.list_blank^0 / "")
10165 * parsers.check_minimal_indent * enumerator)
10166 + Ct(parsers.LooseListItem(Cb("listtype"))
10167 * ((parsers.list_blank^0 / "")
10168 * (parsers.check_minimal_indent / "")
10169 * parsers.LooseListItem(enumerator))^0)
10170 * Cc(false)
10171) * Ct(Cb("listtype")) / ordered_list
10172 end
10173
10174 parsers.OrderedList = parsers.OrderedListOfType(parsers.period)
10175 + parsers.OrderedListOfType(parsers.rparent)

```

### 3.1.6.11 Blank (local)

```

10176 parsers.Blank = parsers.blankline / ""
10177 + V("Reference")

```

### 3.1.6.12 Headings (local)

```

10178 function parsers.parse_heading_text(s)
10179 local inlines = self.parser_functions.parse_inlines(s)
10180 local flatten_inlines = self.writer.flatten_inlines
10181 self.writer.flatten_inlines = true
10182 local flat_text = self.parser_functions.parse_inlines(s)
10183 flat_text = util.rope_to_string(flat_text)
10184 self.writer.flatten_inlines = flatten_inlines
10185 return {flat_text, inlines}
10186 end
10187

```



```

10188 -- parse atx header
10189 parsers.AtxHeading = parsers.check_trail_no_rem
10190 * Cg(parsers.heading_start, "level")
10191 * ((C(parsers.optionalspace
10192 * parsers.hash^0
10193 * parsers.optionalspace
10194 * parsers.newline)
10195 + parsers.spacechar^1
10196 * C(parsers.line))
10197 / strip_atx_end
10198 / parsers.parse_heading_text)
10199 * Cb("level")
10200 / writer.heading
10201
10202 parsers.heading_line = parsers.linechar^1
10203 - parsers.thematic_break_lines
10204
10205 parsers.heading_text = parsers.heading_line
10206 * ((V("Endline") / "\n")
10207 * (parsers.heading_line
10208 - parsers.heading_level))^0
10209 * parsers.newline^-1
10210
10211 parsers.SettextHeading = parsers.freeze_trail
10212 * parsers.check_trail_no_rem
10213 * #(parsers.heading_text
10214 * parsers.check_minimal_indent
10215 * parsers.check_trail
10216 * parsers.heading_level)
10217 * Cs(parsers.heading_text)
10218 / parsers.parse_heading_text
10219 * parsers.check_minimal_indent_and_trail
10220 * parsers.heading_level
10221 * parsers.newline
10222 * parsers.unfreeze_trail
10223 / writer.heading
10224
10225 parsers.Heading = parsers.AtxHeading + parsers.SettextHeading

```

### 3.1.6.13 Syntax Specification

Define `reader->finalize_grammar` as a function that constructs the PEG grammar of markdown, applies syntax extensions `extensions` and returns a conversion function that takes a markdown string and turns it into a plain  $\text{\TeX}$  output.

```

10226 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.

```

10227 local walkable_syntax = (function(global_walkable_syntax)
10228 local local_walkable_syntax = {}
10229 for lhs, rule in pairs(global_walkable_syntax) do
10230 local_walkable_syntax[lhs] = util.table_copy(rule)
10231 end
10232 return local_walkable_syntax
10233 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.

```

10234 local current_extension_name = nil
10235 self.insert_pattern = function(selector, pattern, pattern_name)
10236 assert(pattern_name == nil or type(pattern_name) == "string")
10237 local _, _, lhs, pos, rhs
10238 = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
10239 assert(lhs ~= nil,
10240 [[Expected selector in form]]
10241 .. ["LHS (before|after|instead of) RHS", not "]]
10242 .. selector .. ["]])
10243 assert(walkable_syntax[lhs] ~= nil,
10244 [[Rule]] .. lhs
10245 .. [[-> ... does not exist in markdown grammar]])
10246 assert(pos == "before" or pos == "after" or pos == "instead of",
10247 [[Expected positional specifier "before", "after",]]
10248 .. [[or "instead of", not "]]
10249 .. pos .. ["]])
10250 local rule = walkable_syntax[lhs]
10251 local index = nil
10252 for current_index, current_rhs in ipairs(rule) do
10253 if type(current_rhs) == "string" and current_rhs == rhs then
10254 index = current_index
10255 if pos == "after" then
10256 index = index + 1
10257 end
10258 break
10259 end
10260 end
10261 assert(index ~= nil,
10262 [[Rule]] .. lhs .. [[->]] .. rhs
10263 .. [[does not exist in markdown grammar]])
10264 local accountable_pattern
10265 if current_extension_name then
10266 accountable_pattern

```

```

10267 = {pattern, current_extension_name, pattern_name}
10268 else
10269 assert(type(pattern) == "string",
10270 [[reader->insert_pattern() was called outside]]
10271 .. [[an extension with]]
10272 .. [[a PEG pattern instead of a rule name]])
10273 accountable_pattern = pattern
10274 end
10275 if pos == "instead of" then
10276 rule[index] = accountable_pattern
10277 else
10278 table.insert(rule, index, accountable_pattern)
10279 end
10280 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.

```

10281 local syntax =
10282 { "Blocks",
10283
10284 Blocks = V("InitializeState")
10285 * V("ExpectedJekyllData")
10286 * V("Blank")^0

```

Only create interblock separators between pairs of blocks that are not both paragraphs. Between a pair of paragraphs, any number of blank lines will always produce a paragraph separator.

```

10287 * (V("Block")
10288 * (V("Blank")^0 * parsers.eof
10289 + (V("Blank")^2 / writer.paragraphsep
10290 + V("Blank")^0 / writer.interblocksep
10291)
10292)
10293 + (V("Paragraph") + V("Plain"))
10294 * (V("Blank")^0 * parsers.eof
10295 + (V("Blank")^2 / writer.paragraphsep
10296 + V("Blank")^0 / writer.interblocksep
10297)
10298)
10299 * V("Block")
10300 * (V("Blank")^0 * parsers.eof
10301 + (V("Blank")^2 / writer.paragraphsep
10302 + V("Blank")^0 / writer.interblocksep
10303)
10304)
10305 + (V("Paragraph") + V("Plain"))
10306 * (V("Blank")^0 * parsers.eof
10307 + V("Blank")^0 / writer.paragraphsep

```

```

10308)
10309)~0,
10310
10311 ExpectedJekyllData = parsers.succeed,
10312
10313 Blank = parsers.Blank,
10314 Reference = parsers.Reference,
10315
10316 Blockquote = parsers.Blockquote,
10317 Verbatim = parsers.Verbatim,
10318 ThematicBreak = parsers.ThematicBreak,
10319 BulletList = parsers.BulletList,
10320 OrderedList = parsers.OrderedList,
10321 DisplayHtml = parsers.DisplayHtml,
10322 Heading = parsers.Heading,
10323 Paragraph = parsers.Paragraph,
10324 Plain = parsers.Plain,
10325
10326 ListStarter = parsers.ListStarter,
10327 EndlineExceptions = parsers.EndlineExceptions,
10328 NoSoftLineBreakEndlineExceptions
10329 = parsers.NoSoftLineBreakEndlineExceptions,
10330
10331 Str = parsers.Str,
10332 Space = parsers.Space,
10333 NoSoftLineBreakSpace
10334 = parsers.NoSoftLineBreakSpace,
10335 OptionalIndent = parsers.OptionalIndent,
10336 Endline = parsers.Endline,
10337 EndlineNoSub = parsers.EndlineNoSub,
10338 NoSoftLineBreakEndline
10339 = parsers.NoSoftLineBreakEndline,
10340 EndlineBreak = parsers.EndlineBreak,
10341 LinkAndEmph = parsers.LinkAndEmph,
10342 Code = parsers.Code,
10343 AutoLinkUrl = parsers.AutoLinkUrl,
10344 AutoLinkEmail = parsers.AutoLinkEmail,
10345 AutoLinkRelativeReference
10346 = parsers.AutoLinkRelativeReference,
10347 InlineHtml = parsers.InlineHtml,
10348 HtmlEntity = parsers.HtmlEntity,
10349 EscapedChar = parsers.EscapedChar,
10350 Smart = parsers.Smart,
10351 Symbol = parsers.Symbol,
10352 SpecialChar = parsers.fail,
10353 InitializeState = parsers.succeed,
10354 }

```

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]`.

```

10355 self.update_rule = function(rule_name, get_pattern)
10356 assert(current_extension_name ~= nil)
10357 assert(syntax[rule_name] ~= nil,
10358 [[Rule]] .. rule_name
10359 .. [[-> ... does not exist in markdown grammar]])
10360 local previous_pattern
10361 local extension_name
10362 if walkable_syntax[rule_name] then
10363 local previous_accountable_pattern
10364 = walkable_syntax[rule_name][1]
10365 previous_pattern = previous_accountable_pattern[1]
10366 extension_name
10367 = previous_accountable_pattern[2]
10368 .. ", " .. current_extension_name
10369 else
10370 previous_pattern = nil
10371 extension_name = current_extension_name
10372 end
10373 local pattern

```

Instead of a function, a PEG pattern `pattern` may also be supplied with roughly the same effect as supplying the following function, which will define `walkable_syntax[left-hand side terminal symbol]` unless it has been previously defined.

```

function(previous_pattern)
 assert(previous_pattern == nil)
 return pattern
end

```

```

10374 if type(get_pattern) == "function" then
10375 pattern = get_pattern(previous_pattern)
10376 else
10377 assert(previous_pattern == nil,
10378 [[Rule]] .. rule_name ..
10379 [[has already been updated by]] .. extension_name)
10380 pattern = get_pattern
10381 end
10382 local accountable_pattern = { pattern, extension_name, rule_name }
10383 walkable_syntax[rule_name] = { accountable_pattern }
10384 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.

```

10385 local special_characters = {}
10386 self.add_special_character = function(c)
10387 table.insert(special_characters, c)
10388 syntax.SpecialChar = S(table.concat(special_characters, ""))
10389 end
10390
10391 self.add_special_character("*")
10392 self.add_special_character("[")
10393 self.add_special_character("]")
10394 self.add_special_character("<")
10395 self.add_special_character("!")
10396 self.add_special_character("\\")

```

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

```

10397 self.initialize_named_group = function(name, value)
10398 local pattern = Ct("")
10399 if value ~= nil then
10400 pattern = pattern / value
10401 end
10402 syntax.InitializeState = syntax.InitializeState
10403 * Cg(pattern, name)
10404 end

```

Add a named group for indentation.

```

10405 self.initialize_named_group("indent_info")

```

Apply syntax extensions.

```

10406 for _, extension in ipairs(extensions) do
10407 current_extension_name = extension.name
10408 extension.extend_writer(writer)
10409 extension.extend_reader(self)
10410 end
10411 current_extension_name = nil

```

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

```

10412 if options.debugExtensions then
10413 local sorted_lhs = {}
10414 for lhs, _ in pairs(walkable_syntax) do
10415 table.insert(sorted_lhs, lhs)
10416 end
10417 table.sort(sorted_lhs)
10418
10419 local output_lines = {"{"}

```

```

10420 for lhs_index, lhs in ipairs(sorted_lhs) do
10421 local encoded_lhs = util.encode_json_string(lhs)
10422 table.insert(output_lines, [[]] .. encoded_lhs .. [[(: []]])
10423 local rule = walkable_syntax[lhs]
10424 for rhs_index, rhs in ipairs(rule) do
10425 local human_readable_rhs
10426 if type(rhs) == "string" then
10427 human_readable_rhs = rhs
10428 else
10429 local pattern_name
10430 if rhs[3] then
10431 pattern_name = rhs[3]
10432 else
10433 pattern_name = "Anonymous Pattern"
10434 end
10435 local extension_name = rhs[2]
10436 human_readable_rhs = pattern_name .. [[(]]
10437 .. extension_name .. [[]]]
10438 end
10439 local encoded_rhs
10440 = util.encode_json_string(human_readable_rhs)
10441 local output_line = [[]] .. encoded_rhs
10442 if rhs_index < #rule then
10443 output_line = output_line .. ","
10444 end
10445 table.insert(output_lines, output_line)
10446 end
10447 local output_line = "]"
10448 if lhs_index < #sorted_lhs then
10449 output_line = output_line .. ","
10450 end
10451 table.insert(output_lines, output_line)
10452 end
10453 table.insert(output_lines, "}")
10454
10455 local output = table.concat(output_lines, "\n")
10456 local output_filename = options.debugExtensionsFileName
10457 local output_file = assert(io.open(output_filename, "w"),
10458 [[Could not open file]] .. output_filename
10459 .. [[for writing]])
10460 assert(output_file:write(output))
10461 assert(output_file:close())
10462 end

```

Materialize [walkable\\_syntax](#) and merge it into [syntax](#) to produce the complete PEG grammar of markdown. Whenever a rule exists in both [walkable\\_syntax](#) and [syntax](#), the rule from [walkable\\_syntax](#) overrides the rule from [syntax](#).

```

10463 for lhs, rule in pairs(walkable_syntax) do
10464 syntax[lhs] = parsers.fail
10465 for _, rhs in ipairs(rule) do
10466 local pattern

```

Although the interface of the `reader->insert_pattern` method does not document this (see Section 2.1.2), we allow the `reader->insert_pattern` and `reader->update_rule` methods to insert not just PEG patterns, but also rule names that reference the PEG grammar of Markdown.

```

10467 if type(rhs) == "string" then
10468 pattern = V(rhs)
10469 else
10470 pattern = rhs[1]
10471 if type(pattern) == "string" then
10472 pattern = V(pattern)
10473 end
10474 end
10475 syntax[lhs] = syntax[lhs] + pattern
10476 end
10477 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.

```

10478 if options.underscores then
10479 self.add_special_character("_")
10480 end
10481
10482 if not options.codeSpans then
10483 syntax.Code = parsers.fail
10484 else
10485 self.add_special_character("`")
10486 end
10487
10488 if not options.html then
10489 syntax.DisplayHtml = parsers.fail
10490 syntax.InlineHtml = parsers.fail
10491 syntax.HtmlEntity = parsers.fail
10492 else
10493 self.add_special_character("&")
10494 end
10495
10496 if options.preserveTabs then
10497 options.stripIndent = false
10498 end
10499
10500 if not options.smartEllipses then

```



```

10501 syntax.Smart = parsers.fail
10502 else
10503 self.add_special_character(".")
10504 end
10505
10506 if not options.relativeReferences then
10507 syntax.AutoLinkRelativeReference = parsers.fail
10508 end
10509
10510 if options.contentLevel == "inline" then
10511 syntax[1] = "Inlines"
10512 syntax.Inlines = V("InitializeState")
10513 * parsers.Inline^0
10514 * (parsers.spacing^0
10515 * parsers.eof / "")
10516 syntax.Space = parsers.Space + parsers.blankline / writer.space
10517 end
10518
10519 local blocks_nested_t = util.table_copy(syntax)
10520 blocks_nested_t.ExpectedJekyllData = parsers.succeed
10521 parsers.blocks_nested = Ct(blocks_nested_t)
10522
10523 parsers.blocks = Ct(syntax)
10524
10525 local inlines_t = util.table_copy(syntax)
10526 inlines_t[1] = "Inlines"
10527 inlines_t.Inlines = V("InitializeState")
10528 * parsers.Inline^0
10529 * (parsers.spacing^0
10530 * parsers.eof / "")
10531 parsers.inlines = Ct(inlines_t)
10532
10533 local inlines_no_inline_note_t = util.table_copy(inlines_t)
10534 inlines_no_inline_note_t.InlineNote = parsers.fail
10535 parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
10536
10537 local inlines_no_html_t = util.table_copy(inlines_t)
10538 inlines_no_html_t.DisplayHtml = parsers.fail
10539 inlines_no_html_t.InlineHtml = parsers.fail
10540 inlines_no_html_t.HtmlEntity = parsers.fail
10541 parsers.inlines_no_html = Ct(inlines_no_html_t)
10542
10543 local inlines_nbsp_t = util.table_copy(inlines_t)
10544 inlines_nbsp_t.Endline = parsers.NonbreakingEndline
10545 inlines_nbsp_t.Space = parsers.NonbreakingSpace
10546 parsers.inlines_nbsp = Ct(inlines_nbsp_t)
10547

```

```

10548 local inlines_no_link_or_emphasis_t = util.table_copy(inlines_t)
10549 inlines_no_link_or_emphasis_t.LinkAndEmph = parsers.fail
10550 inlines_no_link_or_emphasis_t.EndlineExceptions
10551 = parsers.EndlineExceptions - parsers.eof
10552 parsers.inlines_no_link_or_emphasis
10553 = Ct(inlines_no_link_or_emphasis_t)

```

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

```

10554 return function(input)

```

Unicode-normalize the input.

```

10555 if options.unicodeNormalization then
10556 local form = options.unicodeNormalizationForm
10557 if form == "nfc" then
10558 input = uni_algos.normalize.NFC(input)
10559 elseif form == "nfd" then
10560 input = uni_algos.normalize.NFD(input)
10561 elseif form == "nfkc" then
10562 input = uni_algos.normalize.NFKC(input)
10563 elseif form == "nfkd" then
10564 input = uni_algos.normalize.NFKD(input)
10565 else
10566 return writer.error(
10567 format("Unknown normalization form %s.", form))
10568 end
10569 end

```

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

```

10570 input = input:gsub("\r\n?", "\n")
10571 if input:sub(-1) ~= "\n" then
10572 input = input .. "\n"
10573 end

```

Clear the table of references.

```

10574 references = {}
10575 local document = self.parser_functions.parse_blocks(input)
10576 local output = util.ropetostring(writer.document(document))

```

Remove block element / paragraph separators immediately followed by the output of `writer->undosep`, possibly interleaved by section ends. Then, remove any leftover output of `writer->undosep`.

```

10577 local undosep_start, undosep_end
10578 local potential_secend_start, secend_start
10579 local potential_sep_start, sep_start
10580 while true do
10581 -- find a `writer->undosep`
10582 undosep_start, undosep_end

```

```

10583 = output:find(writer.undosep_text, 1, true)
10584 if undosep_start == nil then break end
10585 -- skip any preceding section ends
10586 secend_start = undosep_start
10587 while true do
10588 potential_secend_start = secend_start - #writer.secend_text
10589 if potential_secend_start < 1
10590 or output:sub(potential_secend_start,
10591 secend_start - 1) ~= writer.secend_text
10592 then
10593 break
10594 end
10595 secend_start = potential_secend_start
10596 end
10597 -- find an immediately preceding
10598 -- block element / paragraph separator
10599 sep_start = secend_start
10600 potential_sep_start = sep_start - #writer.interblocksep_text
10601 if potential_sep_start >= 1
10602 and output:sub(potential_sep_start,
10603 sep_start - 1) == writer.interblocksep_text
10604 then
10605 sep_start = potential_sep_start
10606 else
10607 potential_sep_start = sep_start - #writer.paragraphsep_text
10608 if potential_sep_start >= 1
10609 and output:sub(potential_sep_start,
10610 sep_start - 1) == writer.paragraphsep_text
10611 then
10612 sep_start = potential_sep_start
10613 end
10614 end
10615 -- remove `writer->undosep` and immediately preceding
10616 -- block element / paragraph separator
10617 output = output:sub(1, sep_start - 1)
10618 .. output:sub(secend_start, undosep_start - 1)
10619 .. output:sub(undosep_end + 1)
10620 end
10621 return output
10622 end
10623 end
10624 return self
10625 end

```

### 3.1.7 Built-In Syntax Extensions

Create `extensions` hash table that contains built-in syntax extensions. Syntax

extensions are functions that produce objects with two methods: `extend_writer` and `extend_reader`. The `extend_writer` object takes a `writer` object as the only parameter and mutates it. Similarly, `extend_reader` takes a `reader` object as the only parameter and mutates it.

```
10626 M.extensions = {}
```

### 3.1.7.1 Bracketed Spans

The `extensions.bracketed_spans` function implements the Pandoc bracketed span syntax extension.

```
10627 M.extensions.bracketed_spans = function()
10628 return {
10629 name = "built-in bracketed_spans syntax extension",
10630 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.

```
10631 function self.span(s, attr)
10632 if self.flatten_inlines then return s end
10633 return {"\\markdownRendererBracketedSpanAttributeContextBegin",
10634 self.attributes(attr),
10635 s,
10636 "\\markdownRendererBracketedSpanAttributeContextEnd{}}"}
10637 end
10638 end, extend_reader = function(self)
10639 local parsers = self.parsers
10640 local writer = self.writer
10641
10642 local span_label = parsers.lbracket
10643 * (Cs((parsers.alphanumeric^1
10644 + parsers.inticks
10645 + parsers.autolink
10646 + V("InlineHtml")
10647 + (parsers.backslash * parsers.backslash)
10648 + (parsers.backslash
10649 * (parsers.lbracket + parsers.rbracket)
10650 + V("Space") + V("Endline")
10651 + (parsers.any
10652 - (parsers.newline
10653 + parsers.lbracket
10654 + parsers.rbracket
10655 + parsers.blankline^2))))^1)
10656 / self.parser_functions.parse_inlines)
10657 * parsers.rbracket
10658
10659 local Span = span_label
10660 * Ct(parsers.attributes)
```

```

10661 / writer.span
10662
10663 self.insert_pattern("Inline before LinkAndEmph",
10664 Span, "Span")
10665 end
10666 }
10667 end

```

### 3.1.7.2 Citations

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

```

10668 M.extensions.citations = function(citation_nbsps)
10669 return {
10670 name = "built-in citations syntax extension",
10671 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.

```

10672 function self.citations(text_cites, cites)
10673 local buffer = {}
10674 if self.flatten_inlines then
10675 for _,cite in ipairs(cites) do
10676 if cite.prenote then
10677 table.insert(buffer, {cite.prenote, " "})
10678 end
10679 table.insert(buffer, cite.name)
10680 if cite.postnote then
10681 table.insert(buffer, {" ", cite.postnote})
10682 end
10683 end
10684 else

```

```

10685 table.insert(buffer,
10686 {"\\markdownRenderer",
10687 text_cites and "TextCite" or "Cite",
10688 "{", #cites, "}"})
10689 for _,cite in ipairs(cites) do
10690 table.insert(buffer,
10691 {cite.suppress_author and "-" or "+", "{",
10692 cite.prenote or "", "}{" ,
10693 cite.postnote or "", "}{" , cite.name, "}"})
10694 end
10695 end
10696 return buffer
10697 end
10698 end, extend_reader = function(self)
10699 local parsers = self.parsers
10700 local writer = self.writer
10701
10702 local citation_chars
10703 = parsers.alphanumeric
10704 + S("#$%&-+<>~/_")
10705
10706 local citation_name
10707 = Cs(parsers.dash~1) * parsers.at
10708 * Cs(citation_chars
10709 * (((citation_chars
10710 + parsers.internal_punctuation
10711 - parsers.comma - parsers.semicolon)
10712 * -#((parsers.internal_punctuation
10713 - parsers.comma
10714 - parsers.semicolon)^0
10715 * -(citation_chars
10716 + parsers.internal_punctuation
10717 - parsers.comma
10718 - parsers.semicolon)))^0
10719 * citation_chars)^-1)
10720
10721 local citation_body_prenote
10722 = Cs((parsers.alphanumeric^1
10723 + parsers.bracketed
10724 + parsers.inticks
10725 + parsers.autolink
10726 + V("InlineHtml")
10727 + V("Space") + V("EndlineNoSub")
10728 + (parsers.anyescaped
10729 - (parsers.newline
10730 + parsers.rbracket
10731 + parsers.blankline^2))

```

```

10732 - (parsers.spnl
10733 * parsers.dash~-1
10734 * parsers.at))^1)
10735
10736 local citation_body_postnote
10737 = Cs((parsers.alphanumeric~1
10738 + parsers.bracketed
10739 + parsers.inticks
10740 + parsers.autolink
10741 + V("InlineHtml")
10742 + V("Space") + V("EndlineNoSub")
10743 + (parsers.anyescaped
10744 - (parsers.newline
10745 + parsers.rbracket
10746 + parsers.semicolon
10747 + parsers.blankline~2))
10748 - (parsers.spnl * parsers.rbracket))^1)
10749
10750 local citation_body_chunk
10751 = (citation_body_prenote
10752 * parsers.spnlc_sep
10753 + Cc("")
10754 * parsers.spnlc
10755)
10756 * citation_name
10757 * (parsers.internal_punctuation
10758 - parsers.semicolon)^-1
10759 * (parsers.spnlc / function(_) return end
10760 * citation_body_postnote
10761 + Cc("")
10762 * parsers.spnlc
10763)
10764
10765 local citation_body
10766 = citation_body_chunk
10767 * (parsers.semicolon
10768 * parsers.spnlc
10769 * citation_body_chunk
10770)^0
10771
10772 local citation_headless_body_postnote
10773 = Cs((parsers.alphanumeric~1
10774 + parsers.bracketed
10775 + parsers.inticks
10776 + parsers.autolink
10777 + V("InlineHtml")
10778 + V("Space") + V("Endline"))

```

```

10779 + (parsers.anyescaped
10780 - (parsers.newline
10781 + parsers.rbracket
10782 + parsers.at
10783 + parsers.semicolon + parsers.blankline^2))
10784 - (parsers.spnl * parsers.rbracket))^0)
10785
10786 local citation_headless_body
10787 = citation_headless_body_postnote
10788 * (parsers.semicolon
10789 * parsers.spnlc
10790 * citation_body_chunk
10791)^0
10792
10793 local citations
10794 = function(text_cites, raw_cites)
10795 local function normalize(str)
10796 if str == "" then
10797 str = nil
10798 else
10799 str = (citation_nbsps and
10800 self.parser_functions.parse_inlines_nbsp or
10801 self.parser_functions.parse_inlines)(str)
10802 end
10803 return str
10804 end
10805
10806 local cites = {}
10807 for i = 1,#raw_cites,4 do
10808 cites[#cites+1] = {
10809 prenote = normalize(raw_cites[i]),
10810 suppress_author = raw_cites[i+1] == "-",
10811 name = writer.identifier(raw_cites[i+2]),
10812 postnote = normalize(raw_cites[i+3]),
10813 }
10814 end
10815 return writer.citations(text_cites, cites)
10816 end
10817
10818 local TextCitations
10819 = Ct((parsers.spnlc
10820 * Cc("")
10821 * citation_name
10822 * ((parsers.spnlc
10823 * parsers.lbracket
10824 * citation_headless_body
10825 * parsers.rbracket) + Cc("")))^1)

```



```

10826 / function(raw_cites)
10827 return citations(true, raw_cites)
10828 end
10829
10830 local ParenthesizedCitations
10831 = Ct((parsers.spnlc
10832 * parsers.lbracket
10833 * citation_body
10834 * parsers.rbracket)^1)
10835 / function(raw_cites)
10836 return citations(false, raw_cites)
10837 end
10838
10839 local Citations = TextCitations + ParenthesizedCitations
10840
10841 self.insert_pattern("Inline before LinkAndEmph",
10842 Citations, "Citations")
10843
10844 self.add_special_character("@")
10845 self.add_special_character("-")
10846 end
10847 }
10848 end

```

### 3.1.7.3 Content Blocks

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

```

10849 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.

```

10850 local languages_json = (function()
10851 local base, prev, curr
10852 for _, pathname in ipairs{kpse.lookup(language_map,
10853 {all=true})} do
10854 local file = io.open(pathname, "r")
10855 if not file then goto continue end
10856 local input = assert(file:read("*a"))
10857 assert(file:close())
10858 local json = input:gsub('"[^\\n]-"', '[%1]=')
10859 curr = load("_ENV = {}; return "..json)()
10860 if type(curr) == "table" then
10861 if base == nil then

```

```

10862 base = curr
10863 else
10864 setmetatable(prev, { __index = curr })
10865 end
10866 prev = curr
10867 end
10868 ::continue::
10869 end
10870 return base or {}
10871 end()
10872
10873 return {
10874 name = "built-in content_blocks syntax extension",
10875 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.

```

10876 function self.contentblock(src,suf,type,tit)
10877 if not self.is_writing then return "" end
10878 src = src.." "..suf
10879 suf = suf:lower()
10880 if type == "onlineimage" then
10881 return {"\\markdownRendererContentBlockOnlineImage{" ,suf,"} ",
10882 "{" ,self.string(src),"} ",
10883 "{" ,self.uri(src),"} ",
10884 "{" ,self.string(tit or ""),"} "}
10885 elseif languages_json[suf] then
10886 return {"\\markdownRendererContentBlockCode{" ,suf,"} ",
10887 "{" ,self.string(languages_json[suf]),"} ",
10888 "{" ,self.string(src),"} ",
10889 "{" ,self.uri(src),"} ",
10890 "{" ,self.string(tit or ""),"} "}
10891 else
10892 return {"\\markdownRendererContentBlock{" ,suf,"} ",
10893 "{" ,self.string(src),"} ",
10894 "{" ,self.uri(src),"} ",
10895 "{" ,self.string(tit or ""),"} "}
10896 end
10897 end
10898 end, extend_reader = function(self)
10899 local parsers = self.parsers
10900 local writer = self.writer
10901
10902 local contentblock_tail
10903 = parsers.optionalttitle

```

```

10904 * (parsers.newline + parsers.eof)
10905
10906 -- case insensitive online image suffix:
10907 local onlineimagesuffix
10908 = (function(...)
10909 local parser = nil
10910 for _, suffix in ipairs({...}) do
10911 local pattern=nil
10912 for i=1,#suffix do
10913 local char=suffix:sub(i,i)
10914 char = S(char:lower()..char:upper())
10915 if pattern == nil then
10916 pattern = char
10917 else
10918 pattern = pattern * char
10919 end
10920 end
10921 if parser == nil then
10922 parser = pattern
10923 else
10924 parser = parser + pattern
10925 end
10926 end
10927 return parser
10928 end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
10929
10930 -- online image url for iA Writer content blocks with
10931 -- mandatory suffix, allowing nested brackets:
10932 local onlineimageurl
10933 = (parsers.less
10934 * Cs((parsers.anyescaped
10935 - parsers.more
10936 - parsers.spacing
10937 - #(parsers.period
10938 * onlineimagesuffix
10939 * parsers.more
10940 * contentblock_tail))^0)
10941 * parsers.period
10942 * Cs(onlineimagesuffix)
10943 * parsers.more
10944 + (Cs((parsers.inparens
10945 + (parsers.anyescaped
10946 - parsers.spacing
10947 - parsers.rparent
10948 - #(parsers.period
10949 * onlineimagesuffix
10950 * contentblock_tail))))^0)

```

```

10951 * parsers.period
10952 * Cs(onlineimagesuffix))
10953) * Cc("onlineimage")
10954
10955 -- filename for iA Writer content blocks with mandatory suffix:
10956 local localfilepath
10957 = parsers.slash
10958 * Cs((parsers.anyescaped
10959 - parsers.tab
10960 - parsers.newline
10961 - #(parsers.period
10962 * parsers.alphanumeric^1
10963 * contentblock_tail))^1)
10964 * parsers.period
10965 * Cs(parsers.alphanumeric^1)
10966 * Cc("localfile")
10967
10968 local ContentBlock
10969 = parsers.check_trail_no_rem
10970 * (localfilepath + onlineimageurl)
10971 * contentblock_tail
10972 / writer.contentblock
10973
10974 self.insert_pattern("Block before Blockquote",
10975 ContentBlock, "ContentBlock")
10976 end
10977 }
10978 end

```

### 3.1.7.4 Definition Lists

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

```

10979 M.extensions.definition_lists = function(tight_lists)
10980 return {
10981 name = "built-in definition_lists syntax extension",
10982 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.

```

10983 local function dliitem(term, defs)
10984 local retVal = {"\\markdownRendererDlItem{",term,""}
10985 for _, def in ipairs(defs) do
10986 retVal[#retVal+1]

```

```

10987 = {"\\markdownRendererDlDefinitionBegin ",def,
10988 "\\markdownRendererDlDefinitionEnd "}
10989 end
10990 retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
10991 return retVal
10992 end
10993
10994 function self.definitionlist(items,tight)
10995 if not self.is_writing then return "" end
10996 local buffer = {}
10997 for _,item in ipairs(items) do
10998 buffer[#buffer + 1] = dlistem(item.term, item.definitions)
10999 end
11000 if tight and tight_lists then
11001 return {"\\markdownRendererDlBeginTight\n", buffer,
11002 "\n\\markdownRendererDlEndTight"}
11003 else
11004 return {"\\markdownRendererDlBegin\n", buffer,
11005 "\n\\markdownRendererDlEnd"}
11006 end
11007 end
11008 end, extend_reader = function(self)
11009 local parsers = self.parsers
11010 local writer = self.writer
11011
11012 local defstartchar = S("~:")
11013
11014 local defstart
11015 = parsers.check_trail_length(0) * defstartchar
11016 * #parsers.spacing
11017 * (parsers.tab + parsers.space^-3)
11018 + parsers.check_trail_length(1)
11019 * defstartchar * #parsers.spacing
11020 * (parsers.tab + parsers.space^-2)
11021 + parsers.check_trail_length(2)
11022 * defstartchar * #parsers.spacing
11023 * (parsers.tab + parsers.space^-1)
11024 + parsers.check_trail_length(3)
11025 * defstartchar * #parsers.spacing
11026
11027 local indented_line
11028 = (parsers.check_minimal_indent / "")
11029 * parsers.check_code_trail * parsers.line
11030
11031 local blank
11032 = parsers.check_minimal_blank_indent_and_any_trail
11033 * parsers.optionalspace * parsers.newline

```

```

11034
11035 local dlchunk = Cs(parsers.line * (indented_line - blank)^0)
11036
11037 local indented_blocks = function(bl)
11038 return Cs(bl
11039 * (blank^1 * (parsers.check_minimal_indent / ""))
11040 * parsers.check_code_trail * -parsers.blankline * bl)^0
11041 * (blank^1 + parsers.eof))
11042 end
11043
11044 local function definition_list_item(term, defs, _)
11045 return { term = self.parser_functions.parse_inlines(term),
11046 definitions = defs }
11047 end
11048
11049 local DefinitionListItemLoose
11050 = C(parsers.line) * blank^0
11051 * Ct((parsers.check_minimal_indent * (defstart
11052 * indented_blocks(dlchunk)
11053 / self.parser_functions.parse_blocks_nested))^1)
11054 * Cc(false) / definition_list_item
11055
11056 local DefinitionListItemTight
11057 = C(parsers.line)
11058 * Ct((parsers.check_minimal_indent * (defstart * dlchunk
11059 / self.parser_functions.parse_blocks_nested))^1)
11060 * Cc(true) / definition_list_item
11061
11062 local DefinitionList
11063 = (Ct(DefinitionListItemLoose^1) * Cc(false)
11064 + Ct(DefinitionListItemTight^1)
11065 * (blank^0
11066 * -DefinitionListItemLoose * Cc(true))
11067) / writer.definitionlist
11068
11069 self.insert_pattern("Block after Heading",
11070 DefinitionList, "DefinitionList")
11071 end
11072 }
11073 end

```

### 3.1.7.5 Fancy Lists

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

```

11074 M.extensions.fancy_lists = function()
11075 return {

```

```

11076 name = "built-in fancy_lists syntax extension",
11077 extend_writer = function(self)
11078 local options = self.options
11079

```

Define `writer->fancylis`t 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.

```

11080 function self.fancylis(items,tight,startnum,numstyle,numdelim)
11081 if not self.is_writing then return "" end
11082 local buffer = {}
11083 local num = startnum
11084 for _,item in ipairs(items) do
11085 if item ~= "" then
11086 buffer[#buffer + 1] = self.fancyitem(item,num)
11087 end
11088 if num ~= nil and item ~= "" then
11089 num = num + 1
11090 end
11091 end
11092 local contents = util.intersperse(buffer,"\n")
11093 if tight and options.tightLists then
11094 return {"\\markdownRenderFancy01BeginTight{",
11095 numstyle,"}{",numdelim,"}",contents,
11096 "\\markdownRenderFancy01EndTight "}
11097 else
11098 return {"\\markdownRenderFancy01Begin{",
11099 numstyle,"}{",numdelim,"}",contents,
11100 "\\markdownRenderFancy01End "}

```

```

11101 end
11102 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.

```

11103 function self.fancyitem(s,num)
11104 if num ~= nil then
11105 return {"\\markdownRendererFancyOliItemWithNumber{",num,"}",s,
11106 "\\markdownRendererFancyOliItemEnd "}
11107 else
11108 return {"\\markdownRendererFancyOliItem ",s,
11109 "\\markdownRendererFancyOliItemEnd "}
11110 end
11111 end
11112 end, extend_reader = function(self)
11113 local parsers = self.parsers
11114 local options = self.options
11115 local writer = self.writer
11116
11117 local function combine_markers_and_delims(markers, delims)
11118 local markers_table = {}
11119 for _,marker in ipairs(markers) do
11120 local start_marker
11121 local continuation_marker
11122 if type(marker) == "table" then
11123 start_marker = marker[1]
11124 continuation_marker = marker[2]
11125 else
11126 start_marker = marker
11127 continuation_marker = marker
11128 end
11129 for _,delim in ipairs(delims) do
11130 table.insert(markers_table,
11131 {start_marker, continuation_marker, delim})
11132 end
11133 end
11134 return markers_table
11135 end
11136
11137 local function join_table_with_func(func, markers_table)
11138 local pattern = func(table.unpack(markers_table[1]))
11139 for i = 2, #markers_table do
11140 pattern = pattern + func(table.unpack(markers_table[i]))
11141 end
11142 return pattern
11143 end

```



```

11144
11145 local lowercase_letter_marker = R("az")
11146 local uppercase_letter_marker = R("AZ")
11147
11148 local roman_marker = function(chars)
11149 local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])
11150 local l, x, v, i
11151 = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])
11152 return m^-3
11153 * (c*m + c*d + d^-1 * c^-3)
11154 * (x*c + x*l + l^-1 * x^-3)
11155 * (i*x + i*v + v^-1 * i^-3)
11156 end
11157
11158 local lowercase_roman_marker
11159 = roman_marker({"m", "d", "c", "l", "x", "v", "i"})
11160 local uppercase_roman_marker
11161 = roman_marker({"M", "D", "C", "L", "X", "V", "I"})
11162
11163 local lowercase_opening_roman_marker = P("i")
11164 local uppercase_opening_roman_marker = P("I")
11165
11166 local digit_marker = parsers.dig * parsers.dig^-8
11167
11168 local markers = {
11169 {lowercase_opening_roman_marker, lowercase_roman_marker},
11170 {uppercase_opening_roman_marker, uppercase_roman_marker},
11171 lowercase_letter_marker,
11172 uppercase_letter_marker,
11173 lowercase_roman_marker,
11174 uppercase_roman_marker,
11175 digit_marker
11176 }
11177
11178 local delims = {
11179 parsers.period,
11180 parsers.rparent
11181 }
11182
11183 local markers_table = combine_markers_and_delims(markers, delims)
11184
11185 local function enumerator(start_marker, _,
11186 delimiter_type, interrupting)
11187 local delimiter_range
11188 local allowed_end
11189 if interrupting then
11190 delimiter_range = P("1")

```

```

11191 allowed_end = C(parsers.spacechar~1) * #parsers.linechar
11192 else
11193 delimiter_range = start_marker
11194 allowed_end = C(parsers.spacechar~1)
11195 + #(parsers.newline + parsers.eof)
11196 end
11197
11198 return parsers.check_trail
11199 * Ct(C(delimiter_range) * C(delimiter_type))
11200 * allowed_end
11201 end
11202
11203 local starter = join_table_with_func(enumerator, markers_table)
11204
11205 local TightListItem = function(starter)
11206 return parsers.add_indent(starter, "li")
11207 * parsers.indented_content_tight
11208 end
11209
11210 local LooseListItem = function(starter)
11211 return parsers.add_indent(starter, "li")
11212 * parsers.indented_content_loose
11213 * remove_indent("li")
11214 end
11215
11216 local function roman2number(roman)
11217 local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100,
11218 ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
11219 local numeral = 0
11220
11221 local i = 1
11222 local len = string.len(roman)
11223 while i < len do
11224 local z1, z2 = romans[string.sub(roman, i, i)],
11225 romans[string.sub(roman, i+1, i+1)]
11226 if z1 < z2 then
11227 numeral = numeral + (z2 - z1)
11228 i = i + 2
11229 else
11230 numeral = numeral + z1
11231 i = i + 1
11232 end
11233 end
11234 if i <= len then
11235 numeral = numeral + romans[string.sub(roman,i,i)]
11236 end
11237 return numeral

```

```

11238 end
11239
11240 local function sniffstyle(numstr, delimend)
11241 local numdelim
11242 if delimend == ")" then
11243 numdelim = "OneParen"
11244 elseif delimend == "." then
11245 numdelim = "Period"
11246 else
11247 numdelim = "Default"
11248 end
11249
11250 local num
11251 num = numstr:match("^([I])$")
11252 if num then
11253 return roman2number(num), "UpperRoman", numdelim
11254 end
11255 num = numstr:match("^([i])$")
11256 if num then
11257 return roman2number(string.upper(num)), "LowerRoman", numdelim
11258 end
11259 num = numstr:match("^([A-Z])$")
11260 if num then
11261 return string.byte(num) - string.byte("A") + 1,
11262 "UpperAlpha", numdelim
11263 end
11264 num = numstr:match("^([a-z])$")
11265 if num then
11266 return string.byte(num) - string.byte("a") + 1,
11267 "LowerAlpha", numdelim
11268 end
11269 num = numstr:match("^([IVXLCDM]+)")
11270 if num then
11271 return roman2number(num), "UpperRoman", numdelim
11272 end
11273 num = numstr:match("^([ivxlc dm]+)")
11274 if num then
11275 return roman2number(string.upper(num)), "LowerRoman", numdelim
11276 end
11277 return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
11278 end
11279
11280 local function fancylist(items,tight,start)
11281 local startnum, numstyle, numdelim
11282 = sniffstyle(start[2][1], start[2][2])
11283 return writer.fancylist(items,tight,
11284 options.startNumber and startnum or 1,

```

```

11285 numstyle or "Decimal",
11286 numdelim or "Default")
11287 end
11288
11289 local FancyListOfType
11290 = function(start_marker, continuation_marker, delimiter_type)
11291 local enumerator_start
11292 = enumerator(start_marker, continuation_marker,
11293 delimiter_type)
11294 local enumerator_cont
11295 = enumerator(continuation_marker, continuation_marker,
11296 delimiter_type)
11297 return Cg(enumerator_start, "listtype")
11298 * (Ct(TightListItem(Cb("listtype"))
11299 * ((parsers.check_minimal_indent / "")
11300 * TightListItem(enumerator_cont))^0)
11301 * Cc(true)
11302 * -#((parsers.conditionally_indented_blankline^0 / "")
11303 * parsers.check_minimal_indent * enumerator_cont)
11304 + Ct(LooseListItem(Cb("listtype"))
11305 * ((parsers.conditionally_indented_blankline^0 / "")
11306 * (parsers.check_minimal_indent / "")
11307 * LooseListItem(enumerator_cont))^0)
11308 * Cc(false)
11309) * Ct(Cb("listtype")) / fancylist
11310 end
11311
11312 local FancyList
11313 = join_table_with_func(FancyListOfType, markers_table)
11314
11315 local ListStarter = starter
11316
11317 self.update_rule("OrderedList", FancyList)
11318 self.update_rule("ListStarter", ListStarter)
11319 end
11320 }
11321 end

```

### 3.1.7.6 Fenced Code

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

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

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

```

11322 M.extensions.fenced_code = function(blank_before_code_fence,
11323 allow_attributes,
11324 allow_raw_blocks)
11325 return {
11326 name = "built-in fenced_code syntax extension",
11327 extend_writer = function(self)
11328 local options = self.options
11329

```

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.

```

11330 function self.fencedCode(s, i, attr)
11331 if not self.is_writing then return "" end
11332 s = s:gsub("\n$", "")
11333 local buf = {}
11334 if attr ~= nil then
11335 table.insert(buf,
11336 {"\\markdownRendererFencedCodeAttributeContextBegin",
11337 self.attributes(attr)})
11338 end
11339 local name = util.cache_verbatim(options.cacheDir, s)
11340 table.insert(buf,
11341 {"\\markdownRendererInputFencedCode{",
11342 name,"}{",self.string(i),"}{",self.infostring(i),"}")})
11343 if attr ~= nil then
11344 table.insert(buf,
11345 "\\markdownRendererFencedCodeAttributeContextEnd{")
11346 end
11347 return buf
11348 end
11349

```

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

```

11350 if allow_raw_blocks then
11351 function self.rawBlock(s, attr)
11352 if not self.is_writing then return "" end
11353 s = s:gsub("\n$", "")
11354 local name = util.cache_verbatim(options.cacheDir, s)
11355 return {"\\markdownRendererInputRawBlock{",
11356 name,"}{", self.string(attr),"}")
11357 end
11358 end
11359 end, extend_reader = function(self)
11360 local parsers = self.parsers

```

```

11361 local writer = self.writer
11362
11363 local function captures_geq_length(_,i,a,b)
11364 return #a >= #b and i
11365 end
11366
11367 local function strip_enclosing_whitespaces(str)
11368 return str:gsub("^%s*(.)%s*$", "%1")
11369 end
11370
11371 local tilde_infostring = Cs(Cs((V("HtmlEntity")
11372 + parsers.anyescaped
11373 - parsers.newline)^0)
11374 / strip_enclosing_whitespaces)
11375
11376 local backtick_infostring
11377 = Cs(Cs((V("HtmlEntity")
11378 + (-#(parsers.backslash * parsers.backtick)
11379 * parsers.anyescaped)
11380 - parsers.newline
11381 - parsers.backtick)^0)
11382 / strip_enclosing_whitespaces)
11383
11384 local fenceindent
11385
11386 local function has_trail(indent_table)
11387 return indent_table ~= nil and
11388 indent_table.trail ~= nil and
11389 next(indent_table.trail) ~= nil
11390 end
11391
11392 local function has_indents(indent_table)
11393 return indent_table ~= nil and
11394 indent_table.indents ~= nil and
11395 next(indent_table.indents) ~= nil
11396 end
11397
11398 local function get_last_indent_name(indent_table)
11399 if has_indents(indent_table) then
11400 return indent_table.indents[#indent_table.indents].name
11401 end
11402 end
11403
11404 local count_fenced_start_indent =
11405 function(_, _, indent_table, trail)
11406 local last_indent_name = get_last_indent_name(indent_table)
11407 fenceindent = 0

```

```

11408 if last_indent_name ~= "li" then
11409 fenceindent = #trail
11410 end
11411 return true
11412 end
11413
11414 local fencehead = function(char, infostring)
11415 return Cmt(Cb("indent_info")
11416 * parsers.check_trail, count_fenced_start_indent)
11417 * Cg(char^3, "fencelength")
11418 * parsers.optionalspace
11419 * infostring
11420 * (parsers.newline + parsers.eof)
11421 end
11422
11423 local fencetail = function(char)
11424 return parsers.check_trail_no_rem
11425 * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
11426 * parsers.optionalspace * (parsers.newline + parsers.eof)
11427 + parsers.eof
11428 end
11429
11430 local process_fenced_line =
11431 function(s, i, -- luacheck: ignore s i
11432 indent_table, line_content, is_blank)
11433 local remainder = ""
11434 if has_trail(indent_table) then
11435 remainder = indent_table.trail.internal_remainder
11436 end
11437
11438 if is_blank
11439 and get_last_indent_name(indent_table) == "li" then
11440 remainder = ""
11441 end
11442
11443 local str = remainder .. line_content
11444 local index = 1
11445 local remaining = fenceindent
11446
11447 while true do
11448 local c = str:sub(index, index)
11449 if c == " " and remaining > 0 then
11450 remaining = remaining - 1
11451 index = index + 1
11452 elseif c == "\t" and remaining > 3 then
11453 remaining = remaining - 4
11454 index = index + 1

```

```

11455 else
11456 break
11457 end
11458 end
11459
11460 return true, str:sub(index)
11461 end
11462
11463 local fencedline = function(char)
11464 return Cmt(Cb("indent_info")
11465 * C(parsers.line - fencetail(char))
11466 * Cc(false), process_fenced_line)
11467 end
11468
11469 local blankfencedline
11470 = Cmt(Cb("indent_info")
11471 * C(parsers.blankline)
11472 * Cc(true), process_fenced_line)
11473
11474 local TildeFencedCode
11475 = fencehead(parsers.tilde, tilde_infostring)
11476 * Cs(((parsers.check_minimal_blank_indent / "")
11477 * blankfencedline
11478 + (parsers.check_minimal_indent / "")
11479 * fencedline(parsers.tilde))^0)
11480 * ((parsers.check_minimal_indent / "")
11481 * fencetail(parsers.tilde) + parsers.succeed)
11482
11483 local BacktickFencedCode
11484 = fencehead(parsers.backtick, backtick_infostring)
11485 * Cs(((parsers.check_minimal_blank_indent / "")
11486 * blankfencedline
11487 + (parsers.check_minimal_indent / "")
11488 * fencedline(parsers.backtick))^0)
11489 * ((parsers.check_minimal_indent / "")
11490 * fencetail(parsers.backtick) + parsers.succeed)
11491
11492 local infostring_with_attributes
11493 = Ct(C((parsers.linechar
11494 - (parsers.optionalspace
11495 * parsers.attributes))^0)
11496 * parsers.optionalspace
11497 * Ct(parsers.attributes))
11498
11499 local FencedCode
11500 = ((TildeFencedCode + BacktickFencedCode)
11501 / function(infostring, code)

```



```

11502 local expanded_code = self.expandtabs(code)
11503
11504 if allow_raw_blocks then
11505 local raw_attr = lpeg.match(parsers.raw_attribute,
11506 infostring)
11507 if raw_attr then
11508 return writer.rawBlock(expanded_code, raw_attr)
11509 end
11510 end
11511
11512 local attr = nil
11513 if allow_attributes then
11514 local match = lpeg.match(infostring_with_attributes,
11515 infostring)
11516 if match then
11517 infostring, attr = table.unpack(match)
11518 end
11519 end
11520 return writer.fencedCode(expanded_code, infostring, attr)
11521 end)
11522
11523 self.insert_pattern("Block after Verbatim",
11524 FencedCode, "FencedCode")
11525
11526 local fencestart
11527 if blank_before_code_fence then
11528 fencestart = parsers.fail
11529 else
11530 fencestart = fencehead(parsers.backtick, backtick_infostring)
11531 + fencehead(parsers.tilde, tilde_infostring)
11532 end
11533
11534 self.update_rule("EndlineExceptions", function(previous_pattern)
11535 if previous_pattern == nil then
11536 previous_pattern = parsers.EndlineExceptions
11537 end
11538 return previous_pattern + fencestart
11539 end)
11540
11541 self.add_special_character("`")
11542 self.add_special_character("~")
11543 end
11544 }
11545 end

```

### 3.1.7.7 Fenced Divs

The `extensions.fenced_divs` function implements the Pandoc fenced div syntax extension. When the `blank_before_div_fence` parameter is `true`, the syntax extension requires a blank line between a paragraph and the following fenced code block.

```
11546 M.extensions.fenced_divs = function(blank_before_div_fence)
11547 return {
11548 name = "built-in fenced_divs syntax extension",
11549 extend_writer = function(self)
```

Define `writer->div_begin` as a function that will transform the beginning of an input fenced div with attributes `attributes` to the output format.

```
11550 function self.div_begin(attributes)
11551 local start_output
11552 = {"\\markdownRendererFencedDivAttributeContextBegin\n",
11553 self.attributes(attributes)}
11554 local end_output
11555 = {"\\markdownRendererFencedDivAttributeContextEnd{}}"}
11556 return self.push_attributes(
11557 "div", attributes, start_output, end_output)
11558 end
```

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

```
11559 function self.div_end()
11560 return self.pop_attributes("div")
11561 end
11562 end, extend_reader = function(self)
11563 local parsers = self.parsers
11564 local writer = self.writer
```

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

```
11565 local fenced_div_infostring
11566 = C((parsers.linechar
11567 - (parsers.spacechar^1
11568 * parsers.colon^1))^1)
11569
11570 local fenced_div_begin = parsers.nonindentspace
11571 * parsers.colon^3
11572 * parsers.optionalspace
11573 * fenced_div_infostring
11574 * (parsers.spacechar^1
11575 * parsers.colon^1)^0
11576 * parsers.optionalspace
11577 * (parsers.newline + parsers.eof)
11578
11579 local fenced_div_end = parsers.nonindentspace
11580 * parsers.colon^3
```

```

11581 * parsers.optionalspace
11582 * (parsers.newline + parsers.eof)

```

Initialize a named group named `fenced_div_level` for tracking how deep we are nested in divs and the named group `fenced_div_num_opening_indents` for tracking the indent of the starting div fence. The former named group is immutable and should roll back properly when we fail to match a fenced div. The latter is mutable and may contain items from unsuccessful matches on top. However, we always know how many items at the head of the latter we can trust by consulting the former.

```

11583 self.initialize_named_group("fenced_div_level", "0")
11584 self.initialize_named_group("fenced_div_num_opening_indents")
11585
11586 local function increment_div_level()
11587 local push_indent_table =
11588 function(s, i, indent_table, -- luacheck: ignore s i
11589 fenced_div_num_opening_indents, fenced_div_level)
11590 fenced_div_level = tonumber(fenced_div_level) + 1
11591 local num_opening_indents = 0
11592 if indent_table.indents ~= nil then
11593 num_opening_indents = #indent_table.indents
11594 end
11595 fenced_div_num_opening_indents[fenced_div_level]
11596 = num_opening_indents
11597 return true, fenced_div_num_opening_indents
11598 end
11599
11600 local increment_level =
11601 function(s, i, fenced_div_level) -- luacheck: ignore s i
11602 fenced_div_level = tonumber(fenced_div_level) + 1
11603 return true, tostring(fenced_div_level)
11604 end
11605
11606 return Cg(Cmt(Cb("indent_info")
11607 * Cb("fenced_div_num_opening_indents")
11608 * Cb("fenced_div_level"), push_indent_table)
11609 , "fenced_div_num_opening_indents")
11610 * Cg(Cmt(Cb("fenced_div_level"), increment_level)
11611 , "fenced_div_level")
11612 end
11613
11614 local function decrement_div_level()
11615 local pop_indent_table =
11616 function(s, i, -- luacheck: ignore s i
11617 fenced_div_indent_table, fenced_div_level)
11618 fenced_div_level = tonumber(fenced_div_level)
11619 fenced_div_indent_table[fenced_div_level] = nil
11620 return true, tostring(fenced_div_level - 1)

```

```

11621 end
11622
11623 return Cg(Cmt(Cb("fenced_div_num_opening_indents")
11624 * Cb("fenced_div_level"), pop_indent_table)
11625 , "fenced_div_level")
11626 end
11627
11628
11629 local non_fenced_div_block
11630 = parsers.check_minimal_indent * V("Block")
11631 - parsers.check_minimal_indent_and_trail * fenced_div_end
11632
11633 local non_fenced_div_paragraph
11634 = parsers.check_minimal_indent * V("Paragraph")
11635 - parsers.check_minimal_indent_and_trail * fenced_div_end
11636
11637 local blank = parsers.minimally_indented_blank
11638
11639 local block_separated = parsers.block_sep_group(blank)
11640 * non_fenced_div_block
11641
11642 local loop_body_pair
11643 = parsers.create_loop_body_pair(block_separated,
11644 non_fenced_div_paragraph,
11645 parsers.block_sep_group(blank),
11646 parsers.par_sep_group(blank))
11647
11648 local content_loop = (non_fenced_div_block
11649 * loop_body_pair.block^0
11650 + non_fenced_div_paragraph
11651 * block_separated
11652 * loop_body_pair.block^0
11653 + non_fenced_div_paragraph
11654 * loop_body_pair.par^0)
11655 * blank^0
11656
11657 local FencedDiv = fenced_div_begin
11658 / function (infostring)
11659 local attr
11660 = lpeg.match(Ct(parsers.attributes),
11661 infostring)
11662 if attr == nil then
11663 attr = {"." .. infostring}
11664 end
11665 return attr
11666 end
11667 / writer.div_begin

```

```

11668 * increment_div_level()
11669 * parsers.skipblanklines
11670 * Ct(content_loop)
11671 * parsers.minimally_indented_blank^0
11672 * parsers.check_minimal_indent_and_trail
11673 * fenced_div_end
11674 * decrement_div_level()
11675 * (Cc("") / writer.div_end)
11676
11677 self.insert_pattern("Block after Verbatim",
11678 FencedDiv, "FencedDiv")
11679
11680 self.add_special_character(":".")
11681

```

If the `blank_before_div_fence` parameter is `false`, we will have the closing div at the beginning of a line break the current paragraph if we are currently nested in a div and the indentation matches the opening div fence.

```

11682 local function is_inside_div()
11683 local check_div_level =
11684 function(s, i, fenced_div_level) -- luacheck: ignore s i
11685 fenced_div_level = tonumber(fenced_div_level)
11686 return fenced_div_level > 0
11687 end
11688
11689 return Cmt(Cb("fenced_div_level"), check_div_level)
11690 end
11691
11692 local function check_indent()
11693 local compare_indent =
11694 function(s, i, indent_table, -- luacheck: ignore s i
11695 fenced_div_num_opening_indents, fenced_div_level)
11696 fenced_div_level = tonumber(fenced_div_level)
11697 local num_current_indents
11698 = (indent_table.current_line_indents ~= nil and
11699 #indent_table.current_line_indents) or 0
11700 local num_opening_indents
11701 = fenced_div_num_opening_indents[fenced_div_level]
11702 return num_current_indents == num_opening_indents
11703 end
11704
11705 return Cmt(Cb("indent_info")
11706 * Cb("fenced_div_num_opening_indents")
11707 * Cb("fenced_div_level"), compare_indent)
11708 end
11709
11710 local fencestart = is_inside_div()

```

```

11711 * fenced_div_end
11712 * check_indent()
11713
11714 if not blank_before_div_fence then
11715 self.update_rule("EndlineExceptions", function(previous_pattern)
11716 if previous_pattern == nil then
11717 previous_pattern = parsers.EndlineExceptions
11718 end
11719 return previous_pattern + fencestart
11720 end)
11721 end
11722 end
11723 }
11724 end

```

### 3.1.7.8 Header Attributes

The `extensions.header_attributes` function implements the Pandoc header attribute syntax extension.

```

11725 M.extensions.header_attributes = function()
11726 return {
11727 name = "built-in header_attributes syntax extension",
11728 extend_writer = function()
11729 end, extend_reader = function(self)
11730 local parsers = self.parsers
11731 local writer = self.writer
11732
11733 local function strip_atx_end(s)
11734 return s:gsub("%s+##%s*$", "")
11735 end
11736
11737 local AtxHeading = Cg(parsers.heading_start, "level")
11738 * parsers.optionalspace
11739 * (C(((parsers.linechar
11740 - (parsers.attributes
11741 * parsers.optionalspace
11742 * parsers.newline))
11743 * (parsers.linechar
11744 - parsers.lbrace)^0)^1)
11745 / strip_atx_end
11746 / parsers.parse_heading_text)
11747 * Cg(Ct(parsers.newline
11748 + (parsers.attributes
11749 * parsers.optionalspace
11750 * parsers.newline)), "attributes")
11751 * Cb("level")
11752 * Cb("attributes")

```

```

11753 / writer.heading
11754
11755 local function strip_trailing_spaces(s)
11756 return s:gsub("%s*$","")
11757 end
11758
11759 local heading_line = (parsers.linechar
11760 - (parsers.attributes
11761 * parsers.optionalspace
11762 * parsers.newline))^1
11763 - parsers.thematic_break_lines
11764
11765 local heading_text
11766 = heading_line
11767 * ((V("Endline") / "\n")
11768 * (heading_line - parsers.heading_level))^0
11769 * parsers.newline^-1
11770
11771 local SettextHeading
11772 = parsers.freeze_trail * parsers.check_trail_no_rem
11773 * #(heading_text
11774 * (parsers.attributes
11775 * parsers.optionalspace
11776 * parsers.newline)^-1
11777 * parsers.check_minimal_indent
11778 * parsers.check_trail
11779 * parsers.heading_level)
11780 * Cs(heading_text) / strip_trailing_spaces
11781 / parsers.parse_heading_text
11782 * Cg(Ct((parsers.attributes
11783 * parsers.optionalspace
11784 * parsers.newline)^-1), "attributes")
11785 * parsers.check_minimal_indent_and_trail * parsers.heading_level
11786 * Cb("attributes")
11787 * parsers.newline
11788 * parsers.unfreeze_trail
11789 / writer.heading
11790
11791 local Heading = AtxHeading + SettextHeading
11792 self.update_rule("Heading", Heading)
11793 end
11794 }
11795 end

```

### 3.1.7.9 Inline Code Attributes

The `extensions.inline_code_attributes` function implements the Pandoc in-line code attribute syntax extension.

```
11796 M.extensions.inline_code_attributes = function()
11797 return {
11798 name = "built-in inline_code_attributes syntax extension",
11799 extend_writer = function()
11800 end, extend_reader = function(self)
11801 local writer = self.writer
11802
11803 local CodeWithAttributes = parsers.inticks
11804 * Ct(parsers.attributes)
11805 / writer.code
11806
11807 self.insert_pattern("Inline before Code",
11808 CodeWithAttributes,
11809 "CodeWithAttributes")
11810 end
11811 }
11812 end
```

### 3.1.7.10 Line Blocks

The `extensions.line_blocks` function implements the Pandoc line block syntax extension.

```
11813 M.extensions.line_blocks = function()
11814 return {
11815 name = "built-in line_blocks syntax extension",
11816 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.

```
11817 function self.lineblock(lines)
11818 if not self.is_writing then return "" end
11819 local buffer = {}
11820 for i = 1, #lines - 1 do
11821 buffer[#buffer + 1] = { lines[i], self.hard_line_break }
11822 end
11823 buffer[#buffer + 1] = lines[#lines]
11824
11825 return {"\\markdownRendererLineBlockBegin\n"
11826 ,buffer,
11827 "\n\\markdownRendererLineBlockEnd "}
11828 end
11829 end, extend_reader = function(self)
11830 local parsers = self.parsers
11831 local writer = self.writer
11832 end
```



```

11833 local LineBlock
11834 = Ct((Cs(((parsers.pipe * parsers.space) / ""
11835 * ((parsers.space)/entities.char_entity("nbsp"))^0
11836 * parsers.linechar^0 * (parsers.newline/"")
11837 * (-parsers.pipe
11838 * (parsers.space^1/" ")
11839 * parsers.linechar^1
11840 * (parsers.newline/"")
11841)^0
11842 * (parsers.blankline/"")^0)
11843 / self.parser_functions.parse_inlines)^1)
11844 / writer.lineblock
11845
11846 self.insert_pattern("Block after Blockquote",
11847 LineBlock, "LineBlock")
11848 end
11849 }
11850 end

```

### 3.1.7.11 Marked text

The `extensions.mark` function implements the Pandoc mark syntax extension.

```

11851 M.extensions.mark = function()
11852 return {
11853 name = "built-in mark syntax extension",
11854 extend_writer = function(self)

```

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

```

11855 function self.mark(s)
11856 if self.flatten_inlines then return s end
11857 return {"\\markdownRendererMark{", s, "}" }
11858 end
11859 end, extend_reader = function(self)
11860 local parsers = self.parsers
11861 local writer = self.writer
11862
11863 local doubleequals = P("==")
11864
11865 local Mark
11866 = parsers.between(V("Inline"), doubleequals, doubleequals)
11867 / function (inlines) return writer.mark(inlines) end
11868
11869 self.add_special_character("=")
11870 self.insert_pattern("Inline before LinkAndEmph",
11871 Mark, "Mark")
11872 end
11873 }

```

11874 end

### 3.1.7.12 Link Attributes

The `extensions.link_attributes` function implements the Pandoc link attribute syntax extension.

```
11875 M.extensions.link_attributes = function()
11876 return {
11877 name = "built-in link_attributes syntax extension",
11878 extend_writer = function()
11879 end, extend_reader = function(self)
11880 local parsers = self.parsers
11881 local options = self.options
11882
```

The following patterns define link reference definitions with attributes.

```
11883 local define_reference_parser
11884 = (parsers.check_trail / "")
11885 * parsers.link_label
11886 * parsers.colon
11887 * parsers.spnlc * parsers.url
11888 * (parsers.spnlc_sep * parsers.title
11889 * (parsers.spnlc * Ct(parsers.attributes))
11890 * parsers.only_blank
11891 + parsers.spnlc_sep * parsers.title * parsers.only_blank
11892 + Cc("") * (parsers.spnlc * Ct(parsers.attributes))
11893 * parsers.only_blank
11894 + Cc("") * parsers.only_blank)
11895
11896 local ReferenceWithAttributes = define_reference_parser
11897 / self.register_link
11898
11899 self.update_rule("Reference", ReferenceWithAttributes)
11900
```

The following patterns define direct and indirect links with attributes.

```
11901
11902 local LinkWithAttributesAndEmph
11903 = Ct(parsers.link_and_emph_table * Cg(Cc(true),
11904 "match_link_attributes"))
11905 / self.defer_link_and_emphasis_processing
11906
11907 self.update_rule("LinkAndEmph", LinkWithAttributesAndEmph)
11908
```

The following patterns define autolinks with attributes.

```
11909 local AutoLinkUrlWithAttributes
11910 = parsers.auto_link_url
```

```

11911 * Ct(parsers.attributes)
11912 / self.auto_link_url
11913
11914 self.insert_pattern("Inline before AutoLinkUrl",
11915 AutoLinkUrlWithAttributes,
11916 "AutoLinkUrlWithAttributes")
11917
11918 local AutoLinkEmailWithAttributes
11919 = parsers.auto_link_email
11920 * Ct(parsers.attributes)
11921 / self.auto_link_email
11922
11923 self.insert_pattern("Inline before AutoLinkEmail",
11924 AutoLinkEmailWithAttributes,
11925 "AutoLinkEmailWithAttributes")
11926
11927 if options.relativeReferences then
11928
11929 local AutoLinkRelativeReferenceWithAttributes
11930 = parsers.auto_link_relative_reference
11931 * Ct(parsers.attributes)
11932 / self.auto_link_url
11933
11934 self.insert_pattern(
11935 "Inline before AutoLinkRelativeReference",
11936 AutoLinkRelativeReferenceWithAttributes,
11937 "AutoLinkRelativeReferenceWithAttributes")
11938
11939 end
11940
11941 end
11942 }
11943 end

```

### 3.1.7.13 Notes

The `extensions.notes` function implements the Pandoc note and inline note syntax extensions. When the `note` parameter is `true`, the Pandoc note syntax extension will be enabled. When the `inline_notes` parameter is `true`, the Pandoc inline note syntax extension will be enabled.

```

11944 M.extensions.notes = function(notes, inline_notes)
11945 assert(notes or inline_notes)
11946 return {
11947 name = "built-in notes syntax extension",
11948 extend_writer = function(self)

```

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

```
11949 function self.note(s)
11950 if self.flatten_inlines then return "" end
11951 return {"\\markdownRendererNote{",s,""}
11952 end
11953 end, extend_reader = function(self)
11954 local parsers = self.parsers
11955 local writer = self.writer
11956
11957 local rawnotes = parsers.rawnotes
11958
11959 if inline_notes then
11960 local InlineNote
11961 = parsers.circumflex
11962 * (parsers.link_label
11963 / self.parser_functions.parse_inlines_no_inline_note)
11964 / writer.note
11965
11966 self.insert_pattern("Inline after LinkAndEmph",
11967 InlineNote, "InlineNote")
11968 end
11969 if notes then
11970 local function strip_first_char(s)
11971 return s:sub(2)
11972 end
11973
11974 local RawNoteRef
11975 = #(parsers.lbracket * parsers.circumflex)
11976 * parsers.link_label / strip_first_char
11977
11978 -- like indirect_link
11979 local function lookup_note(ref)
11980 return writer.defer_call(function()
11981 local found = rawnotes[self.normalize_tag(ref)]
11982 if found then
11983 return writer.note(
11984 self.parser_functions.parse_blocks_nested(found))
11985 else
11986 return {"[",
11987 self.parser_functions.parse_inlines("^" .. ref), "]" }
11988 end
11989 end)
11990 end
11991
11992 local function register_note(ref,rawnote)
11993 local normalized_tag = self.normalize_tag(ref)
```

```

11994 if rawnotes[normalized_tag] == nil then
11995 rawnotes[normalized_tag] = rawnote
11996 end
11997 return ""
11998 end
11999
12000 local NoteRef = RawNoteRef / lookup_note
12001
12002 local optionally_indented_line
12003 = parsers.check_optional_indent_and_any_trail * parsers.line
12004
12005 local blank
12006 = parsers.check_optional_blank_indent_and_any_trail
12007 * parsers.optionalspace * parsers.newline
12008
12009 local chunk
12010 = Cs(parsers.line
12011 * (optionally_indented_line - blank)^0)
12012
12013 local indented_blocks = function(bl)
12014 return Cs(bl
12015 * (blank^1 * (parsers.check_optional_indent / "")
12016 * parsers.check_code_trail
12017 * -parsers.blankline * bl)^0)
12018 end
12019
12020 local NoteBlock
12021 = parsers.check_trail_no_rem
12022 * RawNoteRef * parsers.colon
12023 * parsers.spnlc * indented_blocks(chunk)
12024 / register_note
12025
12026 self.update_rule("Reference", function(previous_pattern)
12027 if previous_pattern == nil then
12028 previous_pattern = parsers.Reference
12029 end
12030 return NoteBlock + previous_pattern
12031 end)
12032
12033 self.insert_pattern("Inline before LinkAndEmph",
12034 NoteRef, "NoteRef")
12035 end
12036
12037 self.add_special_character("^")
12038 end
12039 }
12040 end

```

### 3.1.7.14 Pipe Tables

The `extensions.pipe_table` function implements the PHP Markdown table syntax extension (also known as pipe tables in Pandoc). When the `table_captions` parameter is `true`, the function also implements the Pandoc table caption syntax extension for table captions. When the `table_attributes` parameter is also `true`, the function also allows attributes to be attached to the (possibly empty) table captions.

```
12041 M.extensions.pipe_tables = function(table_captions, table_attributes)
12042
12043 local function make_pipe_table_rectangular(rows)
12044 local num_columns = #rows[2]
12045 local rectangular_rows = {}
12046 for i = 1, #rows do
12047 local row = rows[i]
12048 local rectangular_row = {}
12049 for j = 1, num_columns do
12050 rectangular_row[j] = row[j] or ""
12051 end
12052 table.insert(rectangular_rows, rectangular_row)
12053 end
12054 return rectangular_rows
12055 end
12056
12057 local function pipe_table_row(allow_empty_first_column
12058 , nonempty_column
12059 , column_separator
12060 , column)
12061 local row_beginning
12062 if allow_empty_first_column then
12063 row_beginning = -- empty first column
12064 #(parsers.spacechar^4
12065 * column_separator)
12066 * parsers.optionalspace
12067 * column
12068 * parsers.optionalspace
12069 -- non-empty first column
12070 + parsers.nonindentSPACE
12071 * nonempty_column~-1
12072 * parsers.optionalspace
12073 else
12074 row_beginning = parsers.nonindentSPACE
12075 * nonempty_column~-1
12076 * parsers.optionalspace
12077 end
12078
12079 return Ct(row_beginning
```

```

12080 * (-- single column with no leading pipes
12081 #(column_separator
12082 * parsers.optionalspace
12083 * parsers.newline)
12084 * column_separator
12085 * parsers.optionalspace
12086 -- single column with leading pipes or
12087 -- more than a single column
12088 + (column_separator
12089 * parsers.optionalspace
12090 * column
12091 * parsers.optionalspace)^1
12092 * (column_separator
12093 * parsers.optionalspace)^-1))
12094 end
12095
12096 return {
12097 name = "built-in pipe_tables syntax extension",
12098 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.

```

12099 function self.table(rows, caption, attributes)
12100 if not self.is_writing then return "" end
12101 local buffer = {}
12102 if attributes ~= nil then
12103 table.insert(buffer,
12104 "\\markdownRendererTableAttributeContextBegin\n")
12105 table.insert(buffer, self.attributes(attributes))
12106 end
12107 table.insert(buffer,
12108 {"\\markdownRendererTable{",
12109 caption or "", "}{" , #rows - 1, "}{" ,
12110 #rows[1], "}"}))
12111 local temp = rows[2] -- put alignments on the first row
12112 rows[2] = rows[1]
12113 rows[1] = temp
12114 for i, row in ipairs(rows) do
12115 table.insert(buffer, "{")
12116 for _, column in ipairs(row) do
12117 if i > 1 then -- do not use braces for alignments
12118 table.insert(buffer, "{")
12119 end
12120 table.insert(buffer, column)
12121 if i > 1 then
12122 table.insert(buffer, "}")
12123 end

```

```

12124 end
12125 table.insert(buffer, "{")
12126 end
12127 if attributes ~= nil then
12128 table.insert(buffer,
12129 "\\markdownRendererTableAttributeContextEnd{}")
12130 end
12131 return buffer
12132 end
12133 end, extend_reader = function(self)
12134 local parsers = self.parsers
12135 local writer = self.writer
12136
12137 local table_hline_separator = parsers.pipe + parsers.plus
12138
12139 local table_hline_column = (parsers.dash
12140 - #(parsers.dash
12141 * (parsers.spacechar
12142 + table_hline_separator
12143 + parsers.newline)))^1
12144 * (parsers.colon * Cc("r")
12145 + parsers.dash * Cc("d"))
12146 + parsers.colon
12147 * (parsers.dash
12148 - #(parsers.dash
12149 * (parsers.spacechar
12150 + table_hline_separator
12151 + parsers.newline)))^1
12152 * (parsers.colon * Cc("c")
12153 + parsers.dash * Cc("l"))
12154
12155 local table_hline = pipe_table_row(false
12156 , table_hline_column
12157 , table_hline_separator
12158 , table_hline_column)
12159
12160 local table_caption_beginning
12161 = (parsers.check_minimal_blank_indent_and_any_trail_no_rem
12162 * parsers.optionalspace * parsers.newline)^0
12163 * parsers.check_minimal_indent_and_trail
12164 * (P("Table")^-1 * parsers.colon)
12165 * parsers.optionalspace
12166
12167 local function strip_trailing_spaces(s)
12168 return s:gsub("%s*$", "")
12169 end
12170

```



```

12171 local table_row
12172 = pipe_table_row(true
12173 , (C((parsers.linechar - parsers.pipe)^1)
12174 / strip_trailing_spaces
12175 / self.parser_functions.parse_inlines)
12176 , parsers.pipe
12177 , (C((parsers.linechar - parsers.pipe)^0)
12178 / strip_trailing_spaces
12179 / self.parser_functions.parse_inlines))
12180
12181 local table_caption
12182 if table_captions then
12183 table_caption = #table_caption_beginning
12184 * table_caption_beginning
12185 if table_attributes then
12186 table_caption = table_caption
12187 * (C((((parsers.linechar
12188 - (parsers.attributes
12189 * parsers.optionalspace
12190 * parsers.newline
12191 * -(parsers.optionalspace
12192 * parsers.linechar))))
12193 + (parsers.newline
12194 * #(parsers.optionalspace
12195 * parsers.linechar)
12196 * C(parsers.optionalspace)
12197 / writer.space))
12198 * (parsers.linechar
12199 - parsers.lbrace)^0)^1)
12200 / self.parser_functions.parse_inlines)
12201 * (parsers.newline
12202 + (Ct(parsers.attributes)
12203 * parsers.optionalspace
12204 * parsers.newline))
12205 else
12206 table_caption = table_caption
12207 * C((parsers.linechar
12208 + (parsers.newline
12209 * #(parsers.optionalspace
12210 * parsers.linechar)
12211 * C(parsers.optionalspace)
12212 / writer.space))^1)
12213 / self.parser_functions.parse_inlines
12214 * parsers.newline
12215 end
12216 else
12217 table_caption = parsers.fail

```

```

12218 end
12219
12220 local PipeTable
12221 = Ct(table_row * parsers.newline
12222 * (parsers.check_minimal_indent_and_trail / {})
12223 * table_hline * parsers.newline
12224 * ((parsers.check_minimal_indent / {})
12225 * table_row * parsers.newline)^0)
12226 / make_pipe_table_rectangular
12227 * table_caption^-1
12228 / writer.table
12229
12230 self.insert_pattern("Block after Blockquote",
12231 PipeTable, "PipeTable")
12232 end
12233 }
12234 end

```

### 3.1.7.15 Raw Attributes

The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```

12235 M.extensions.raw_inline = function()
12236 return {
12237 name = "built-in raw_inline syntax extension",
12238 extend_writer = function(self)
12239 local options = self.options
12240

```

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

```

12241 function self.rawInline(s, attr)
12242 if not self.is_writing then return "" end
12243 if self.flatten_inlines then return s end
12244 local name = util.cache_verbatim(options.cacheDir, s)
12245 return {"\\markdownRendererInputRawInline{",
12246 name,"}{" , self.string(attr),"}" }
12247 end
12248 end, extend_reader = function(self)
12249 local writer = self.writer
12250
12251 local RawInline = parsers.inticks
12252 * parsers.raw_attribute
12253 / writer.rawInline
12254
12255 self.insert_pattern("Inline before Code",
12256 RawInline, "RawInline")
12257 end

```

```

12258 }
12259 end

```

### 3.1.7.16 Strike-Through

The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```

12260 M.extensions.strike_through = function()
12261 return {
12262 name = "built-in strike_through syntax extension",
12263 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.

```

12264 function self.strike_through(s)
12265 if self.flatten_inlines then return s end
12266 return {"\\markdownRendererStrikeThrough{" ,s,"}"}
12267 end
12268 end, extend_reader = function(self)
12269 local parsers = self.parsers
12270 local writer = self.writer
12271
12272 local StrikeThrough = (
12273 parsers.between(parsers.Inline, parsers.doubletildes,
12274 parsers.doubletildes)
12275) / writer.strike_through
12276
12277 self.insert_pattern("Inline after LinkAndEmph",
12278 StrikeThrough, "StrikeThrough")
12279
12280 self.add_special_character("~")
12281 end
12282 }
12283 end

```

### 3.1.7.17 Subscripts

The `extensions.subscripts` function implements the Pandoc subscript syntax extension.

```

12284 M.extensions.subscripts = function()
12285 return {
12286 name = "built-in subscripts syntax extension",
12287 extend_writer = function(self)

```

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

```

12288 function self.subscript(s)
12289 if self.flatten_inlines then return s end

```

```

12290 return {"\\markdownRendererSubscript{" ,s,""}
12291 end
12292 end, extend_reader = function(self)
12293 local parsers = self.parsers
12294 local writer = self.writer
12295
12296 local Subscript = (
12297 parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
12298) / writer.subscript
12299
12300 self.insert_pattern("Inline after LinkAndEmph",
12301 Subscript, "Subscript")
12302
12303 self.add_special_character("~")
12304 end
12305 }
12306 end

```

### 3.1.7.18 Superscripts

The `extensions.superscripts` function implements the Pandoc superscript syntax extension.

```

12307 M.extensions.superscripts = function()
12308 return {
12309 name = "built-in superscripts syntax extension",
12310 extend_writer = function(self)

```

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

```

12311 function self.superscript(s)
12312 if self.flatten_inlines then return s end
12313 return {"\\markdownRendererSuperscript{" ,s,""}
12314 end
12315 end, extend_reader = function(self)
12316 local parsers = self.parsers
12317 local writer = self.writer
12318
12319 local Superscript = (
12320 parsers.between(parsers.Str, parsers.circumflex,
12321 parsers.circumflex)
12322) / writer.superscript
12323
12324 self.insert_pattern("Inline after LinkAndEmph",
12325 Superscript, "Superscript")
12326
12327 self.add_special_character("^")
12328 end
12329 }

```

```
12330 end
```

### 3.1.7.19 T<sub>E</sub>X Math

The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```
12331 M.extensions.tex_math = function(tex_math_dollars,
12332 tex_math_single_backslash,
12333 tex_math_double_backslash)
12334 return {
12335 name = "built-in tex_math syntax extension",
12336 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.

```
12337 function self.display_math(s)
12338 if self.flatten_inlines then return s end
12339 return {"\\markdownRendererDisplayMath{",self.math(s),"}"}
12340 end
```

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

```
12341 function self.inline_math(s)
12342 if self.flatten_inlines then return s end
12343 return {"\\markdownRendererInlineMath{",self.math(s),"}"}
12344 end
12345 end, extend_reader = function(self)
12346 local parsers = self.parsers
12347 local writer = self.writer
12348
12349 local function between(p, starter, ender)
12350 return (starter * Cs(p * (p - ender)^0) * ender)
12351 end
12352
12353 local function strip_preceding_whitespaces(str)
12354 return str:gsub("^%s*(.-)$", "%1")
12355 end
12356
12357 local allowed_before_closing
12358 = B(parsers.backslash * parsers.any
12359 + parsers.any * (parsers.any - parsers.backslash))
12360
12361 local allowed_before_closing_no_space
12362 = B(parsers.backslash * parsers.any
12363 + parsers.any * (parsers.nonspacechar - parsers.backslash))
12364 end
```

The following patterns implement the Pandoc dollar math syntax extension.

```

12365 local dollar_math_content
12366 = (parsers.newline * (parsers.check_optional_indent / ""))
12367 + parsers.backslash^-1
12368 * parsers.linechar)
12369 - parsers.blankline^2
12370 - parsers.dollar
12371
12372 local inline_math_opening_dollars = parsers.dollar
12373 * #(parsers.nonspacechar)
12374
12375 local inline_math_closing_dollars
12376 = allowed_before_closing_no_space
12377 * parsers.dollar
12378 * -#(parsers.digit)
12379
12380 local inline_math_dollars = between(Cs(dollar_math_content),
12381 inline_math_opening_dollars,
12382 inline_math_closing_dollars)
12383
12384 local display_math_opening_dollars = parsers.dollar
12385 * parsers.dollar
12386
12387 local display_math_closing_dollars = parsers.dollar
12388 * parsers.dollar
12389
12390 local display_math_dollars = between(Cs(dollar_math_content),
12391 display_math_opening_dollars,
12392 display_math_closing_dollars)

```

The following patterns implement the Pandoc single and double backslash math syntax extensions.

```

12393 local backslash_math_content
12394 = (parsers.newline * (parsers.check_optional_indent / ""))
12395 + parsers.linechar)
12396 - parsers.blankline^2

```

The following patterns implement the Pandoc double backslash math syntax extension.

```

12397 local inline_math_opening_double = parsers.backslash
12398 * parsers.backslash
12399 * parsers.lparent
12400
12401 local inline_math_closing_double = allowed_before_closing
12402 * parsers.spacechar^0
12403 * parsers.backslash
12404 * parsers.backslash
12405 * parsers.rparent
12406

```

```

12407 local inline_math_double = between(Cs(backslash_math_content),
12408 inline_math_opening_double,
12409 inline_math_closing_double)
12410 / strip_preceding_whitespace
12411
12412 local display_math_opening_double = parsers.backslash
12413 * parsers.backslash
12414 * parsers.lbracket
12415
12416 local display_math_closing_double = allowed_before_closing
12417 * parsers.spacechar^0
12418 * parsers.backslash
12419 * parsers.backslash
12420 * parsers.rbracket
12421
12422 local display_math_double = between(Cs(backslash_math_content),
12423 display_math_opening_double,
12424 display_math_closing_double)
12425 / strip_preceding_whitespace

```

The following patterns implement the Pandoc single backslash math syntax extension.

```

12426 local inline_math_opening_single = parsers.backslash
12427 * parsers.lparent
12428
12429 local inline_math_closing_single = allowed_before_closing
12430 * parsers.spacechar^0
12431 * parsers.backslash
12432 * parsers.rparent
12433
12434 local inline_math_single = between(Cs(backslash_math_content),
12435 inline_math_opening_single,
12436 inline_math_closing_single)
12437 / strip_preceding_whitespace
12438
12439 local display_math_opening_single = parsers.backslash
12440 * parsers.lbracket
12441
12442 local display_math_closing_single = allowed_before_closing
12443 * parsers.spacechar^0
12444 * parsers.backslash
12445 * parsers.rbracket
12446
12447 local display_math_single = between(Cs(backslash_math_content),
12448 display_math_opening_single,
12449 display_math_closing_single)
12450 / strip_preceding_whitespace
12451
12452 local display_math = parsers.fail

```

```

12453
12454 local inline_math = parsers.fail
12455
12456 if tex_math_dollars then
12457 display_math = display_math + display_math_dollars
12458 inline_math = inline_math + inline_math_dollars
12459 end
12460
12461 if tex_math_double_backslash then
12462 display_math = display_math + display_math_double
12463 inline_math = inline_math + inline_math_double
12464 end
12465
12466 if tex_math_single_backslash then
12467 display_math = display_math + display_math_single
12468 inline_math = inline_math + inline_math_single
12469 end
12470
12471 local TexMath = display_math / writer.display_math
12472 + inline_math / writer.inline_math
12473
12474 self.insert_pattern("Inline after LinkAndEmph",
12475 TexMath, "TexMath")
12476
12477 if tex_math_dollars then
12478 self.add_special_character("$")
12479 end
12480
12481 if tex_math_single_backslash or tex_math_double_backslash then
12482 self.add_special_character("\\")
12483 self.add_special_character("[")
12484 self.add_special_character("]")
12485 self.add_special_character("(")
12486 self.add_special_character("(")
12487 end
12488 end
12489 }
12490 end

```

### 3.1.7.20 YAML Metadata

The `extensions.jekyll_data` function implements the Pandoc YAML metadata block syntax extension. When the `expect_jekyll_data` parameter is `true`, then a markdown document may begin directly with YAML metadata and may contain nothing but YAML metadata. When both `expect_jekyll_data` and



`ensure_jekyll_data` parameters are `true`, then a a markdown document must begin directly with YAML metadata and must contain nothing but YAML metadata.

```
12491 M.extensions.jekyll_data = function(expect_jekyll_data,
12492 ensure_jekyll_data)
12493 return {
12494 name = "built-in jekyll_data syntax extension",
12495 extend_writer = function(self)
```

Define `writer->jekyllData` as a function that will transform an input YAML table `d` to the output format. The table is the value for the key `p` in the parent table; if `p` is nil, then the table has no parent. All scalar keys and values encountered in the table will be cast to a string following YAML serialization rules. String values will also be transformed using the function `t` for the typographic output format used by the `\markdownRendererJekyllDataTypographicString` macro.

```
12496 function self.jekyllData(d, t, p)
12497 if not self.is_writing then return "" end
12498
12499 local buf = {}
12500
12501 local keys = {}
12502 for k, _ in pairs(d) do
12503 table.insert(keys, k)
12504 end
```

For reproducibility, sort the keys. For mixed string-and-numeric keys, sort numeric keys before string keys.

```
12505 table.sort(keys, function(first, second)
12506 if type(first) ~= type(second) then
12507 return type(first) < type(second)
12508 else
12509 return first < second
12510 end
12511 end)
12512
12513 if not p then
12514 table.insert(buf, "\\markdownRendererJekyllDataBegin")
12515 end
12516
12517 local is_sequence = false
12518 if #d > 0 and #d == #keys then
12519 for i=1, #d do
12520 if d[i] == nil then
12521 goto not_a_sequence
12522 end
12523 end
12524 is_sequence = true
12525 end
```

```

12526 ::not_a_sequence::
12527
12528 if is_sequence then
12529 table.insert(buf,
12530 "\\markdownRendererJekyllDataSequenceBegin{")
12531 table.insert(buf, self.identifier(p or "null"))
12532 table.insert(buf, "{")
12533 table.insert(buf, #keys)
12534 table.insert(buf, "}")
12535 else
12536 table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
12537 table.insert(buf, self.identifier(p or "null"))
12538 table.insert(buf, "{")
12539 table.insert(buf, #keys)
12540 table.insert(buf, "}")
12541 end
12542
12543 for _, k in ipairs(keys) do
12544 local v = d[k]
12545 local typ = type(v)
12546 k = tostring(k or "null")
12547 if typ == "table" and next(v) ~= nil then
12548 table.insert(
12549 buf,
12550 self.jekyllData(v, t, k)
12551)
12552 else
12553 k = self.identifier(k)
12554 v = tostring(v)
12555 if typ == "boolean" then
12556 table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
12557 table.insert(buf, k)
12558 table.insert(buf, "{")
12559 table.insert(buf, v)
12560 table.insert(buf, "}")
12561 elseif typ == "number" then
12562 table.insert(buf, "\\markdownRendererJekyllDataNumber{")
12563 table.insert(buf, k)
12564 table.insert(buf, "{")
12565 table.insert(buf, v)
12566 table.insert(buf, "}")
12567 elseif typ == "string" then
12568 table.insert(buf,
12569 "\\markdownRendererJekyllDataProgrammaticString{")
12570 table.insert(buf, k)
12571 table.insert(buf, "{")
12572 table.insert(buf, self.identifier(v))

```

```

12573 table.insert(buf, "}")
12574 table.insert(buf,
12575 "\\markdownRendererJekyllDataTypographicString{")
12576 table.insert(buf, k)
12577 table.insert(buf, "{")
12578 table.insert(buf, t(v))
12579 table.insert(buf, "}")
12580 elseif typ == "table" then
12581 table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
12582 table.insert(buf, k)
12583 table.insert(buf, "}")
12584 else
12585 local error = self.error(format(
12586 "Unexpected type %s for value of "
12587 .. "YAML key %s.", typ, k))
12588 table.insert(buf, error)
12589 end
12590 end
12591 end
12592
12593 if is_sequence then
12594 table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
12595 else
12596 table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
12597 end
12598
12599 if not p then
12600 table.insert(buf, "\\markdownRendererJekyllDataEnd")
12601 end
12602
12603 return buf
12604 end
12605 end, extend_reader = function(self)
12606 local parsers = self.parsers
12607 local writer = self.writer
12608
12609 local JekyllData
12610 = Cmt(C((parsers.line - P("---") - P("..."))^0)
12611 , function(s, i, text) -- luacheck: ignore s i
12612 local data
12613 local ran_ok, _ = pcall(function()
12614 local tinyyaml = require("tinyyaml")
12615 data = tinyyaml.parse(text, {timestamps=false})
12616 end)
12617 if ran_ok and data ~= nil then
12618 return true, writer.jekyllData(data, function(s)
12619 return self.parser_functions.parse_blocks_nested(s)

```

```

12620 end, nil)
12621 else
12622 return false
12623 end
12624 end
12625)
12626
12627 local UnexpectedJekyllData
12628 = P("----")
12629 * parsers.blankline / 0
12630 -- if followed by blank, it's thematic break
12631 * #(-parsers.blankline)
12632 * JekyllData
12633 * (P("----") + P("..."))
12634
12635 local ExpectedJekyllData
12636 = (P("----")
12637 * parsers.blankline / 0
12638 -- if followed by blank, it's thematic break
12639 * #(-parsers.blankline)
12640)^-1
12641 * JekyllData
12642 * (P("----") + P("..."))^-1
12643
12644 if ensure_jekyll_data then
12645 ExpectedJekyllData = ExpectedJekyllData
12646 * parsers.eof
12647 else
12648 ExpectedJekyllData = (ExpectedJekyllData
12649 * (V("Blank")^0 / writer.interblocksep)
12650)^-1
12651 end
12652
12653 self.insert_pattern("Block before Blockquote",
12654 UnexpectedJekyllData, "UnexpectedJekyllData")
12655 if expect_jekyll_data then
12656 self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
12657 end
12658 end
12659 }
12660 end

```

### 3.1.8 Conversion from Markdown to Plain T<sub>E</sub>X

The `new` function of file `markdown.lua` loads file `markdown-parser.lua` and calls its function `new` unless option `eagerCache` or `finalizeCache` has been enabled and

a cached conversion output exists, in which case it is returned without loading file `markdown-parser.lua`.

```
12661 function M.new(options)
```

Make the `options` table inherit from the `defaultOptions` table.

```
12662 options = options or {}
12663 setmetatable(options, { __index = function (_, key)
12664 return defaultOptions[key] end })
```

Return a conversion function that tries to produce a cached conversion output exists. If no cached conversion output exists, we load the file `markdown-parser.lua` and use it to convert the input.

```
12665 local parser_convert = nil
12666 return function(input, include_flat_output)
12667 local function convert(input)
12668 if parser_convert == nil then
```

Lazy-load `markdown-parser.lua` and check that it originates from the same version of the Markdown package.

```
12669 local parser = require("markdown-parser")
12670 if metadata.version ~= parser.metadata.version then
12671 warn("markdown.lua " .. metadata.version .. " used with " ..
12672 "markdown-parser.lua " .. parser.metadata.version .. ".")
12673 end
12674 parser_convert = parser.new(options)
12675 end
12676 return parser_convert(input)
12677 end
```

If we cache markdown documents, produce the cache file and transform its filename to plain TeX output.

When determining the name of the cache file, create salt for the hashing function out of the package version and the passed options recognized by the Lua interface (see Section 2.1.3).

```
12678 local raw_output, flat_output
12679 if options.eagerCache or options.finalizeCache then
12680 local salt = util.salt(options)
12681 local name, result = util.cache(options.cacheDir, input, salt,
12682 convert, ".md.tex")
12683 raw_output = [[\input{}}] .. name .. [[}\relax]]
12684 flat_output = function()
12685 if result == nil then
12686 local input_file = assert(io.open(name, "r"),
12687 [[Could not open file]] .. name .. [[for reading]])
12688 result = assert(input_file:read("*a"))
12689 assert(input_file:close())
12690 end
```

```

12691 return result
12692 end

```

Otherwise, return the result of the conversion directly.

```

12693 else
12694 raw_output = convert(input)
12695 flat_output = function()
12696 return raw_output
12697 end
12698 end

```

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

```

12699 if options.finalizeCache then
12700 local file, mode
12701 if options.frozenCacheCounter > 0 then
12702 mode = "a"
12703 else
12704 mode = "w"
12705 end
12706 file = assert(io.open(options.frozenCacheFileName, mode),
12707 [[Could not open file]] .. options.frozenCacheFileName
12708 .. [[for writing]])
12709 assert(file:write(
12710 [[\expandafter\global\expandafter\def\csname]]
12711 .. [[markdownFrozenCache]] .. options.frozenCacheCounter
12712 .. [[\endcsname{}]] .. raw_output .. [[]]] .. "\n"))
12713 assert(file:close())
12714 end

```

Besides the canonical output of the conversion, which may contain cached files behind `\input`, also return a function that always produces a flat output regardless of caching as the second return value.

```

12715 if include_flat_output then
12716 return raw_output, flat_output
12717 else
12718 return raw_output
12719 end
12720 end
12721 end

```

The `new` function from file `markdown-parser.lua` returns a conversion function that takes a markdown string and turns it into a plain T<sub>E</sub>X output. See Section 2.1.1.

```

12722 function M.new(options)

```

Make the `options` table inherit from the `defaultOptions` table.

```

12723 options = options or {}

```

```

12724 setmetatable(options, { __index = function (_, key)
12725 return defaultOptions[key] end })

```

If the singleton cache contains a conversion function for the same `options`, reuse it.

```

12726 if options.singletonCache and singletonCache.convert then
12727 for k, v in pairs(defaultOptions) do
12728 if type(v) == "table" then
12729 for i = 1, math.max(#singletonCache.options[k], #options[k]) do
12730 if singletonCache.options[k][i] ~= options[k][i] then
12731 goto miss
12732 end
12733 end

```

The `cacheDir` option is disregarded.

```

12734 elseif k ~= "cacheDir"
12735 and singletonCache.options[k] ~= options[k] then
12736 goto miss
12737 end
12738 end
12739 return singletonCache.convert
12740 end
12741 ::miss::

```

Apply built-in syntax extensions based on `options`.

```

12742 local extensions = {}
12743
12744 if options.bracketedSpans then
12745 local bracketed_spans_extension = M.extensions.bracketed_spans()
12746 table.insert(extensions, bracketed_spans_extension)
12747 end
12748
12749 if options.contentBlocks then
12750 local content_blocks_extension = M.extensions.content_blocks(
12751 options.contentBlocksLanguageMap)
12752 table.insert(extensions, content_blocks_extension)
12753 end
12754
12755 if options.definitionLists then
12756 local definition_lists_extension = M.extensions.definition_lists(
12757 options.tightLists)
12758 table.insert(extensions, definition_lists_extension)
12759 end
12760
12761 if options.fencedCode then
12762 local fenced_code_extension = M.extensions.fenced_code(
12763 options.blankBeforeCodeFence,
12764 options.fencedCodeAttributes,
12765 options.rawAttribute)

```

```

12766 table.insert(extensions, fenced_code_extension)
12767 end
12768
12769 if options.fencedDivs then
12770 local fenced_div_extension = M.extensions.fenced_divs(
12771 options.blankBeforeDivFence)
12772 table.insert(extensions, fenced_div_extension)
12773 end
12774
12775 if options.headerAttributes then
12776 local header_attributes_extension = M.extensions.header_attributes()
12777 table.insert(extensions, header_attributes_extension)
12778 end
12779
12780 if options.inlineCodeAttributes then
12781 local inline_code_attributes_extension =
12782 M.extensions.inline_code_attributes()
12783 table.insert(extensions, inline_code_attributes_extension)
12784 end
12785
12786 if options.jekyllData then
12787 local jekyll_data_extension = M.extensions.jekyll_data(
12788 options.expectJekyllData, options.ensureJekyllData)
12789 table.insert(extensions, jekyll_data_extension)
12790 end
12791
12792 if options.linkAttributes then
12793 local link_attributes_extension =
12794 M.extensions.link_attributes()
12795 table.insert(extensions, link_attributes_extension)
12796 end
12797
12798 if options.lineBlocks then
12799 local line_block_extension = M.extensions.line_blocks()
12800 table.insert(extensions, line_block_extension)
12801 end
12802
12803 if options.mark then
12804 local mark_extension = M.extensions.mark()
12805 table.insert(extensions, mark_extension)
12806 end
12807
12808 if options.pipeTables then
12809 local pipe_tables_extension = M.extensions.pipe_tables(
12810 options.tableCaptions, options.tableAttributes)
12811 table.insert(extensions, pipe_tables_extension)
12812 end

```



```

12813
12814 if options.rawAttribute then
12815 local raw_inline_extension = M.extensions.raw_inline()
12816 table.insert(extensions, raw_inline_extension)
12817 end
12818
12819 if options.strikeThrough then
12820 local strike_through_extension = M.extensions.strike_through()
12821 table.insert(extensions, strike_through_extension)
12822 end
12823
12824 if options.subscripts then
12825 local subscript_extension = M.extensions.subscripts()
12826 table.insert(extensions, subscript_extension)
12827 end
12828
12829 if options.superscripts then
12830 local superscript_extension = M.extensions.superscripts()
12831 table.insert(extensions, superscript_extension)
12832 end
12833
12834 if options.texMathDollars or
12835 options.texMathSingleBackslash or
12836 options.texMathDoubleBackslash then
12837 local tex_math_extension = M.extensions.tex_math(
12838 options.texMathDollars,
12839 options.texMathSingleBackslash,
12840 options.texMathDoubleBackslash)
12841 table.insert(extensions, tex_math_extension)
12842 end
12843
12844 if options.notes or options.inlineNotes then
12845 local notes_extension = M.extensions.notes(
12846 options.notes, options.inlineNotes)
12847 table.insert(extensions, notes_extension)
12848 end
12849
12850 if options.citations then
12851 local citations_extension
12852 = M.extensions.citations(options.citationNbsps)
12853 table.insert(extensions, citations_extension)
12854 end
12855
12856 if options.fancyLists then
12857 local fancy_lists_extension = M.extensions.fancy_lists()
12858 table.insert(extensions, fancy_lists_extension)
12859 end

```

Apply user-defined syntax extensions based on `options.extensions`.

```
12860 for _, user_extension_filename in ipairs(options.extensions) do
12861 local user_extension = (function(filename)
```

First, load and compile the contents of the user-defined syntax extension.

```
12862 local pathname = assert(kpse.find_file(filename),
12863 [[Could not locate user-defined syntax extension "]]
12864 .. filename)
12865 local input_file = assert(io.open(pathname, "r"),
12866 [[Could not open user-defined syntax extension "]]
12867 .. pathname .. [{" for reading}]]
12868 local input = assert(input_file:read("*a"))
12869 assert(input_file:close())
12870 local user_extension, err = load([[
12871 local sandbox = {}
12872 setmetatable(sandbox, {__index = _G})
12873 _ENV = sandbox
12874]] .. input)()
12875 assert(user_extension,
12876 [[Failed to compile user-defined syntax extension "]]
12877 .. pathname .. [{" ": }] .. (err or [{" }]))
```

Then, validate the user-defined syntax extension.

```
12878 assert(user_extension.api_version ~= nil,
12879 [[User-defined syntax extension "]] .. pathname
12880 .. [{" does not specify mandatory field "api_version"}]])
12881 assert(type(user_extension.api_version) == "number",
12882 [[User-defined syntax extension "]] .. pathname
12883 .. [{" specifies field "api_version" of type "]]
12884 .. type(user_extension.api_version)
12885 .. [{" but "number" was expected}]]
12886 assert(user_extension.api_version > 0
12887 and user_extension.api_version
12888 <= metadata.user_extension_api_version,
12889 [[User-defined syntax extension "]] .. pathname
12890 .. [{" uses syntax extension API version "]]
12891 .. user_extension.api_version .. [{" but markdown.lua]]
12892 .. metadata.version .. [{" uses API version "]]
12893 .. metadata.user_extension_api_version
12894 .. [{" , which is incompatible}]]
12895
12896 assert(user_extension.grammar_version ~= nil,
12897 [[User-defined syntax extension "]] .. pathname
12898 .. [{" does not specify mandatory field "grammar_version"}]])
12899 assert(type(user_extension.grammar_version) == "number",
12900 [[User-defined syntax extension "]] .. pathname
12901 .. [{" specifies field "grammar_version" of type "]]
12902 .. type(user_extension.grammar_version)
```

```

12903 .. [[[" but "number" was expected]]]
12904 assert(user_extension.grammar_version == metadata.grammar_version,
12905 [[User-defined syntax extension "]] .. pathname
12906 .. [[[" uses grammar version "]]
12907 .. user_extension.grammar_version
12908 .. [[[" but markdown.lua "]] .. metadata.version
12909 .. [[[" uses grammar version "]] .. metadata.grammar_version
12910 .. [[[, which is incompatible]]])
12911
12912 assert(user_extension.finalize_grammar ~= nil,
12913 [[User-defined syntax extension "]] .. pathname
12914 .. [[[" does not specify mandatory "finalize_grammar" field]])
12915 assert(type(user_extension.finalize_grammar) == "function",
12916 [[User-defined syntax extension "]] .. pathname
12917 .. [[[" specifies field "finalize_grammar" of type "]]
12918 .. type(user_extension.finalize_grammar)
12919 .. [[[" but "function" was expected]]])

```

Finally, cast the user-defined syntax extension to the internal format of user extensions used by the Markdown package (see Section 3.1.7.)

```

12920 local extension = {
12921 name = [[user-defined "]] .. pathname .. [[[" syntax extension]],
12922 extend_reader = user_extension.finalize_grammar,
12923 extend_writer = function() end,
12924 }
12925 return extension
12926 end)(user_extension_filename)
12927 table.insert(extensions, user_extension)
12928 end

```

Produce a conversion function from markdown to plain TeX.

```

12929 local writer = M.writer.new(options)
12930 local reader = M.reader.new(writer, options)
12931 local convert = reader.finalize_grammar(extensions)

```

Force garbage collection to reclaim memory for temporary objects created in `writer.new`, `reader.new`, and `reader->finalize_grammar`.

```

12932 collectgarbage("collect")

```

Update the singleton cache.

```

12933 if options.singletonCache then
12934 local singletonCacheOptions = {}
12935 for k, v in pairs(options) do
12936 singletonCacheOptions[k] = v
12937 end
12938 setmetatable(singletonCacheOptions,
12939 { __index = function (_, key)
12940 return defaultOptions[key] end })

```

```

12941 singletonCache.options = singletonCacheOptions
12942 singletonCache.convert = convert
12943 end

```

Return the conversion function from markdown to plain T<sub>E</sub>X.

```

12944 return convert
12945 end

12946 return M

```

### 3.1.9 Command-Line Implementation

The command-line implementation provides the actual conversion routine for the command-line interface described in Section 2.1.7.

```

12947
12948 local input
12949 if input_filename then
12950 local input_file = assert(io.open(input_filename, "r"),
12951 [[Could not open file]] .. input_filename .. [[for reading]])
12952 input = assert(input_file:read("*a"))
12953 assert(input_file:close())
12954 else
12955 input = assert(io.read("*a"))
12956 end
12957

```

First, ensure that the `options.cacheDir` directory exists.

```

12958 local lfs = require("lfs")
12959 if options.cacheDir and not lfs.isdir(options.cacheDir) then
12960 assert(lfs.mkdir(options["cacheDir"]))
12961 end

```

If Kpathsea has not been loaded before or if LuaT<sub>E</sub>X has not yet been initialized, configure Kpathsea on top of loading it.

```

12962 local kpse
12963 (function()
12964 local should_initialize = package.loaded.kpse == nil
12965 or tex.initialize ~= nil
12966 kpse = require("kpse")
12967 if should_initialize then
12968 kpse.set_program_name("luatex")
12969 end
12970 end)()
12971 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.

```

12972 if metadata.version ~= md.metadata.version then

```

```

12973 warn("markdown-cli.lua " .. metadata.version .. " used with " ..
12974 "markdown.lua " .. md.metadata.version .. ".")
12975 end
12976
12977 local convert = md.new(options)
12978 local raw_output, flat_output = convert(input, true)
12979 local output
12980 if flat_output == nil then
12981 if options.eagerCache then
12982 warn("markdown.lua has not produced flat output, so I am using " ..
12983 "backwards-compatible raw output instead. This may cause " ..
12984 "the conversion result to be hidden behind "\\input'.')
12985 end
12986 output = raw_output
12987 else
12988 output = flat_output()
12989 end
12990
12991 if output_filename then
12992 local output_file = assert(io.open(output_filename, "w"),
12993 [[Could not open file]] .. output_filename .. [[for writing]])
12994 assert(output_file:write(output))
12995 assert(output_file:close())
12996 else
12997 assert(io.write(output))
12998 end
12999
13000 Remove the options.cacheDir directory if it is empty.
13001 if options.cacheDir then
13002 lfs.rmdir(options.cacheDir)
13003 end

```

## 3.2 Plain T<sub>E</sub>X Implementation

The plain T<sub>E</sub>X implementation provides macros for the interfacing between T<sub>E</sub>X and Lua and for the buffering of input text. These macros are then used to implement the macros for the conversion from markdown to plain T<sub>E</sub>X exposed by the plain T<sub>E</sub>X interface (see Section 2.2).

### 3.2.1 Logging Facilities

```

13002 \ExplSyntaxOn
13003 \cs_if_free:NT
13004 \markdownInfo
13005 {
13006 \cs_new:Npn
13007 \markdownInfo #1

```

```

13008 {
13009 \msg_info:nne
13010 { markdown }
13011 { generic-message }
13012 { #1 }
13013 }
13014 }
13015 \cs_if_free:NT
13016 \markdownWarning
13017 {
13018 \cs_new:Npn
13019 \markdownWarning #1
13020 {
13021 \msg_warning:nne
13022 { markdown }
13023 { generic-message }
13024 { #1 }
13025 }
13026 }
13027 \cs_if_free:NT
13028 \markdownError
13029 {
13030 \cs_new:Npn
13031 \markdownError #1 #2
13032 {
13033 \msg_error:nnee
13034 { markdown }
13035 { generic-message-with-help-text }
13036 { #1 }
13037 { #2 }
13038 }
13039 }
13040 \msg_new:nnn
13041 { markdown }
13042 { generic-message }
13043 { #1 }
13044 \msg_new:nnnn
13045 { markdown }
13046 { generic-message-with-help-text }
13047 { #1 }
13048 { #2 }
13049 \cs_generate_variant:Nn
13050 \msg_info:nnn
13051 { nne }
13052 \cs_generate_variant:Nn
13053 \msg_warning:nnn
13054 { nne }

```

```

13055 \cs_generate_variant:Nn
13056 \msg_error:nnnn
13057 { nnee }
13058 \ExplSyntaxOff

```

### 3.2.2 Themes

This section implements the theme-loading mechanism and the built-in themes provided with the Markdown package. Furthermore, this section also implements the built-in plain T<sub>E</sub>X themes provided with the Markdown package.

```

13059 \ExplSyntaxOn
13060 \prop_new:N \g_@@_plain_tex_loaded_themes_linenos_prop
13061 \prop_new:N \g_@@_plain_tex_loaded_themes_versions_prop
13062 \cs_new:Nn
13063 \@@_plain_tex_load_theme:nnn
13064 {
13065 \prop_get:NnNTF
13066 \g_@@_plain_tex_loaded_themes_linenos_prop
13067 { #1 }
13068 \l_tmpa_tl
13069 {
13070 \prop_get:NnN
13071 \g_@@_plain_tex_loaded_themes_versions_prop
13072 { #1 }
13073 \l_tmpb_tl
13074 \str_if_eq:nVTF
13075 { #2 }
13076 \l_tmpb_tl
13077 {
13078 \msg_warning:nnnVn
13079 { markdown }
13080 { repeatedly-loaded-plain-tex-theme }
13081 { #1 }
13082 \l_tmpa_tl
13083 { #2 }
13084 }
13085 {
13086 \msg_error:nnnnVV
13087 { markdown }
13088 { different-versions-of-plain-tex-theme }
13089 { #1 }
13090 { #2 }
13091 \l_tmpb_tl
13092 \l_tmpa_tl
13093 }
13094 }
13095 {

```

```

13096 \prop_gput:Nnx
13097 \g_@@_plain_tex_loaded_themes_linenos_prop
13098 { #1 }
13099 { \tex_the:D \tex_inputlineno:D } % noqa: W200
13100 \prop_gput:Nnn
13101 \g_@@_plain_tex_loaded_themes_versions_prop
13102 { #1 }
13103 { #2 }

```

Load built-in plain TeX themes from the prop `\g_@@_plain_tex_built_in_themes_prop` and from the filesystem otherwise.

```

13104 \prop_if_in:NnTF
13105 \g_@@_plain_tex_built_in_themes_prop
13106 { #1 }
13107 {
13108 \msg_info:nnnn
13109 { markdown }
13110 { loading-built-in-plain-tex-theme }
13111 { #1 }
13112 { #2 }
13113 \prop_item:Nn
13114 \g_@@_plain_tex_built_in_themes_prop
13115 { #1 }
13116 }
13117 {
13118 \msg_info:nnnn
13119 { markdown }
13120 { loading-plain-tex-theme }
13121 { #1 }
13122 { #2 }
13123 \file_input:n
13124 { markdown theme #3 }
13125 }
13126 }
13127 }
13128 \msg_new:nnn
13129 { markdown }
13130 { loading-plain-tex-theme }
13131 { Loading-version-#2-of-plain-TeX-Markdown-theme-#1 }
13132 \msg_new:nnn
13133 { markdown }
13134 { loading-built-in-plain-tex-theme }
13135 { Loading-version-#2-of-built-in-plain-TeX-Markdown-theme-#1 }
13136 \msg_new:nnn
13137 { markdown }
13138 { repeatedly-loaded-plain-tex-theme }
13139 {

```



```

13140 Version~#3~of~plain~TeX~Markdown~theme~#1~was~previously~
13141 loaded~on~line~#2,~not~loading~it~again
13142 }
13143 \msg_new:nnn
13144 { markdown }
13145 { different-versions-of-plain-tex-theme }
13146 {
13147 Tried~to~load~version~#2~of~plain~TeX~Markdown~theme~#1~
13148 but~version~#3~has~already~been~loaded~on~line~#4
13149 }
13150 \cs_generate_variant:Nn
13151 \prop_gput:Nnn
13152 { Nnx }
13153 \cs_gset_eq:NN
13154 \@@_load_theme:nnn
13155 \@@_plain_tex_load_theme:nnn
13156 \cs_generate_variant:Nn
13157 \@@_load_theme:nnn
13158 { VeV }
13159 \cs_generate_variant:Nn
13160 \msg_error:nnnnnn
13161 { nnnnVV }
13162 \cs_generate_variant:Nn
13163 \msg_warning:nnnnn
13164 { nnnVn }

```

Developers can use the `\markdownLoadPlainTeXTheme` macro to load a corresponding plain T<sub>E</sub>X theme from within themes for higher-level T<sub>E</sub>X formats such as L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt.

```

13165 \cs_new:Npn
13166 \markdownLoadPlainTeXTheme
13167 {

```

First, we extract the name of the current theme from the `\g_@@_current_theme_tl` macro.

```

13168 \tl_set:NV
13169 \l_tmpa_tl
13170 \g_@@_current_theme_tl
13171 \tl_reverse:N
13172 \l_tmpa_tl
13173 \tl_set:Ne
13174 \l_tmpb_tl
13175 {
13176 \tl_tail:V
13177 \l_tmpa_tl
13178 }
13179 \tl_reverse:N
13180 \l_tmpb_tl

```

Next, we munge the theme name.

```
13181 \str_set:NV
13182 \l_tmpa_str
13183 \l_tmpb_tl
13184 \str_replace_all:Nnn
13185 \l_tmpa_str
13186 { / }
13187 { _ }
```

Finally, we load the plain TeX theme.

```
13188 \@@_plain_tex_load_theme:VeV
13189 \l_tmpb_tl
13190 { \markdownThemeVersion }
13191 \l_tmpa_str
13192 }
13193 \cs_generate_variant:Nn
13194 \tl_set:Nn
13195 { Ne }
13196 \cs_generate_variant:Nn
13197 \@@_plain_tex_load_theme:nnn
13198 { VeV }
```

The `witiko/dot` theme nags users that they should use the name `witiko/diagrams@v1` instead.

```
13199 \prop_gput:Nnn
13200 \g_@@_plain_tex_built_in_themes_prop
13201 { witiko / dot }
13202 {
13203 \str_if_eq:enF
13204 { \markdownThemeVersion }
13205 { silent }
13206 {
13207 \markdownWarning
13208 {
13209 The~theme~name~"witiko/dot"~has~been~soft-deprecated.
13210 \iow_newline:
13211 Consider~changing~the~name~to~"witiko/diagrams@v1".
13212 }
13213 }
13214 }
```

We enable the `fencedCode` Lua option.

```
13214 \markdownSetup { fencedCode }
```

We store the previous definition of the fenced code token renderer prototype:

```
13215 \cs_set_eq:NN
13216 \@@_dot_previous_definition:nnn
13217 \markdownRendererInputFencedCodePrototype
```

If the infostring starts with `dot ...`, we redefine the fenced code block token renderer prototype, so that it typesets the code block via Graphviz tools if and only if the `frozenCache` plain T<sub>E</sub>X option is disabled and the code block has not been previously typeset:

```

13218 \regex_const:Nn
13219 \c_@@_dot_infostring_regex
13220 { ^dot(\s+(.+)?)? }
13221 \seq_new:N
13222 \l_@@_dot_matches_seq
13223 \markdownSetup {
13224 rendererPrototypes = {
13225 inputFencedCode = {
13226 \regex_extract_once:NnNTF
13227 \c_@@_dot_infostring_regex
13228 { #2 }
13229 \l_@@_dot_matches_seq
13230 {
13231 \@@_if_option:nF
13232 { frozenCache }
13233 {
13234 \sys_shell_now:n
13235 {
13236 if!~test~-e~#1.pdf.source~
13237 ||!~diff~#1~#1.pdf.source;
13238 then~
13239 dot~-Tpdf~-o~#1.pdf~#1;
13240 cp~#1~#1.pdf.source;
13241 fi
13242 }
13243 }

```

We include the typeset image using the image token renderer:

```

13244 \exp_args:NNne
13245 \exp_last_unbraced:No
13246 \markdownRendererImage
13247 {
13248 { Graphviz~image }
13249 { #1.pdf }
13250 { #1.pdf }
13251 }
13252 {
13253 \seq_item:Nn
13254 \l_@@_dot_matches_seq
13255 { 3 }
13256 }
13257 }

```

If the infostring does not start with `dot ...`, we use the previous definition of the fenced code token renderer prototype:

```

13258 {
13259 \@@_dot_previous_definition:nnn
13260 { #1 }
13261 { #2 }
13262 { #3 }
13263 }
13264 },
13265 },
13266 }
13267 }
```

The theme `witiko/diagrams` loads either the theme `witiko/dot` for version `v1` or the theme `witiko/diagrams/v2` for version `v2`.

```

13268 \prop_gput:Nnn
13269 \g_@@_plain_tex_built_in_themes_prop
13270 { witiko / diagrams }
13271 {
13272 \str_case:enF
13273 { \markdownThemeVersion }
13274 {
13275 { latest }
13276 {
13277 \markdownWarning
13278 {
13279 Write~"witiko/diagrams@v2"~to~pin~version~"v2"~of~the~
13280 theme~"witiko/diagrams".~This~will~keep~your~documents~
13281 from~suddenly~breaking~when~we~have~released~future~
13282 versions~of~the~theme~with~backwards~incompatible~
13283 syntax~and~behavior.
13284 }
13285 \markdownSetup
13286 {
13287 import = witiko/diagrams/v2,
13288 }
13289 }
13290 { v2 }
13291 {
13292 \markdownSetup
13293 {
13294 import = witiko/diagrams/v2,
13295 }
13296 }
13297 { v1 }
13298 {
13299 \markdownSetup
```

```

13300 {
13301 import = witiko/dot@silent,
13302 }
13303 }
13304 }
13305 {
13306 \msg_error:nnnn
13307 { markdown }
13308 { unknown-theme-version }
13309 { witiko/diagrams }
13310 { \markdownThemeVersion }
13311 { v1 }
13312 }
13313 }
13314 \cs_generate_variant:Nn
13315 \msg_error:nnnnn
13316 { nnnn }
13317 \msg_new:nnnn
13318 { markdown }
13319 { unknown-theme-version }
13320 { Unknown~version~"#2"~of~theme~"#1"~has~been~requested. }
13321 { Known~versions~are:~#3 }

```

Next, we implement the theme `witiko/diagrams/v2`.

```

13322 \prop_gput:Nnn
13323 \g_@@_plain_tex_built_in_themes_prop
13324 { witiko / diagrams / v2 }
13325 {

```

We enable the `fencedCode` and `fencedCodeAttributes` Lua option.

```

13326 \@@_setup:n
13327 {
13328 fencedCode = true,
13329 fencedCodeAttributes = true,
13330 }

```

Store the previous fenced code token renderer prototype.

```

13331 \cs_set_eq:NN
13332 \@@_diagrams_previous_fenced_code:nnn
13333 \markdownRendererInputFencedCodePrototype

```

Store the caption and the desired format of the diagram.

```

13334 \tl_new:N
13335 \l_@@_diagrams_caption_tl
13336 \tl_new:N
13337 \l_@@_diagrams_format_tl
13338 \tl_set:Nn
13339 \l_@@_diagrams_format_tl
13340 { pdf }

```

```

13341 \@@_setup:n
13342 {
13343 rendererPrototypes = {
Route attributes on fenced code blocks to the image attribute renderer prototypes.
13344 fencedCodeAttributeContextBegin = {
13345 \group_begin:
13346 \markdownRendererImageAttributeContextBegin
13347 \cs_set_eq:NN
13348 \@@_diagrams_previous_key_value:nn
13349 \markdownRendererAttributeKeyValuePrototype
13350 \@@_setup:n
13351 {
13352 rendererPrototypes = {
13353 attributeKeyValue = {
13354 \str_case:nnF
13355 { ##1 }
13356 {
13357 { caption }
13358 {
13359 \tl_set:Nn
13360 \l_@@_diagrams_caption_tl
13361 { ##2 }
13362 }
13363 { format }
13364 {
13365 \tl_set:Nn
13366 \l_@@_diagrams_format_tl
13367 { ##2 }
13368 }
13369 }
13370 {
13371 \@@_diagrams_previous_key_value:nn
13372 { ##1 }
13373 { ##2 }
13374 }
13375 },
13376 },
13377 }
13378 },
13379 fencedCodeAttributeContextEnd = {
13380 \markdownRendererImageAttributeContextEnd
13381 \group_end:
13382 },
13383 },
13384 }
13385 \cs_new:Nn
13386 \@@_diagrams_render_diagram:nnnn

```

```

13387 {
13388 \@@_if_option:nF
13389 { frozenCache }
13390 {
13391 \sys_shell_now:n
13392 {
13393 if~!~test~-e~#2.source~
13394 ||~!~diff~#1~#2.source;
13395 then~
13396 (#3);
13397 cp~#1~#2.source;
13398 fi
13399 }
13400 \exp_args:NNnV
13401 \exp_last_unbraced:No
13402 \markdownRendererImage
13403 {
13404 { #4 }
13405 { #2 }
13406 { #2 }
13407 }
13408 \l_@@_diagrams_caption_tl
13409 }
13410 }

```

Use the prop `\g_markdown_diagrams_infostrings_prop` to determine how the code with a given infostring should be processed and routed to the token renderer prototype(s) for images.

```

13411 \prop_new:N
13412 \g_markdown_diagrams_infostrings_prop

```

If we know a processing function for a given infostring, use it.

```

13413 \@@_setup:n
13414 {
13415 rendererPrototypes = {
13416 inputFencedCode = {
13417 \prop_get:NnNTF
13418 \g_markdown_diagrams_infostrings_prop
13419 { #2 }
13420 \l_tmpa_tl
13421 {
13422 \cs_set:NV
13423 \@@_diagrams_infostrings_current:n
13424 \l_tmpa_tl
13425 \@@_diagrams_infostrings_current:n
13426 { #1 }
13427 }

```

Otherwise, use the previous fenced code token renderer prototype.

```

13428 {
13429 \@@_diagrams_previous_fenced_code:nmm
13430 { #1 }
13431 { #2 }
13432 { #3 }
13433 }
13434 },
13435 },
13436 }
13437 \cs_generate_variant:Nn
13438 \cs_set:Nn
13439 { NV }

```

Typeset fenced code with infostring `dot` using the command `dot` from the package `Graphviz`.

```

13440 \cs_set:Nn
13441 \@@_diagrams_infostrings_current:n
13442 {
13443 \@@_diagrams_render_diagram:nnnn
13444 { #1 }
13445 { #1.pdf }
13446 { dot~--Tpdf~--o~#1.pdf~#1 }
13447 { Graphviz~image }
13448 }
13449 \@@_tl_set_from_cs:NNn
13450 \l_tmpa_tl
13451 \@@_diagrams_infostrings_current:n
13452 { 1 }
13453 \prop_gput:NnV
13454 \g_markdown_diagrams_infostrings_prop
13455 { dot }
13456 \l_tmpa_tl

```

Typeset fenced code with infostring `mermaid` using the command `mmdc` from the npm package `@mermaid-js/mermaid-cli`.

```

13457 \cs_set:Nn
13458 \@@_diagrams_infostrings_current:n
13459 {
13460 \@@_diagrams_render_diagram:nnnn
13461 { #1 }
13462 { #1.pdf }
13463 { mmdc~--pdfFit~--i~#1~--o~#1.pdf }
13464 { Mermaid~image }
13465 }
13466 \@@_tl_set_from_cs:NNn
13467 \l_tmpa_tl

```



```

13468 \@@_diagrams_infostrings_current:n
13469 { 1 }
13470 \prop_gput:NnV
13471 \g_markdown_diagrams_infostrings_prop
13472 { mermaid }
13473 \l_tmpa_tl

```

Typeset fenced code with infostring `plantuml` using the command `plantuml` from the package PlantUML.

```

13474 \regex_const:Nn
13475 \c_@@_diagrams_filename_suffix_regex
13476 { \.[^\.]*$ }
13477 \cs_set:Nn
13478 \@@_diagrams_infostrings_current:n
13479 {

```

Use the output format provided by the user.

```

13480 \tl_set:Nn
13481 \l_tmpa_tl
13482 { #1 }
13483 \regex_replace_once:NxN
13484 \c_@@_diagrams_filename_suffix_regex
13485 {
13486 .
13487 \tl_use:N
13488 \l_@@_diagrams_format_tl
13489 }
13490 \l_tmpa_tl
13491 \tl_set:Nn
13492 \l_tmpb_tl
13493 { plantuml~-t }
13494 \tl_put_right:NV
13495 \l_tmpb_tl
13496 \l__markdown_diagrams_format_tl
13497 \tl_put_right:Nn
13498 \l_tmpb_tl
13499 { ~#1 }

```

For the SVG format, use Inkscape to convert the resulting image to PDF.

```

13500 \str_if_eq:VnT
13501 \l_@@_diagrams_format_tl
13502 { svg }
13503 {
13504 \tl_put_right:Nn
13505 \l_tmpb_tl
13506 { ;~inkscape~ }
13507 \tl_put_right:NV
13508 \l_tmpb_tl

```

```

13509 \l_tmpa_tl
13510 \tl_put_right:Nn
13511 \l_tmpb_tl
13512 { ---export-area-drawing---export-dpi=300~-o~ }
13513 \tl_set:Nn
13514 \l_tmpa_tl
13515 { #1 }
13516 \regex_replace_once:NnN
13517 \c_@@_diagrams_filename_suffix_regex
13518 { .pdf }
13519 \l_tmpa_tl
13520 \tl_put_right:NV
13521 \l_tmpb_tl
13522 \l_tmpa_tl
13523 }
13524 \@@_diagrams_render_diagram:nVVn
13525 { #1 }
13526 \l_tmpa_tl
13527 \l_tmpb_tl
13528 { PlantUML~image }
13529 }
13530 \cs_generate_variant:Nn
13531 \@@_diagrams_render_diagram:nnnn
13532 { nVVn }
13533 \cs_generate_variant:Nn
13534 \regex_replace_once:NnN
13535 { NxN }
13536 \@@_tl_set_from_cs:NNn
13537 \l_tmpa_tl
13538 \@@_diagrams_infostrings_current:n
13539 { 1 }
13540 \prop_gput:NnV
13541 \g_markdown_diagrams_infostrings_prop
13542 { plantuml }
13543 \l_tmpa_tl
13544 }

```

We locally change the category code of percent signs, so that we can use them in the shell code:

```

13545 \group_begin:
13546 \char_set_catcode_other:N \%

```

The [witiko/graphicx/http](https://github.com/witiko/graphicx/http) theme stores the previous definition of the image token renderer prototype:

```

13547 \prop_gput:Nnn
13548 \g_@@_plain_tex_built_in_themes_prop
13549 { witiko / graphicx / http }
13550 {

```

```

13551 \cs_set_eq:NN
13552 \@@_graphicx_http_previous_definition:nnnn
13553 \markdownRendererImagePrototype

```

We define variables and functions to enumerate the images for caching and to store the pathname of the file containing the pathname of the downloaded image file.

```

13554 \int_new:N
13555 \g_@@_graphicx_http_image_number_int
13556 \int_gset:Nn
13557 \g_@@_graphicx_http_image_number_int
13558 { 0 }
13559 \cs_new:Nn
13560 \@@_graphicx_http_filename:
13561 {
13562 \markdownOptionCacheDir
13563 / witiko_graphicx_http .
13564 \int_use:N
13565 \g_@@_graphicx_http_image_number_int
13566 }

```

We define a function that will receive two arguments that correspond to the URL of the online image and to the pathname, where the online image should be downloaded. The function produces a shell command that tries to download the online image to the pathname.

```

13567 \cs_new:Nn
13568 \@@_graphicx_http_download:nn
13569 {
13570 wget~-0~#2~#1~
13571 ||~curl~--location~-o~#2~#1~
13572 ||~rm~-f~#2
13573 }

```

We redefine the image token renderer prototype, so that it tries to download an online image.

```

13574 \str_new:N
13575 \l_@@_graphicx_http_filename_str
13576 \ior_new:N
13577 \g_@@_graphicx_http_filename_ior
13578 \markdownSetup {
13579 rendererPrototypes = {
13580 image = {
13581 \@@_if_option:nF
13582 { frozenCache }
13583 {

```

The image will be downloaded only if the image URL has the http or https protocols and the `frozenCache` plain T<sub>E</sub>X option is disabled:

```

13584 \sys_shell_now:e

```

```

13585 {
13586 mkdir~-p~" \markdownOptionCacheDir ";
13587 if~printf~'%s'~"#3"~|~grep~-q~-E~'^https?:';
13588 then~

```

The image will be downloaded to the pathname `cacheDir/⟨the MD5 digest of the image URL⟩.⟨the suffix of the image URL⟩`:

```

13589 OUTPUT_PREFIX=" \markdownOptionCacheDir ";
13590 OUTPUT_BODY="$(printf~'%s'~'#3'
13591 |~md5sum~|~cut~-d'~'~-f1)";
13592 OUTPUT_SUFFIX="$(printf~'%s'~'#3'
13593 |~sed~'s/.*[.]//')";
13594 OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";

```

The image will be downloaded only if it has not already been downloaded:

```

13595 if~!~[~-e~"$OUTPUT"~];
13596 then~
13597 \@@_graphicx_http_download:nn
13598 { '#3' }
13599 { "$OUTPUT" } ;
13600 printf~'%s'~"$OUTPUT"~
13601 >~" \@@_graphicx_http_filename: ";
13602 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:

```

13603 else~
13604 printf~'%s'~'#3'~
13605 >~" \@@_graphicx_http_filename: ";
13606 fi
13607 }
13608 }

```

We load the pathname of the downloaded image and we typeset the image using the previous definition of the image renderer prototype:

```

13609 \ior_open:Ne
13610 \g_@@_graphicx_http_filename_ior
13611 { \@@_graphicx_http_filename: }
13612 \ior_str_get:NN
13613 \g_@@_graphicx_http_filename_ior
13614 \l_@@_graphicx_http_filename_str
13615 \ior_close:N
13616 \g_@@_graphicx_http_filename_ior
13617 \@@_graphicx_http_previous_definition:nnVn
13618 { #1 }
13619 { #2 }
13620 \l_@@_graphicx_http_filename_str
13621 { #4 }

```

```

13622 \int_gincr:N
13623 \g_@@_graphicx_http_image_number_int
13624 }
13625 }
13626 }
13627 \cs_generate_variant:Nn
13628 \ior_open:Nn
13629 { Ne }
13630 \cs_generate_variant:Nn
13631 \@@_graphicx_http_previous_definition:nnnn
13632 { nnVn }
13633 }
13634 \group_end:

```

The [witiko/tilde](#) theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```

13635 \prop_gput:Nnn
13636 \g_@@_plain_tex_built_in_themes_prop
13637 { witiko / tilde }
13638 {
13639 \markdownSetup {
13640 rendererPrototypes = {
13641 tilde = {~},
13642 },
13643 }
13644 }

```

The themes [witiko/example/foo](#) and [witiko/example/bar](#) are supposed to be used in code examples. They don't do anything.

```

13645 \clist_map_inline:nn
13646 { foo, bar }
13647 {
13648 \prop_gput:Nnn
13649 \g_@@_plain_tex_built_in_themes_prop
13650 { witiko / example / #1 }
13651 {
13652 \markdownWarning
13653 {
13654 The~theme~witiko/example/#1~is~supposed~to~be~used~in~code~
13655 examples.~Using~it~in~actual~code~has~no~effect,~except~
13656 this~warning~message,~and~is~usually~a~mistake.
13657 }
13658 }
13659 }
13660 \ExplSyntaxOff

```

The [witiko/markdown/defaults](#) plain T<sub>E</sub>X theme provides default definitions for token renderer prototypes. See Section 3.2.3 for the actual definitions.

### 3.2.3 Token Renderer Prototypes

The following definitions should be considered placeholder.

```
13661 \def\markdownRendererInterblockSeparatorPrototype{\par}%
13662 \def\markdownRendererParagraphSeparatorPrototype{%
13663 \markdownRendererInterblockSeparator}%
13664 \def\markdownRendererHardLineBreakPrototype{\hfil\break}%
13665 \def\markdownRendererSoftLineBreakPrototype{ }%
13666 \let\markdownRendererEllipsisPrototype\dots
13667 \def\markdownRendererNbspPrototype{~}%
13668 \def\markdownRendererLeftBracePrototype{\char`\{}%
13669 \def\markdownRendererRightBracePrototype{\char`\}%
13670 \def\markdownRendererDollarSignPrototype{\char`$}%
13671 \def\markdownRendererPercentSignPrototype{\char`\}%
13672 \def\markdownRendererAmpersandPrototype{&}%
13673 \def\markdownRendererUnderscorePrototype{\char`_}%
13674 \def\markdownRendererHashPrototype{\char`\#}%
13675 \def\markdownRendererCircumflexPrototype{\char`^}%
13676 \def\markdownRendererBackslashPrototype{\char`\}%
13677 \def\markdownRendererTildePrototype{\char`~}%
13678 \def\markdownRendererPipePrototype{|}%
13679 \def\markdownRendererCodeSpanPrototype#1{{\tt#1}}%
13680 \def\markdownRendererLinkPrototype#1#2#3#4#{#2}%
13681 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
13682 \markdownInput{#3}}%
13683 \def\markdownRendererContentBlockOnlineImagePrototype{%
13684 \markdownRendererImage}%
13685 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
13686 \markdownRendererInputFencedCode{#3}{#2}{#2}}%
13687 \def\markdownRendererImagePrototype#1#2#3#4#{#2}%
13688 \def\markdownRendererUlBeginPrototype{}%
13689 \def\markdownRendererUlBeginTightPrototype{}%
13690 \def\markdownRendererUlItemPrototype{}%
13691 \def\markdownRendererUlItemEndPrototype{}%
13692 \def\markdownRendererUlEndPrototype{}%
13693 \def\markdownRendererUlEndTightPrototype{}%
13694 \def\markdownRendererOlBeginPrototype{}%
13695 \def\markdownRendererOlBeginTightPrototype{}%
13696 \def\markdownRendererFancyOlBeginPrototype#1#2{%
13697 \markdownRendererOlBegin}%
13698 \def\markdownRendererFancyOlBeginTightPrototype#1#2{%
13699 \markdownRendererOlBeginTight}%
13700 \def\markdownRendererOlItemPrototype{}%
13701 \def\markdownRendererOlItemWithNumberPrototype#1{}%
13702 \def\markdownRendererOlItemEndPrototype{}%
13703 \def\markdownRendererFancyOlItemPrototype{\markdownRendererOlItem}%
13704 \def\markdownRendererFancyOlItemWithNumberPrototype%
```

```

13705 \markdownRendererOlItemWithNumber}%
13706 \def\markdownRendererFancyOlItemEndPrototype{}%
13707 \def\markdownRendererOlEndPrototype{}%
13708 \def\markdownRendererOlEndTightPrototype{}%
13709 \def\markdownRendererFancyOlEndPrototype{\markdownRendererOlEnd}%
13710 \def\markdownRendererFancyOlEndTightPrototype{%
13711 \markdownRendererOlEndTight}%
13712 \def\markdownRendererDlBeginPrototype{}%
13713 \def\markdownRendererDlBeginTightPrototype{}%
13714 \def\markdownRendererDlItemPrototype#1{#1}%
13715 \def\markdownRendererDlItemEndPrototype{}%
13716 \def\markdownRendererDlDefinitionBeginPrototype{}%
13717 \def\markdownRendererDlDefinitionEndPrototype{\par}%
13718 \def\markdownRendererDlEndPrototype{}%
13719 \def\markdownRendererDlEndTightPrototype{}%
13720 \def\markdownRendererEmphasisPrototype#1{\it#1}%
13721 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
13722 \def\markdownRendererBlockQuoteBeginPrototype{\begingroup\it}%
13723 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
13724 \def\markdownRendererLineBlockBeginPrototype{\begingroup\parindent=0pt}%
13725 \def\markdownRendererLineBlockEndPrototype{\endgroup}%
13726 \def\markdownRendererInputVerbatimPrototype#1{%
13727 \par{\tt\input#1\relax}}\par}%
13728 \def\markdownRendererInputFencedCodePrototype#1#2#3{%
13729 \markdownRendererInputVerbatim{#1}}%
13730 \def\markdownRendererHeadingOnePrototype#1{#1}%
13731 \def\markdownRendererHeadingTwoPrototype#1{#1}%
13732 \def\markdownRendererHeadingThreePrototype#1{#1}%
13733 \def\markdownRendererHeadingFourPrototype#1{#1}%
13734 \def\markdownRendererHeadingFivePrototype#1{#1}%
13735 \def\markdownRendererHeadingSixPrototype#1{#1}%
13736 \def\markdownRendererThematicBreakPrototype{}%
13737 \def\markdownRendererNotePrototype#1{#1}%
13738 \def\markdownRendererCitePrototype#1{}%
13739 \def\markdownRendererTextCitePrototype#1{}%
13740 \def\markdownRendererTickedBoxPrototype{[X]}%
13741 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
13742 \def\markdownRendererUntickedBoxPrototype{[]}%
13743 \def\markdownRendererStrikeThroughPrototype#1{#1}%
13744 \def\markdownRendererSuperscriptPrototype#1{#1}%
13745 \def\markdownRendererSubscriptPrototype#1{#1}%
13746 \def\markdownRendererDisplayMathPrototype#1{$$#1$$}%
13747 \def\markdownRendererInlineMathPrototype#1{$#1$}%
13748 \ExplSyntaxOn
13749 \cs_gset:Npn
13750 \markdownRendererHeaderAttributeContextBeginPrototype
13751 {

```

```

13752 \group_begin:
13753 \color_group_begin:
13754 }
13755 \cs_gset:Npn
13756 \markdownRendererHeaderAttributeContextEndPrototype
13757 {
13758 \color_group_end:
13759 \group_end:
13760 }
13761 \cs_gset_eq:NN
13762 \markdownRendererBracketedSpanAttributeContextBeginPrototype
13763 \markdownRendererHeaderAttributeContextBeginPrototype
13764 \cs_gset_eq:NN
13765 \markdownRendererBracketedSpanAttributeContextEndPrototype
13766 \markdownRendererHeaderAttributeContextEndPrototype
13767 \cs_gset_eq:NN
13768 \markdownRendererFencedDivAttributeContextBeginPrototype
13769 \markdownRendererHeaderAttributeContextBeginPrototype
13770 \cs_gset_eq:NN
13771 \markdownRendererFencedDivAttributeContextEndPrototype
13772 \markdownRendererHeaderAttributeContextEndPrototype
13773 \cs_gset_eq:NN
13774 \markdownRendererFencedCodeAttributeContextBeginPrototype
13775 \markdownRendererHeaderAttributeContextBeginPrototype
13776 \cs_gset_eq:NN
13777 \markdownRendererFencedCodeAttributeContextEndPrototype
13778 \markdownRendererHeaderAttributeContextEndPrototype
13779 \cs_gset:Npn
13780 \markdownRendererReplacementCharacterPrototype
13781 { \codepoint_str_generate:n { fffd } }
13782 \ExplSyntaxOff
13783 \def\markdownRendererSectionBeginPrototype{}%
13784 \def\markdownRendererSectionEndPrototype{}%
13785 \ExplSyntaxOn
13786 \cs_gset:Npn
13787 \markdownRendererWarningPrototype
13788 #1#2#3#4
13789 {
13790 \tl_set:Nn
13791 \l_tmpa_tl
13792 { #2 }
13793 \tl_if_empty:nF
13794 { #4 }
13795 {
13796 \tl_put_right:Nn
13797 \l_tmpa_tl
13798 { \iow_newline: #4 }

```



```

13799 }
13800 \exp_args:NV
13801 \markdownWarning
13802 \l_tmpa_tl
13803 }
13804 \ExplSyntaxOff
13805 \def\markdownRendererErrorPrototype#1#2#3#4{%
13806 \markdownError{#2}{#4}}%

```

### 3.2.3.1 Raw Attributes

In the raw block and inline raw span renderer prototypes, execute the content with TeX when the raw attribute is `tex`, display the content as markdown when the raw attribute is `md`, and ignore the content otherwise.

```

13807 \ExplSyntaxOn
13808 \cs_new:Nn
13809 \@@_plain_tex_default_input_raw_inline:nn
13810 {
13811 \str_case:nn
13812 { #2 }
13813 {
13814 { md } { \markdownInput{#1} }
13815 { tex } { \markdownEscape{#1} \unskip }
13816 }
13817 }
13818 \cs_new:Nn
13819 \@@_plain_tex_default_input_raw_block:nn
13820 {
13821 \str_case:nn
13822 { #2 }
13823 {
13824 { md } { \markdownInput{#1} }
13825 { tex } { \markdownEscape{#1} }
13826 }
13827 }
13828 \cs_gset:Npn
13829 \markdownRendererInputRawInlinePrototype#1#2
13830 {
13831 \@@_plain_tex_default_input_raw_inline:nn
13832 { #1 }
13833 { #2 }
13834 }
13835 \cs_gset:Npn
13836 \markdownRendererInputRawBlockPrototype#1#2
13837 {
13838 \@@_plain_tex_default_input_raw_block:nn
13839 { #1 }

```

```

13840 { #2 }
13841 }
13842 \ExplSyntaxOff

```

### 3.2.3.2 Simple YAML Metadata Renderer Prototypes

In this section, we implement the simple high-level interface for processing simple YAML metadata using the key-value [markdown/jekyllData](#). See also Section 2.2.6.1.

To keep track of the current type of structure we inhabit when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_datatypes_seq` stack. At every step of the traversal, the stack will contain one of the following constants at any position  $p$ :

`\c_@@_jekyll_data_sequence_tl` The currently traversed branch of the YAML document contains a sequence at depth  $p$ .

`\c_@@_jekyll_data_mapping_tl` The currently traversed branch of the YAML document contains a mapping at depth  $p$ .

`\c_@@_jekyll_data_scalar_tl` The currently traversed branch of the YAML document contains a scalar value at depth  $p$ .

```

13843 \ExplSyntaxOn
13844 \seq_new:N \g_@@_jekyll_data_datatypes_seq
13845 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
13846 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
13847 \tl_const:Nn \c_@@_jekyll_data_scalar_tl { scalar }

```

To keep track of our current place when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_wildcard_absolute_address_seq` stack of keys using the `\@@_jekyll_data_push_address_segment:n` macro.

```

13848 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
13849 \cs_new:Nn \@@_jekyll_data_push_address_segment:n
13850 {
13851 \seq_if_empty:NF
13852 \g_@@_jekyll_data_datatypes_seq
13853 {
13854 \seq_get_right:NN
13855 \g_@@_jekyll_data_datatypes_seq
13856 \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.

```

13857 \str_if_eq:NNTF

```

```

13858 \l_tmpa_tl
13859 \c_@@_jekyll_data_sequence_tl
13860 {
13861 \seq_put_right:Nn
13862 \g_@@_jekyll_data_wildcard_absolute_address_seq
13863 { * }
13864 }
13865 {
13866 \seq_put_right:Nn
13867 \g_@@_jekyll_data_wildcard_absolute_address_seq
13868 { #1 }
13869 }
13870 }
13871 }

```

Out of `\g_@@_jekyll_data_wildcard_absolute_address_seq`, we will construct the following two token lists:

`\g_@@_jekyll_data_wildcard_absolute_address_tl` An *absolute wildcard*: The wildcard from the root of the document prefixed with a slash (/) with individual keys and asterisks also delimited by slashes. Allows the users to react to complex context-sensitive structures with ease.

For example, the `name` key in the following YAML document would correspond to the `/*/person/name` absolute wildcard:

```
[{person: {name: Elon, surname: Musk}}]
```

`\g_@@_jekyll_data_wildcard_relative_address_tl` A *relative wildcard*: The rightmost segment of the wildcard. Allows the users to react to simple context-free structures.

For example, the `name` key in the following YAML document would correspond to the `name` relative wildcard:

```
[{person: {name: Elon, surname: Musk}}]
```

We will construct `\g_@@_jekyll_data_wildcard_absolute_address_tl` using the `\@@_jekyll_data_concatenate_address:NN` macro and we will construct both token lists using the `\@@_jekyll_data_update_address_tls:` macro.

```

13872 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
13873 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
13874 \cs_new:Nn \@@_jekyll_data_concatenate_address:NN
13875 {
13876 \seq_pop_left:NN #1 \l_tmpa_tl
13877 \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }

```

```

13878 \seq_put_left:NV #1 \l_tmpa_tl
13879 }
13880 \cs_new:Nn \@@_jekyll_data_update_address_tls:
13881 {
13882 \@@_jekyll_data_concatenate_address:NN
13883 \g_@@_jekyll_data_wildcard_absolute_address_seq
13884 \g_@@_jekyll_data_wildcard_absolute_address_tl
13885 \seq_get_right:NN
13886 \g_@@_jekyll_data_wildcard_absolute_address_seq
13887 \g_@@_jekyll_data_wildcard_relative_address_tl
13888 }

```

To make sure that the stacks and token lists stay in sync, we will use the `\@@_jekyll_data_push:nN` and `\@@_jekyll_data_pop:` macros.

```

13889 \cs_new:Nn \@@_jekyll_data_push:nN
13890 {
13891 \@@_jekyll_data_push_address_segment:n
13892 { #1 }
13893 \seq_put_right:NV
13894 \g_@@_jekyll_data_datatypes_seq
13895 #2
13896 \@@_jekyll_data_update_address_tls:
13897 }
13898 \cs_new:Nn \@@_jekyll_data_pop:
13899 {
13900 \seq_pop_right:NN
13901 \g_@@_jekyll_data_wildcard_absolute_address_seq
13902 \l_tmpa_tl
13903 \seq_pop_right:NN
13904 \g_@@_jekyll_data_datatypes_seq
13905 \l_tmpa_tl
13906 \@@_jekyll_data_update_address_tls:
13907 }

```

To set a single key–value, we will use the `\@@_jekyll_data_set_keyval_known:nn` macro, ignoring unknown keys. To set key–values for both absolute and relative wildcards, we will use the `\@@_jekyll_data_set_keyvals_known:nn` macro.

```

13908 \cs_new:Nn \@@_jekyll_data_set_keyval_known:nn
13909 {
13910 \keys_set_known:nn
13911 { markdown/jekyllData }
13912 { { #1 } = { #2 } }
13913 }
13914 \cs_generate_variant:Nn
13915 \@@_jekyll_data_set_keyval_known:nn
13916 { Vn }
13917 \cs_new:Nn \@@_jekyll_data_set_keyvals_known:nn
13918 {

```

```

13919 \@@_jekyll_data_push:nN
13920 { #1 }
13921 \c_@@_jekyll_data_scalar_tl
13922 \@@_jekyll_data_set_keyval_known:Vn
13923 \g_@@_jekyll_data_wildcard_absolute_address_tl
13924 { #2 }
13925 \@@_jekyll_data_set_keyval_known:Vn
13926 \g_@@_jekyll_data_wildcard_relative_address_tl
13927 { #2 }
13928 \@@_jekyll_data_pop:
13929 }

```

Finally, we will register our macros as token renderer prototypes to be able to react to the traversal of a YAML document.

```

13930 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
13931 \@@_jekyll_data_push:nN
13932 { #1 }
13933 \c_@@_jekyll_data_sequence_tl
13934 }
13935 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
13936 \@@_jekyll_data_push:nN
13937 { #1 }
13938 \c_@@_jekyll_data_mapping_tl
13939 }
13940 \def\markdownRendererJekyllDataSequenceEndPrototype{
13941 \@@_jekyll_data_pop:
13942 }
13943 \def\markdownRendererJekyllDataMappingEndPrototype{
13944 \@@_jekyll_data_pop:
13945 }
13946 \def\markdownRendererJekyllDataBooleanPrototype#1#2{
13947 \@@_jekyll_data_set_keyvals_known:nn
13948 { #1 }
13949 { #2 }
13950 }
13951 \def\markdownRendererJekyllDataEmptyPrototype#1{}
13952 \def\markdownRendererJekyllDataNumberPrototype#1#2{
13953 \@@_jekyll_data_set_keyvals_known:nn
13954 { #1 }
13955 { #2 }
13956 }

```

We will process all string scalar values assuming that they may contain markdown markup and are intended for typesetting.

```

13957 \def\markdownRendererJekyllDataProgrammaticStringPrototype#1#2{}
13958 \def\markdownRendererJekyllDataTypographicStringPrototype#1#2{
13959 \@@_jekyll_data_set_keyvals_known:nn
13960 { #1 }

```

```

13961 { #2 }
13962 }
13963 \ExplSyntaxOff

```

### 3.2.3.3 Complex YAML Metadata Renderer Prototypes

In this section, we implement the high-level interface for routing complex YAML metadata to expl3 key-values using the option `jeekyllDataKeyValue=<module>`. See also Section 2.2.6.1.

```

13964 \ExplSyntaxOn
13965 \@@_with_various_cases:nn
13966 { jeekyllDataKeyValue }
13967 {
13968 \keys_define:nn
13969 { markdown/options }
13970 {
13971 #1 .code:n = {
13972 \@@_route_jeekyll_data_to_key_values:n
13973 { ##1 }
13974 },

```

When no `<module>` has been provided, assume that the YAML metadata specify absolute paths to key-values.

```

13975 #1 .default:n = { },
13976 }
13977 }
13978 \seq_new:N
13979 \l_@@_jeekyll_data_current_position_seq
13980 \tl_new:N
13981 \l_@@_jeekyll_data_current_position_tl
13982 \cs_new:Nn
13983 \@@_route_jeekyll_data_to_key_values:n
13984 {
13985 \markdownSetup
13986 {
13987 renderers = {
13988 jeekyllData(Sequence|Mapping)Begin = {
13989 \bool_lazy_and:nnTF
13990 {
13991 \seq_if_empty_p:N
13992 \l_@@_jeekyll_data_current_position_seq
13993 }
13994 {
13995 \str_if_eq_p:nn
13996 { ##1 }
13997 { null }
13998 }

```

```

13999 {
14000 \tl_if_empty:nF
14001 { #1 }
14002 {
14003 \seq_put_right:Nn
14004 \l_@@_jekyll_data_current_position_seq
14005 { #1 }
14006 }
14007 }
14008 {
14009 \seq_put_right:Nn
14010 \l_@@_jekyll_data_current_position_seq
14011 { ##1 }
14012 }
14013 },
14014 jekyllData(Sequence|Mapping)End = {
14015 \seq_pop_right:NN
14016 \l_@@_jekyll_data_current_position_seq
14017 \l_tmpa_tl
14018 },

```

For every YAML key `path.to.<key>` with a value of type *<non-string type>*, set the key *<non-string type>* of the key-value *<module>/path/to/<key>* if it is known and the key *<key>* of the key-value *<module>/path/to* otherwise. *<Non-string type>* is one of `boolean`, `number`, and `empty`.

```

14019 jekyllDataBoolean = {
14020 \tl_set:Nx
14021 \l_@@_jekyll_data_current_position_tl
14022 {
14023 \seq_use:Nn
14024 \l_@@_jekyll_data_current_position_seq
14025 { / }
14026 }
14027 \keys_if_exist:VnTF
14028 \l_@@_jekyll_data_current_position_tl
14029 { ##1 / boolean }
14030 {
14031 \@@_keys_set:xn
14032 {
14033 \tl_use:N
14034 \l_@@_jekyll_data_current_position_tl
14035 / ##1 / boolean
14036 }
14037 { ##2 }
14038 }
14039 {
14040 \@@_keys_set:xn

```

```

14041 {
14042 \tl_use:N
14043 \l_@@_jekyll_data_current_position_tl
14044 / ##1
14045 }
14046 { ##2 }
14047 }
14048 },
14049 jekyllDataNumber = {
14050 \tl_set:Nx
14051 \l_@@_jekyll_data_current_position_tl
14052 {
14053 \seq_use:Nn
14054 \l_@@_jekyll_data_current_position_seq
14055 { / }
14056 }
14057 \keys_if_exist:VnTF
14058 \l_@@_jekyll_data_current_position_tl
14059 { ##1 / number }
14060 {
14061 \@@_keys_set:xn
14062 {
14063 \tl_use:N
14064 \l_@@_jekyll_data_current_position_tl
14065 / ##1 / number
14066 }
14067 { ##2 }
14068 }
14069 {
14070 \@@_keys_set:xn
14071 {
14072 \tl_use:N
14073 \l_@@_jekyll_data_current_position_tl
14074 / ##1
14075 }
14076 { ##2 }
14077 }
14078 },

```

For the *⟨non-string type⟩* of `empty`, no value is passed to the key–value. Therefore, a default value should always be defined for nullable keys using the key property `.default:n`.

```

14079 jekyllDataEmpty = {
14080 \tl_set:Nx
14081 \l_@@_jekyll_data_current_position_tl
14082 {
14083 \seq_use:Nn

```



```

14084 \l_@@_jekyll_data_current_position_seq
14085 { / }
14086 }
14087 \keys_if_exist:VnTF
14088 \l_@@_jekyll_data_current_position_tl
14089 { ##1 / empty }
14090 {
14091 \keys_set:xn
14092 {
14093 \tl_use:N
14094 \l_@@_jekyll_data_current_position_tl
14095 / ##1
14096 }
14097 { empty }
14098 }
14099 {
14100 \keys_set:Vn
14101 \l_@@_jekyll_data_current_position_tl
14102 { ##1 }
14103 }
14104 },

```

For every YAML key `path.to.<key>` with a value of type `string`, set the keys `typographicString` and `programmaticString` of the key-value `<module>/path/to/<key>` if they are known with the typographic and programmatic strings of the value, respectively. Furthermore, set the key `<key>` of the key-value `<module>/path/to` with the typographic string of the value unless the key `typographicString` is known. If the key `programmaticString` is known, only set the key `<key>` if it is known. In contrast, if neither `typographicString` nor `programmaticString` are known, set `<key>` normally, i.e. regardless of whether it is known or unknown.

```

14105 jekyllDataTypographicString = {
14106 \tl_set:Nx
14107 \l_@@_jekyll_data_current_position_tl
14108 {
14109 \seq_use:Nn
14110 \l_@@_jekyll_data_current_position_seq
14111 { / }
14112 }
14113 \keys_if_exist:VnTF
14114 \l_@@_jekyll_data_current_position_tl
14115 { ##1 / typographicString }
14116 {
14117 \@@_keys_set:xn
14118 {
14119 \tl_use:N

```

```

14120 \l_@@_jekyll_data_current_position_tl
14121 / ##1 / typographicString
14122 }
14123 { ##2 }
14124 }
14125 {
14126 \keys_if_exist:VnTF
14127 \l_@@_jekyll_data_current_position_tl
14128 { ##1 / programmaticString }
14129 {
14130 \@@_keys_set_known:xn
14131 {
14132 \tl_use:N
14133 \l_@@_jekyll_data_current_position_tl
14134 / ##1
14135 }
14136 { ##2 }
14137 }
14138 {
14139 \@@_keys_set:xn
14140 {
14141 \tl_use:N
14142 \l_@@_jekyll_data_current_position_tl
14143 / ##1
14144 }
14145 { ##2 }
14146 }
14147 }
14148 },
14149 jekyllDataProgrammaticString = {
14150 \tl_set:Nx
14151 \l_@@_jekyll_data_current_position_tl
14152 {
14153 \seq_use:Nn
14154 \l_@@_jekyll_data_current_position_seq
14155 { / }
14156 }
14157 \keys_if_exist:VnT
14158 \l_@@_jekyll_data_current_position_tl
14159 { ##1 / programmaticString }
14160 {
14161 \@@_keys_set:xn
14162 {
14163 \tl_use:N
14164 \l_@@_jekyll_data_current_position_tl
14165 / ##1 / programmaticString
14166 }

```

```

14167 { ##2 }
14168 }
14169 },
14170 },
14171 }
14172 }
14173 \cs_new:Nn
14174 \@@_keys_set:nn
14175 {
14176 \keys_set:nn
14177 { }
14178 { { #1 } = { #2 } }
14179 }
14180 \cs_new:Nn
14181 \@@_keys_set_known:nn
14182 {
14183 \keys_set_known:nn
14184 { }
14185 { { #1 } = { #2 } }
14186 }
14187 \cs_generate_variant:Nn
14188 \@@_keys_set:nn
14189 { xn }
14190 \cs_generate_variant:Nn
14191 \@@_keys_set_known:nn
14192 { xn }
14193 \cs_generate_variant:Nn
14194 \keys_set:nn
14195 { xn, Vn }
14196 \prg_generate_conditional_variant:Nnn
14197 \keys_if_exist:nn
14198 { Vn }
14199 { T, TF }
14200 \ExplSyntaxOff

```

If plain T<sub>E</sub>X is the top layer, we load the [witiko/markdown/defaults](#) plain T<sub>E</sub>X theme with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```

14201 \ExplSyntaxOn
14202 \str_if_eq:VVT
14203 \c_@@_top_layer_tl
14204 \c_@@_option_layer_plain_tex_tl
14205 {
14206 \use:c
14207 { ExplSyntaxOff }
14208 \@@_if_option:nF
14209 { noDefaults }

```

```

14210 {
14211 \@@_if_option:nTF
14212 { experimental }
14213 {
14214 \@@_setup:n
14215 { theme = witiko/markdown/defaults@experimental }
14216 }
14217 {
14218 \@@_setup:n
14219 { theme = witiko/markdown/defaults }
14220 }
14221 }
14222 \use:c
14223 { ExplSyntaxOn }
14224 }
14225 \ExplSyntaxOff

```

### 3.2.4 Lua Snippets

After the `\markdownPrepareLuaOptions` macro has been fully expanded, the `\markdownLuaOptions` macro will expand to a Lua table that contains the plain  $\text{\TeX}$  options (see Section 2.2.2) in a format recognized by Lua (see Section 2.1.3).

```

14226 \ExplSyntaxOn
14227 \tl_new:N \g_@@_formatted_lua_options_tl
14228 \cs_new:Nn \@@_format_lua_options:
14229 {
14230 \tl_gclear:N
14231 \g_@@_formatted_lua_options_tl
14232 \seq_map_function:NN
14233 \g_@@_lua_options_seq
14234 \@@_format_lua_option:n
14235 }
14236 \cs_new:Nn \@@_format_lua_option:n
14237 {
14238 \@@_typecheck_option:n
14239 { #1 }
14240 \@@_get_option_type:nN
14241 { #1 }
14242 \l_tmpa_tl
14243 \bool_case_true:nF
14244 {
14245 {
14246 \str_if_eq_p:VV
14247 \l_tmpa_tl
14248 \c_@@_option_type_boolean_tl ||
14249 \str_if_eq_p:VV

```

```

14250 \l_tmpa_tl
14251 \c_@@_option_type_number_tl ||
14252 \str_if_eq_p:VV
14253 \l_tmpa_tl
14254 \c_@@_option_type_counter_tl
14255 }
14256 {
14257 \@@_get_option_value:nN
14258 { #1 }
14259 \l_tmpa_tl
14260 \tl_gput_right:Nx
14261 \g_@@_formatted_lua_options_tl
14262 { #1~~~ \l_tmpa_tl ,~ }
14263 }
14264 {
14265 \str_if_eq_p:VV
14266 \l_tmpa_tl
14267 \c_@@_option_type_clist_tl
14268 }
14269 {
14270 \@@_get_option_value:nN
14271 { #1 }
14272 \l_tmpa_tl
14273 \tl_gput_right:Nx
14274 \g_@@_formatted_lua_options_tl
14275 { #1~~~\c_left_brace_str }
14276 \clist_map_inline:Vn
14277 \l_tmpa_tl
14278 {
14279 \@@_lua_escape:xN
14280 { ##1 }
14281 \l_tmpb_tl
14282 \tl_gput_right:Nn
14283 \g_@@_formatted_lua_options_tl
14284 { " }
14285 \tl_gput_right:NV
14286 \g_@@_formatted_lua_options_tl
14287 \l_tmpb_tl
14288 \tl_gput_right:Nn
14289 \g_@@_formatted_lua_options_tl
14290 { " ,~ }
14291 }
14292 \tl_gput_right:Nx
14293 \g_@@_formatted_lua_options_tl
14294 { \c_right_brace_str ,~ }
14295 }
14296 }

```

```

14297 {
14298 \@@_get_option_value:nN
14299 { #1 }
14300 \l_tmpa_tl
14301 \@@_lua_escape:xN
14302 { \l_tmpa_tl }
14303 \l_tmpb_tl
14304 \tl_gput_right:Nn
14305 \g_@@_formatted_lua_options_tl
14306 { #1~~ " }
14307 \tl_gput_right:NV
14308 \g_@@_formatted_lua_options_tl
14309 \l_tmpb_tl
14310 \tl_gput_right:Nn
14311 \g_@@_formatted_lua_options_tl
14312 { " ,~ }
14313 }
14314 }
14315 \cs_generate_variant:Nn
14316 \clist_map_inline:nn
14317 { Vn }
14318 \let
14319 \markdownPrepareLuaOptions
14320 \@@_format_lua_options:
14321 \def
14322 \markdownLuaOptions
14323 {
14324 {
14325 \g_@@_formatted_lua_options_tl
14326 }
14327 }
14328 \sys_if_engine luatex:TF
14329 {
14330 \cs_new:Nn
14331 \@@_lua_escape:nN
14332 {
14333 \tl_set:Nx
14334 #2
14335 {
14336 \lua_escape:n
14337 { #1 }
14338 }
14339 }
14340 }
14341 {
14342 \regex_const:Nn
14343 \c_@@_lua_escape_regex

```

```

14344 { [\\\"'] }
14345 \cs_new:Nn
14346 \@@_lua_escape:nN
14347 {
14348 \tl_set:Nn
14349 #2
14350 { #1 }
14351 \regex_replace_all:NnN
14352 \c_@@_lua_escape_regex
14353 { \u { c_backslash_str } \0 }
14354 #2
14355 }
14356 }
14357 \cs_generate_variant:Nn
14358 \@@_lua_escape:nN
14359 { xN }

```

After the `\markdownPrepareInputFilename` macro has been fully expanded, the `\markdownInputFilename` macro will expand to a Lua string that contains the input filename passed as the first argument.

```

14360 \tl_new:N
14361 \markdownInputFilename
14362 \cs_new:Npn
14363 \markdownPrepareInputFilename
14364 #1
14365 {
14366 \@@_lua_escape:xN
14367 { #1 }
14368 \markdownInputFilename
14369 \tl_gset:Nx
14370 \markdownInputFilename
14371 { " \markdownInputFilename " }
14372 }

```

The `\markdownPrepare` macro contains the Lua code that is executed prior to any conversion from markdown to plain T<sub>E</sub>X. It exposes the `convert` function for the use by any further Lua code.

```

14373 \cs_new:Npn
14374 \markdownPrepare
14375 {

```

First, ensure that the `cacheDir` directory exists.

```

14376 local~lfs = require("lfs")
14377 local~options = \markdownLuaOptions
14378 if~not~lfs.isdir(options.cacheDir) then~
14379 assert(lfs.mkdir(options.cacheDir))
14380 end~

```

Next, load the `markdown` module and create a converter function using the plain T<sub>E</sub>X options, which were serialized to a Lua table via the `\markdownLuaOptions` macro.

```
14381 local~md = require("markdown")
14382 local~convert = md.new(options)
14383 }
```

The `\markdownConvert` macro contains the Lua code that is executed during the conversion from markdown to plain T<sub>E</sub>X. It opens the input file, converts it, and prints the conversion result.

```
14384 \cs_new:Npn
14385 \markdownConvert
14386 {
14387 local~filename = \markdownInputFilename
14388 local~file = assert(io.open(filename, "r"),
14389 [[Could~not~open~file~]] .. filename .. [[~for~reading]])
14390 local~input = assert(file:read("*a"))
14391 assert(file:close())
14392 print(convert(input))
14393 }
14394 \ExplSyntaxOff
```

The `\markdownCleanup` macro contains the Lua code that is executed after any conversion from markdown to plain T<sub>E</sub>X.

```
14395 \def\markdownCleanup{%
Remove the options.cacheDir directory if it is empty.
14396 if options.cacheDir then
14397 lfs.rmdir(options.cacheDir)
14398 end
14399 }%
```

### 3.2.5 Buffering Block-Level Markdown Input

The macros `\markdownInputFileStream` and `\markdownOutputFileStream` contain the number of the input and output file streams that will be used for the IO operations of the package.

```
14400 \csname newread\endcsname\markdownInputFileStream
14401 \csname newwrite\endcsname\markdownOutputFileStream
```

The `\markdownReadAndConvertTab` macro contains the tab character literal.

```
14402 \begingroup
14403 \catcode\^^I=12%
14404 \gdef\markdownReadAndConvertTab{^^I}%
14405 \endgroup
```

The `\markdownReadAndConvert` macro is largely a rewrite of the L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> `\filecontents` macro to plain T<sub>E</sub>X.

```
14406 \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.

```

14407 \catcode\^M=13%
14408 \catcode\^I=13%
14409 \catcode|=0%
14410 \catcode\=12%
14411 |catcode@=14%
14412 |catcode|=12@
14413 |gdef|markdownReadAndConvert#1#2{@
14414 |begingroup@

```

If we are not reading markdown documents from the frozen cache, open the `inputTempFileName` file for writing.

```

14415 |markdownIfOption{frozenCache}{-}{@
14416 |immediate|openout|markdownOutputFileStream@
14417 "|markdownOptionInputTempFileName"|relax@
14418 |markdownInfo{@
14419 Buffering block-level markdown input into the temporary @
14420 input file "|markdownOptionInputTempFileName" and scanning @
14421 for the closing token sequence "#1"}@
14422 }@

```

Locally change the category of the special plain T<sub>E</sub>X characters to *other* in order to prevent unwanted interpretation of the input. Change also the category of the space character, so that we can retrieve it unaltered.

```

14423 |def|do##1{|catcode##1=12}|dospecials@
14424 |catcode|=12@
14425 |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.

```

14426 |def|markdownReadAndConvertStripPercentSign##1{@
14427 |markdownIfOption{stripPercentSigns}{-}{@
14428 |if##1%@
14429 |expandafter|expandafter|expandafter@
14430 |markdownReadAndConvertProcessLine@
14431 |else@
14432 |expandafter|expandafter|expandafter@
14433 |markdownReadAndConvertProcessLine@
14434 |expandafter|expandafter|expandafter##1@
14435 |fi@
14436 }{@

```

```

14437 |expandafter@
14438 |markdownReadAndConvertProcessLine@
14439 |expandafter##1@
14440 }@
14441 }@

```

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.

```

14442 |def|markdownReadAndConvertProcessLine##1#1##2#1##3|relax{@

```

If we are not reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, store the line in the `inputTempFileName` file. If we are reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, gobble the line.

```

14443 |ifx|relax##3|relax@
14444 |markdownIfOption{frozenCache}{-}{@
14445 |immediate|write|markdownOutputFileStream{##1}@
14446 }@
14447 |else@

```

When the ending token sequence appears in the line, make the next newline character close the `inputTempFileName` file, return the character categories back to the former state, convert the `inputTempFileName` file from markdown to plain T<sub>E</sub>X, `\input` the result of the conversion, and expand the ending control sequence.

```

14448 |def^^M{@
14449 |markdownInfo{The ending token sequence was found}@
14450 |markdownIfOption{frozenCache}{-}{@
14451 |immediate|closeout|markdownOutputFileStream@
14452 }@
14453 |endgroup@
14454 |markdownInput{@
14455 |markdownOptionOutputDir@
14456 /|markdownOptionInputTempFileName@
14457 }@
14458 #2}@
14459 |fi@

```

Repeat with the next line.

```

14460 ^^M}@

```

Make the tab character active at expansion time and make it expand to a literal tab character.

```

14461 |catcode`|^I=13@
14462 |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.

```

14463 |catcode`\^^M=13@
14464 |def^^M##1^^M{@
14465 |def^^M###1^^M{@
14466 |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
14467 ^^M}@
14468 ^^M}@

```

Reset the character categories back to the former state.

```
14469 |endgroup
```

Use the `lt3luabridge` library to define the `\markdownLuaExecute` macro, which takes in a Lua scripts and expands to the standard output produced by its execution.

```

14470 \ExplSyntaxOn
14471 \cs_new:Npn
14472 \markdownLuaExecute
14473 #1
14474 {
14475 \int_compare:nNt
14476 { \g_luabridge_method_int }
14477 =
14478 { \c_luabridge_method_shell_int }
14479 {
14480 \sys_if_shell_unrestricted:F
14481 {
14482 \sys_if_shell:TF
14483 {
14484 \msg_error:nn
14485 { markdown }
14486 { restricted-shell-access }
14487 }
14488 {
14489 \msg_error:nn
14490 { markdown }
14491 { disabled-shell-access }
14492 }
14493 }
14494 }
14495 \str_gset:NV
14496 \g_luabridge_output_dirname_str
14497 \markdownOptionOutputDir
14498 \luabridge_now:e
14499 { #1 }
14500 }
14501 \cs_generate_variant:Nn
14502 \msg_new:nnnn
14503 { nnnV }
14504 \tl_set:Nn
14505 \l_tmpa_tl

```

```

14506 {
14507 You~may~need~to~run~TeX~with~the~---shell-escape~or~the~
14508 --enable-write18~flag,~or~write~shell_escape=t~in~the~
14509 texmf.cnf~file.
14510 }
14511 \msg_new:nnnV
14512 { markdown }
14513 { restricted-shell-access }
14514 { Shell-escape-is-restricted }
14515 \l_tmpa_tl
14516 \msg_new:nnnV
14517 { markdown }
14518 { disabled-shell-access }
14519 { Shell-escape-is-disabled }
14520 \l_tmpa_tl
14521 \ExplSyntaxOff

```

### 3.2.6 Buffering Inline Markdown Input

This section describes the implementation of the macro `\markinline`.

```

14522 \ExplSyntaxOn
14523 \tl_new:N
14524 \g_@@_after_markinline_tl
14525 \tl_gset:Nn
14526 \g_@@_after_markinline_tl
14527 { \unskip }
14528 \cs_new:Npn
14529 \markinline
14530 {

```

Locally change the category of the special plain  $\TeX$  characters to *other* in order to prevent unwanted interpretation of the input markdown text as  $\TeX$  code.

```

14531 \group_begin:
14532 \cctab_select:N
14533 \c_other_cctab

```

Unless we are reading markdown documents from the frozen cache, open the file `inputTempFileName` for writing.

```

14534 \@@_if_option:nF
14535 { frozenCache }
14536 {
14537 \immediate
14538 \openout
14539 \markdownOutputFileStream
14540 "\markdownOptionInputTempFileName"
14541 \relax
14542 \msg_info:nne

```

```

14543 { markdown }
14544 { buffering-markinline }
14545 { \markdownOptionInputTempFileName }
14546 }

```

Peek ahead and extract the inline markdown text.

```

14547 \peek_regex_replace_once:nnF
14548 { { (.*) } }
14549 {

```

Unless we are reading markdown documents from the frozen cache, store the text in the file `inputTempFileName` and close it.

```

14550 \c { @@_if_option:nF }
14551 \cB { frozenCache \cE }
14552 \cB {
14553 \c { immediate }
14554 \c { write }
14555 \c { markdownOutputFileStream }
14556 \cB { \1 \cE }
14557 \c { immediate }
14558 \c { closeout }
14559 \c { markdownOutputFileStream }
14560 \cE }

```

Reset the category codes and `\input` the result of the conversion.

```

14561 \c { group_end: }
14562 \c { group_begin: }
14563 \c { @@_setup:n }
14564 \cB { contentLevel = inline \cE }
14565 \c { markdownInput }
14566 \cB {
14567 \c { markdownOptionOutputDir } /
14568 \c { markdownOptionInputTempFileName }
14569 \cE }
14570 \c { group_end: }
14571 \c { tl_use:N }
14572 \c { g_@@_after_markinline_tl }
14573 }
14574 {
14575 \msg_error:nn
14576 { markdown }
14577 { markinline-peek-failure }
14578 \group_end:
14579 \tl_use:N
14580 \g_@@_after_markinline_tl
14581 }
14582 }
14583 \msg_new:nnn

```

```

14584 { markdown }
14585 { buffering-markinline }
14586 { Buffering~inline~markdown~input~into~
14587 the~temporary~input~file~"#1". }
14588 \msg_new:nnnn
14589 { markdown }
14590 { markinline-peek-failure }
14591 { Use~of~\iow_char:N \\ markinline~doesn't~match~its~definition }
14592 { The~macro~should~be~followed~by~inline~
14593 markdown~text~in~curly~braces }
14594 \ExplSyntaxOff

```

### 3.2.7 Typesetting Markdown

The `\markdownInput` macro uses an implementation of the `\markdownLuaExecute` macro to convert the contents of the file whose filename it has received as its single argument from markdown to plain T<sub>E</sub>X.

```

14595 \ExplSyntaxOn
14596 \cs_new:Npn
14597 \markdownInput
14598 #1
14599 {
14600 \@@_if_option:nTF
14601 { frozenCache }
14602 {
14603 \markdownInputRaw
14604 { #1 }
14605 }
14606 {

```

If the file does not exist in the current directory, we will search for it in the directories specified in `\l_file_search_path_seq`. On L<sup>A</sup>T<sub>E</sub>X, this also includes the directories specified in `\input@path`.

```

14607 \tl_set:Nx
14608 \l_tmpa_tl
14609 { #1 }
14610 \file_get_full_name:VNTF
14611 \l_tmpa_tl
14612 \l_tmpb_tl
14613 {
14614 \exp_args:NV
14615 \markdownInputRaw
14616 \l_tmpb_tl
14617 }
14618 {
14619 \msg_error:nnV
14620 { markdown }

```

```

14621 { markdown-file-does-not-exist }
14622 \l_tmpa_tl
14623 }
14624 }
14625 }
14626 \msg_new:nnn
14627 { markdown }
14628 { markdown-file-does-not-exist }
14629 {
14630 Markdown~file~#1~does~not~exist
14631 }
14632 \ExplSyntaxOff
14633 \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.

```

14634 \catcode`\=0%
14635 \catcode`\=12%
14636 \catcode`\&=6%
14637 |gdef|markdownInputRaw#1{%

```

Change the category code of the percent sign (%) to other, so that a user of the [hybrid](#) Lua option or a malevolent actor can't produce TeX comments in the plain TeX output of the Markdown package.

```

14638 |begingroup
14639 |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.

```

14640 |catcode`\#=12

```

If we are reading from the frozen cache, input it, expand the corresponding `\markdownFrozenCache⟨number⟩` macro, and increment `frozenCacheCounter`.

```

14641 |markdownIfOption{frozenCache}{%
14642 |ifnum|markdownOptionFrozenCacheCounter=0|relax
14643 |markdownInfo{Reading frozen cache from
14644 "||markdownOptionFrozenCacheFileName"}%
14645 |input|markdownOptionFrozenCacheFileName|relax
14646 |fi
14647 |markdownInfo{Including markdown document number
14648 "|the|markdownOptionFrozenCacheCounter" from frozen cache}%
14649 |csname markdownFrozenCache%
14650 |the|markdownOptionFrozenCacheCounter|endcsname
14651 |global|advance|markdownOptionFrozenCacheCounter by 1|relax
14652 }{%
14653 |markdownInfo{Including markdown document "&1"}%

```

Attempt to open the markdown document to record it in the `.log` and `.fls` files. This allows external programs such as L<sup>A</sup>T<sub>E</sub>X<sub>M</sub>k to track changes to the markdown document.

```

14654 |openin|markdownInputFileStream{&1}%
14655 |closein|markdownInputFileStream
14656 |markdownPrepareLuaOptions
14657 |markdownPrepareInputFilename{&1}%
14658 |markdownLuaExecute{%
14659 |markdownPrepare
14660 |markdownConvert
14661 |markdownCleanup}%

```

If we are finalizing the frozen cache, increment `frozenCacheCounter`.

```

14662 |markdownIfOption{finalizeCache}{}%
14663 |global|advance|markdownOptionFrozenCacheCounter by 1|relax}{}%
14664 }%
14665 |endgroup
14666 }%
14667 |endgroup

```

The `\markdownEscape` macro resets the category codes of the percent sign and the hash sign back to comment and parameter, respectively, before using the `\input` built-in of T<sub>E</sub>X to execute a T<sub>E</sub>X document in the middle of a markdown document fragment.

```

14668 \gdef\markdownEscape#1{%
14669 \catcode`\%=14\relax
14670 \catcode`\#=6\relax
14671 \input #1\relax
14672 \catcode`\%=12\relax
14673 \catcode`\#=12\relax
14674 }%

```

### 3.3 L<sup>A</sup>T<sub>E</sub>X Implementation

The L<sup>A</sup>T<sub>E</sub>X implementation makes use of the fact that, apart from some subtle differences, L<sup>A</sup>T<sub>E</sub>X implements the majority of the plain T<sub>E</sub>X format [17, Section 9]. As a consequence, we can directly reuse the existing plain T<sub>E</sub>X implementation.

```

14675 \def\markdownVersionSpace{ }%
14676 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
14677 \markdownVersion\markdownVersionSpace markdown renderer]%

```

#### 3.3.1 Typesetting Markdown

The `\markinlinePlainTeX` macro is used to store the original plain T<sub>E</sub>X implementation of the `\markinline` macro. The `\markinline` macro is then redefined to



accept an optional argument with options recognized by the L<sup>A</sup>T<sub>E</sub>X interface (see Section 2.3.3).

```

14678 \ExplSyntaxOn
14679 \cs_gset_eq:NN
14680 \markinlinePlainTeX
14681 \markinline
14682 \cs_gset:Npn
14683 \markinline
14684 {
14685 \peek_regex_replace_once:nn
14686 { (\[(.*?) \]) ? }
14687 {

```

Apply the options locally.

```

14688 \c { group_begin: }
14689 \c { @@_setup:n }
14690 \cB { \2 \cE }
14691 \c { tl_put_right:Nn }
14692 \c { g_@@_after_markinline_tl }
14693 \cB { \c { group_end: } \cE }
14694 \c { markinlinePlainTeX }
14695 }
14696 }
14697 \ExplSyntaxOff

```

The `\markdownInputPlainTeX` macro is used to store the original plain T<sub>E</sub>X implementation of the `\yamlInput` macro. The `\markdownInput` and `\yamlInput` macros are then redefined to accept an optional argument with options recognized by the L<sup>A</sup>T<sub>E</sub>X interface (see Section 2.3.3).

```

14698 \let\markdownInputPlainTeX\markdownInput
14699 \renewcommand\markdownInput[2] [] {%
14700 \begingroup
14701 \markdownSetup{#1}%
14702 \markdownInputPlainTeX{#2}%
14703 \endgroup}%
14704 \renewcommand\yamlInput[2] [] {%
14705 \begingroup
14706 \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData, #1}%
14707 \markdownInputPlainTeX{#2}%
14708 \endgroup}%

```

The `markdown`, `markdown*`, and `yaml` L<sup>A</sup>T<sub>E</sub>X environments are implemented using the `\markdownReadAndConvert` macro.

```

14709 \ExplSyntaxOn
14710 \renewenvironment
14711 { markdown }
14712 {

```

In our implementation of the `markdown` L<sup>A</sup>T<sub>E</sub>X environment, we want to distinguish between the following two cases:

<code>\begin{markdown} [smartEllipses]</code>	<code>\begin{markdown}</code>
<code>% This is an optional argument ^</code>	<code>[smartEllipses]</code>
<code>% ...</code>	<code>% ^ This is link</code>
<code>\end{markdown}</code>	<code>\end{markdown}</code>

Therefore, we cannot use the built-in L<sup>A</sup>T<sub>E</sub>X support for environments with optional arguments or packages such as `xparse`. Instead, we must read the optional argument manually and prevent reading past the end of a line.

To prevent reading past the end of a line when looking for the optional argument of the `markdown` L<sup>A</sup>T<sub>E</sub>X environment and accidentally tokenizing markdown text, we change the category code of carriage return (`\r`, ASCII character 13 in decimal) from 5 (end of line).

While any category code other than 5 (end of line) would work, we switch to the category 13 (active), which is also used by the `\markdownReadAndConvert` macro. This is necessary if we read until the end of a line, because then the carriage return character will be produced by T<sub>E</sub>X via the `\endlinechar` plain T<sub>E</sub>X macro and it needs to have the correct category code, so that `\markdownReadAndConvert` processes it correctly.

```
14713 \group_begin:
14714 \char_set_catcode_active:n { 13 }
```

To prevent doubling the hash signs (`#`, ASCII code 35 in decimal), we switch its category from 6 (parameter) to 12 (letter).

```
14715 \char_set_catcode_letter:n { 35 }
```

After we have matched the opening `[` that begins the optional argument, we accept carriage returns as well.

```
14716 \peek_regex_replace_once:nnF
14717 { \ *\[\r*([~]*)\][^~\r]* }
14718 {
```

After we have matched the optional argument, we switch back the category code of carriage returns and hash signs and we retokenize the content. This will cause single new lines to produce a space token and multiple new lines to produce `\par` tokens. Furthermore, this will cause hash signs followed by a number to be recognized as parameter numbers, which is necessary when we use the optional argument to redefine token renderers and token renderer prototypes.

```
14719 \c { group_end: }
14720 \c { tl_set_rescan:Nnn } \c { l_tmpa_tl } { } { \1 }
```

Then, we pass the retokenized content to the `\markdownSetup` macro.

```
14721 \c { @@_setup:V } \c { l_tmpa_tl }
```

Finally, regardless of whether or not we have matched the optional argument, we let the `\markdownReadAndConvert` macro process the rest of the  $\text{\LaTeX}$  environment.

We also make provision for using the `\markdown` command as a part of a different  $\text{\LaTeX}$  environment as follows:

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

```
14722 \c { exp_args:NV }
14723 \c { markdownReadAndConvert@ }
14724 \c { @currenvir }
14725 }
14726 {
14727 \group_end:
14728 \exp_args:NV
14729 \markdownReadAndConvert@
14730 \@currenvir
14731 }
14732 }
14733 { \markdownEnd }
14734 \renewenvironment
14735 { markdown* }
14736 [1]
14737 {
14738 \@@_if_option:nTF
14739 { experimental }
14740 {
14741 \msg_error:nnn
14742 { markdown }
14743 { latex-markdown-star-deprecated }
14744 { #1 }
14745 }
14746 {
14747 \msg_warning:nnn
14748 { markdown }
14749 { latex-markdown-star-deprecated }
14750 { #1 }
14751 }
14752 \@@_setup:n
14753 { #1 }
14754 \markdownReadAndConvert@
14755 { markdown* }
14756 }
14757 { \markdownEnd }
14758 \renewenvironment
```

```

14759 { yaml }
14760 {
14761 \group_begin:
14762 \yamlSetup
14763 { jekyllData, expectJekyllData, ensureJekyllData }
14764 \markdown
14765 }
14766 { \yamlEnd }
14767 \msg_new:nnn
14768 { markdown }
14769 { latex-markdown-star-deprecated }
14770 {
14771 The~markdown*~LaTeX~environment~has~been~deprecated~and~will~
14772 be~removed~in~the~next~major~version~of~the~Markdown~package.
14773 }
14774 \cs_generate_variant:Nn
14775 \@@_setup:n
14776 { V }
14777 \ExplSyntaxOff
14778 \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*.

```

14779 \catcode`\|=0\catcode`\<=1\catcode`\>=2%
14780 \catcode`\|=12\catcode`\{=12\catcode`\}=12%
14781 |gdef|markdownReadAndConvert@#1<%
14782 |markdownReadAndConvert<\end{#1}>%
14783 <|end<#1>>>%
14784 |endgroup

```

### 3.3.2 Themes

This section overrides the plain  $\TeX$  implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in  $\LaTeX$  themes provided with the Markdown package.

```

14785 \ExplSyntaxOn
14786 \prop_new:N \g_@@_latex_loaded_themes_linenos_prop
14787 \prop_new:N \g_@@_latex_loaded_themes_versions_prop
14788 \cs_gset:Nn
14789 \@@_load_theme:nnn
14790 {

```

If the Markdown package has not yet been loaded, determine whether either this is a built-in theme according to the prop `\g_@@_latex_built_in_themes_prop` or

a file named `markdowntheme<munged theme name>.sty` exists and whether we are still in the preamble.

```
14791 \ifmarkdownLaTeXLoaded
14792 \ifx\@onlypreamble\@notprerr
```

If both conditions are true, end with an error, since we cannot load L<sup>A</sup>T<sub>E</sub>X themes after the preamble.

```
14793 \bool_if:nTF
14794 {
14795 \bool_lazy_or_p:nn
14796 {
14797 \prop_if_in_p:Nn
14798 \g_@@_latex_built_in_themes_prop
14799 { #1 }
14800 }
14801 {
14802 \file_if_exist_p:n
14803 { markdown theme #3.sty }
14804 }
14805 }
14806 {
14807 \msg_error:nnn
14808 { markdown }
14809 { latex-theme-after-preamble }
14810 { #1 }
14811 }
```

Otherwise, try loading a plain T<sub>E</sub>X theme instead.

```
14812 {
14813 \@@_plain_tex_load_theme:nnn
14814 { #1 }
14815 { #2 }
14816 { #3 }
14817 }
14818 \else
```

If the Markdown package has already been loaded but we are still in the preamble, load a L<sup>A</sup>T<sub>E</sub>X theme if it exists or load a plain T<sub>E</sub>X theme otherwise.

```
14819 \bool_if:nTF
14820 {
14821 \bool_lazy_or_p:nn
14822 {
14823 \prop_if_in_p:Nn
14824 \g_@@_latex_built_in_themes_prop
14825 { #1 }
14826 }
14827 {
14828 \file_if_exist_p:n
```

```

14829 { markdown theme #3.sty }
14830 }
14831 }
14832 {
14833 \prop_get:NnNTF
14834 \g_@@_latex_loaded_themes_linenos_prop
14835 { #1 }
14836 \l_tmpa_tl
14837 {
14838 \prop_get:NnN
14839 \g_@@_latex_loaded_themes_versions_prop
14840 { #1 }
14841 \l_tmpb_tl
14842 \str_if_eq:nVTF
14843 { #2 }
14844 \l_tmpb_tl
14845 {
14846 \msg_warning:nnnVn
14847 { markdown }
14848 { repeatedly-loaded-latex-theme }
14849 { #1 }
14850 \l_tmpa_tl
14851 { #2 }
14852 }
14853 }
14854 \msg_error:nnnnVV
14855 { markdown }
14856 { different-versions-of-latex-theme }
14857 { #1 }
14858 { #2 }
14859 \l_tmpb_tl
14860 \l_tmpa_tl
14861 }
14862 }
14863 {
14864 \prop_gput:Nnx
14865 \g_@@_latex_loaded_themes_linenos_prop
14866 { #1 }
14867 { \tex_the:D \tex_inputlineno:D } % noqa: W200
14868 \prop_gput:Nnn
14869 \g_@@_latex_loaded_themes_versions_prop
14870 { #1 }
14871 { #2 }

```

Load built-in plain TeX themes from the prop `\g_@@_latex_built_in_themes_prop` and from the filesystem otherwise.

```

14872 \prop_if_in:NnTF

```

```

14873 \g_@@_latex_built_in_themes_prop
14874 { #1 }
14875 {
14876 \msg_info:nnnn
14877 { markdown }
14878 { loading-built-in-latex-theme }
14879 { #1 }
14880 { #2 }
14881 \prop_item:Nn
14882 \g_@@_latex_built_in_themes_prop
14883 { #1 }
14884 }
14885 {
14886 \msg_info:nnnn
14887 { markdown }
14888 { loading-latex-theme }
14889 { #1 }
14890 { #2 }
14891 \RequirePackage
14892 { markdown theme #3 }
14893 }
14894 }
14895 }
14896 {
14897 \@@_plain_tex_load_theme:nnn
14898 { #1 }
14899 { #2 }
14900 { #3 }
14901 }
14902 \fi
14903 \else

```

If the Markdown package has not yet been loaded, postpone the loading until the Markdown package has finished loading.

```

14904 \msg_info:nnnn
14905 { markdown }
14906 { theme-loading-postponed }
14907 { #1 }
14908 { #2 }
14909 \AtEndOfPackage
14910 {
14911 \@@_set_theme:n
14912 { #1 @ #2 }
14913 }
14914 \fi
14915 }
14916 \msg_new:nnn

```

```

14917 { markdown }
14918 { theme-loading-postponed }
14919 {
14920 Postponing~loading~version~#2~of~Markdown~theme~#1~until~
14921 Markdown~package~has~finished~loading
14922 }
14923 \msg_new:nnn
14924 { markdown }
14925 { loading-built-in-latex-theme }
14926 { Loading~version~#2~of~built-in~LaTeX~Markdown~theme~#1 }
14927 \msg_new:nnn
14928 { markdown }
14929 { loading-latex-theme }
14930 { Loading~version~#2~of~LaTeX~Markdown~theme~#1 }
14931 \msg_new:nnn
14932 { markdown }
14933 { repeatedly-loaded-latex-theme }
14934 {
14935 Version~#3~of~LaTeX~Markdown~theme~#1~was~previously~
14936 loaded~on~line~#2,~not~loading~it~again
14937 }
14938 \msg_new:nnn
14939 { markdown }
14940 { different-versions-of-latex-theme }
14941 {
14942 Tried~to~load~version~#2~of~LaTeX~Markdown~theme~#1~
14943 but~version~#3~has~already~been~loaded~on~line~#4
14944 }
14945 \cs_generate_variant:Nn
14946 \msg_new:nnnn
14947 { nnVV }
14948 \tl_set:Nn
14949 \l_tmpa_tl
14950 { Cannot~load~LaTeX~Markdown~theme~#1~after~ }
14951 \tl_put_right:NV
14952 \l_tmpa_tl
14953 \c_backslash_str
14954 \tl_put_right:Nn
14955 \l_tmpa_tl
14956 { begin { document } }
14957 \tl_set:Nn
14958 \l_tmpb_tl
14959 { Load~Markdown~theme~#1~before~ }
14960 \tl_put_right:NV
14961 \l_tmpb_tl
14962 \c_backslash_str
14963 \tl_put_right:Nn

```



```

14964 \l_tmpb_tl
14965 { begin { document } }
14966 \msg_new:nnVV
14967 { markdown }
14968 { latex-theme-after-preamble }
14969 \l_tmpa_tl
14970 \l_tmpb_tl

```

The [witiko/dot](#) and [witiko/graphicx/http](#) L<sup>A</sup>T<sub>E</sub>X themes load the package `graphicx`, see also Section 1.1.3. Then, they load the corresponding plain T<sub>E</sub>X themes.

```

14971 \tl_set:Nn
14972 \l_tmpa_tl
14973 {
14974 \RequirePackage
14975 { graphicx }
14976 \markdownLoadPlainTeXTheme
14977 }
14978 \prop_gput:NnV
14979 \g_@@_latex_built_in_themes_prop
14980 { witiko / dot }
14981 \l_tmpa_tl
14982 \prop_gput:NnV
14983 \g_@@_latex_built_in_themes_prop
14984 { witiko / graphicx / http }
14985 \l_tmpa_tl
14986 \ExplSyntaxOff

```

The [witiko/markdown/defaults](#) L<sup>A</sup>T<sub>E</sub>X theme also loads the corresponding plain T<sub>E</sub>X theme.

```

14987 \markdownLoadPlainTeXTheme

```

Next, the L<sup>A</sup>T<sub>E</sub>X theme overrides some of the plain T<sub>E</sub>X definitions. See Section 3.3.4 for the actual definitions.

### 3.3.3 Options

The supplied package options are processed using the `\markdownSetup` macro.

```

14988 \DeclareOption*{%
14989 \expandafter\markdownSetup\expandafter{\CurrentOption}}%
14990 \ProcessOptions\relax

```

### 3.3.4 Token Renderer Prototypes

The following configuration should be considered placeholder. If the option `plain` has been enabled (see Section 2.2.2.3), none of the definitions will take effect.

```

14991 \markdownIfOption{plain}{\iffalse}{\iftrue}

```

### 3.3.4.1 Lists

If either the `tightLists` or the `fancyLists` Lua option is enabled and the current document class is not beamer, use a package that provides support for tight and fancy lists.

If either the package `paralist` or the package `enumitem` have already been loaded, use them. Otherwise, if the option `experimental` or any test phase has been enabled, use the package `enumitem`. Otherwise, use the package `paralist`.

```
14992 \ExplSyntaxOn
14993 \bool_new:N
14994 \g_@@_tight_or_fancy_lists_bool
14995 \bool_gset_false:N
14996 \g_@@_tight_or_fancy_lists_bool
14997 \@@_if_option:nTF
14998 { tightLists }
14999 {
15000 \bool_gset_true:N
15001 \g_@@_tight_or_fancy_lists_bool
15002 }
15003 {
15004 \@@_if_option:nT
15005 { fancyLists }
15006 {
15007 \bool_gset_true:N
15008 \g_@@_tight_or_fancy_lists_bool
15009 }
15010 }
15011 \bool_new:N
15012 \g_@@_beamer_paralist_or_enumitem_bool
15013 \bool_gset_true:N
15014 \g_@@_beamer_paralist_or_enumitem_bool
15015 \@ifclassloaded
15016 { beamer }
15017 { }
15018 {
15019 \@ifpackageloaded
15020 { paralist }
15021 { }
15022 {
15023 \@ifpackageloaded
15024 { enumitem }
15025 { }
15026 {
15027 \bool_gset_false:N
15028 \g_@@_beamer_paralist_or_enumitem_bool
15029 }
15030 }
```

```

15031 }
15032 \bool_if:nT
15033 {
15034 \g_@@_tight_or_fancy_lists_bool &&
15035 ! \g_@@_beamer_paralist_or_enumitem_bool
15036 }
15037 {
15038 \bool_if:nTF
15039 {
15040 \bool_lazy_or_p:nn
15041 {
15042 \str_if_eq_p:en
15043 { \markdownThemeVersion }
15044 { experimental }
15045 }
15046 {
15047 \bool_lazy_and_p:nn
15048 {
15049 \prop_if_exist_p:N
15050 \g__pdfmanagement_documentproperties_prop
15051 }
15052 {
15053 \bool_lazy_any_p:n
15054 {
15055 {
15056 \prop_if_in_p:Nn
15057 \g__pdfmanagement_documentproperties_prop
15058 { document / testphase / phase-I }
15059 }
15060 {
15061 \prop_if_in_p:Nn
15062 \g__pdfmanagement_documentproperties_prop
15063 { document / testphase / phase-II }
15064 }
15065 {
15066 \prop_if_in_p:Nn
15067 \g__pdfmanagement_documentproperties_prop
15068 { document / testphase / phase-III }
15069 }
15070 {
15071 \prop_if_in_p:Nn
15072 \g__pdfmanagement_documentproperties_prop
15073 { document / testphase / phase-IV }
15074 }
15075 {
15076 \prop_if_in_p:Nn
15077 \g__pdfmanagement_documentproperties_prop

```

```

15078 { document / testphase / phase-V }
15079 }
15080 {
15081 \prop_if_in_p:Nn
15082 \g__pdfmanagement_documentproperties_prop
15083 { document / testphase / phase-VI }
15084 }
15085 }
15086 }
15087 }
15088 }
15089 {
15090 \RequirePackage
15091 { enumitem }
15092 }
15093 {
15094 \RequirePackage
15095 { paralist }
15096 }
15097 }

```

15098 \ExplSyntaxOff

If we loaded the enumitem package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```

15099 \ExplSyntaxOn
15100 \cs_new:Nn
15101 \@@_latex_fancy_list_item_label_number:nn
15102 {
15103 \str_case:nn
15104 { #1 }
15105 {
15106 { Decimal } { #2 }
15107 { LowerRoman } { \int_to_roman:n { #2 } }
15108 { UpperRoman } { \int_to_Roman:n { #2 } }
15109 { LowerAlpha } { \int_to_alph:n { #2 } }
15110 { UpperAlpha } { \int_to_Alph:n { #2 } }
15111 }
15112 }
15113 \cs_new:Nn
15114 \@@_latex_fancy_list_item_label_delimiter:n
15115 {
15116 \str_case:nn
15117 { #1 }
15118 {
15119 { Default } { . }
15120 { OneParen } {) }
15121 { Period } { . }

```

```

15122 }
15123 }
15124 \cs_new:Nn
15125 \@@_latex_fancy_list_item_label:nnn
15126 {
15127 \@@_latex_fancy_list_item_label_number:nn
15128 { #1 }
15129 { #3 }
15130 \@@_latex_fancy_list_item_label_delimiter:n
15131 { #2 }
15132 }
15133 \cs_generate_variant:Nn
15134 \@@_latex_fancy_list_item_label:nnn
15135 { VVn }
15136 \tl_new:N
15137 \l_@@_latex_fancy_list_item_label_number_style_tl
15138 \tl_new:N
15139 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15140 \ifpackageloaded { enumitem } {
15141 \markdownSetup { rendererPrototypes = {

```

First, let's define the tight list item renderer prototypes.

```

15142 ulBeginTight = {
15143 \begin
15144 { itemize }
15145 [noitemsep]
15146 },
15147 ulEndTight = {
15148 \end
15149 { itemize }
15150 },
15151 olBeginTight = {
15152 \begin
15153 { enumerate }
15154 [noitemsep]
15155 },
15156 olEndTight = {
15157 \end
15158 { enumerate }
15159 },
15160 dlBeginTight = {
15161 \begin
15162 { description }
15163 [noitemsep]
15164 },
15165 dlEndTight = {
15166 \end
15167 { description }

```

```
15168 },
```

Second, let's define the fancy list item renderer prototypes.

```
15169 fancyOlBegin = {
15170 \group_begin:
15171 \tl_set:Nn
15172 \l_@@_latex_fancy_list_item_label_number_style_tl
15173 { #1 }
15174 \tl_set:Nn
15175 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15176 { #2 }
15177 \begin
15178 { enumerate }
15179 },
15180 fancyOlBeginTight = {
15181 \group_begin:
15182 \tl_set:Nn
15183 \l_@@_latex_fancy_list_item_label_number_style_tl
15184 { #1 }
15185 \tl_set:Nn
15186 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15187 { #2 }
15188 \begin
15189 { enumerate }
15190 [noitemsep]
15191 },
15192 fancyOlEnd(|Tight) = {
15193 \end { enumerate }
15194 \group_end:
15195 },
15196 fancyOlItemWithNumber = {
15197 \item
15198 [
15199 \@@_latex_fancy_list_item_label:VVn
15200 \l_@@_latex_fancy_list_item_label_number_style_tl
15201 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15202 { #1 }
15203]
15204 },
15205 } }
```

Otherwise, if we loaded the paralist package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```
15206 }
15207 { \ifpackageloaded { paralist } {
15208 \markdownSetup { rendererPrototypes = {
```

Make tight bullet lists a little less compact by adding extra vertical space above and below them.

```

15209 ulBeginTight = {
15210 \group_begin:
15211 \pltopsep=\topsep
15212 \plpartopsep=\partopsep
15213 \begin { compactitem }
15214 },
15215 ulEndTight = {
15216 \end { compactitem }
15217 \group_end:
15218 },
15219 fancyOlBegin = {
15220 \group_begin:
15221 \tl_set:Nn
15222 \l_@@_latex_fancy_list_item_label_number_style_tl
15223 { #1 }
15224 \tl_set:Nn
15225 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15226 { #2 }
15227 \begin { enumerate }
15228 },
15229 fancyOlEnd = {
15230 \end { enumerate }
15231 \group_end:
15232 },

```

Make tight ordered lists a little less compact by adding extra vertical space above and below them.

```

15233 olBeginTight = {
15234 \group_begin:
15235 \plpartopsep=\partopsep
15236 \pltopsep=\topsep
15237 \begin { compactenum }
15238 },
15239 olEndTight = {
15240 \end { compactenum }
15241 \group_end:
15242 },
15243 fancyOlBeginTight = {
15244 \group_begin:
15245 \tl_set:Nn
15246 \l_@@_latex_fancy_list_item_label_number_style_tl
15247 { #1 }
15248 \tl_set:Nn
15249 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15250 { #2 }

```

```

15251 \plpartopsep=\partopsep
15252 \pltopsep=\topsep
15253 \begin { compactenum }
15254 },
15255 fancyOlEndTight = {
15256 \end { compactenum }
15257 \group_end:
15258 },
15259 fancyOlItemWithNumber = {
15260 \item
15261 [
15262 \@@_latex_fancy_list_item_label:VVn
15263 \l_@@_latex_fancy_list_item_label_number_style_tl
15264 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15265 { #1 }
15266]
15267 },

```

Make tight definition lists a little less compact by adding extra vertical space above and below them.

```

15268 dlBeginTight = {
15269 \group_begin:
15270 \plpartopsep=\partopsep
15271 \pltopsep=\topsep
15272 \begin { compactdesc }
15273 },
15274 dlEndTight = {
15275 \end { compactdesc }
15276 \group_end:
15277 }
15278 } }
15279 }
15280 {

```

Otherwise, if we loaded neither the enumitem package nor the paralist package, define the tight and fancy list renderer prototypes to fall back on the corresponding renderers for the non-tight lists.

```

15281 \markdownSetup
15282 {
15283 rendererPrototypes = {
15284 ulBeginTight = \markdownRendererUlBegin,
15285 ulEndTight = \markdownRendererUlEnd,
15286 fancyOlBegin = \markdownRendererOlBegin,
15287 fancyOlEnd = \markdownRendererOlEnd,
15288 olBeginTight = \markdownRendererOlBegin,
15289 olEndTight = \markdownRendererOlEnd,
15290 fancyOlBeginTight = \markdownRendererOlBegin,
15291 fancyOlEndTight = \markdownRendererOlEnd,

```



```

15292 dlBeginTight = \markdownRendererDlBegin,
15293 dlEndTight = \markdownRendererDlEnd,
15294 },
15295 }
15296 } }
15297 \ExplSyntaxOff
15298 \RequirePackage{amsmath}

```

Unless the unicode-math package has been loaded, load the amssymb package with symbols to be used for tickboxes.

```

15299 \@ifpackageloaded{unicode-math}{
15300 \markdownSetup{rendererPrototypes={
15301 untickedBox = {\mdlgwhtsquare$},
15302 }}
15303 }{
15304 \RequirePackage{amssymb}
15305 \markdownSetup{rendererPrototypes={
15306 untickedBox = {\square$},
15307 }}
15308 }
15309 \RequirePackage{csvsimple}
15310 \RequirePackage{fancyvrb}
15311 \RequirePackage{graphicx}
15312 \markdownSetup{rendererPrototypes={
15313 hardLineBreak = {\},
15314 leftBrace = {\textbraceleft},
15315 rightBrace = {\textbraceright},
15316 dollarSign = {\textdollar},
15317 underscore = {\textunderscore},
15318 circumflex = {\textasciicircum},
15319 backslash = {\textbackslash},
15320 tilde = {\textasciitilde},
15321 pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by  $\text{\TeX}$  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,<sup>34</sup> we can reliably detect math mode inside the renderer.

Here, we will redefine the code span renderer prototype to typeset upright text in math formulae and typewriter text outside math formulae.

```

15322 codeSpan = {%
15323 \ifmmode
15324 \text{#1}%
15325 \else

```

---

<sup>34</sup>This property may actually be undecidable. Suppose a span of text is a part of a macro definition. Then, whether the span of text is part of a math formula or not depends on where the macro is later used, which may easily be *both* inside and outside a math formula.

```

15326 \texttt{#1}%
15327 \fi
15328 }}}
```

### 3.3.4.2 Content Blocks

In content block renderer prototypes, display the content as a table using the package `csvsimple` when the raw attribute is `csv`, display the content using the default templates of the package `luaxml` when the raw attribute is `html`, execute the content with TeX when the raw attribute is `tex`, and display the content as markdown otherwise.

```

15329 \ExplSyntaxOn
15330 \markdownSetup{
15331 rendererPrototypes = {
15332 contentBlock = {
15333 \str_case:nnF
15334 { #1 }
15335 {
15336 { csv }
15337 {
15338 \begin { table }
15339 \begin { center }
15340 \csvautotabular { #3 }
15341 \end { center }
15342 \tl_if_empty:nF
15343 { #4 }
15344 { \caption { #4 } }
15345 \end { table }
15346 }
15347 { html }
15348 {
```

If we are using `TeX4ht`<sup>35</sup>, we will pass HTML elements to the output HTML document unchanged.

```

15349 \cs_if_exist:NTF
15350 \HCode
15351 {
15352 \if_mode_vertical:
15353 \IgnorePar
15354 \fi:
15355 \EndP
15356 \special
15357 { t4ht* < #3 }
15358 \par
15359 \ShowPar
```

---

<sup>35</sup>See <https://tug.org/tex4ht/>.

```

15360 }
15361 {
15362 \@@_luaxml_print_html:n
15363 { #3 }
15364 }
15365 }
15366 { tex }
15367 {
15368 \markdownEscape
15369 { #3 }
15370 }
15371 }
15372 {
15373 \markdownInput
15374 { #3 }
15375 }
15376 },
15377 },
15378 }
15379 \ExplSyntaxOff
15380 \markdownSetup{rendererPrototypes={
15381 ulBegin = {\begin{itemize}},
15382 ulEnd = {\end{itemize}},
15383 olBegin = {\begin{enumerate}},
15384 olItem = {\item{}},
15385 olItemWithNumber = {\item[#1.]},
15386 olEnd = {\end{enumerate}},
15387 dlBegin = {\begin{description}},
15388 dlItem = {\item[#1]},
15389 dlEnd = {\end{description}},
15390 emphasis = {\emph{#1}},
15391 tickedTextBox = {\\boxtimes},
15392 halfTickedTextBox = {\\boxdot}}}

```

If HTML identifiers appear after a heading, we make them produce `\label` macros.

```

15393 \ExplSyntaxOn
15394 \seq_new:N
15395 \g_@@_header_identifiers_seq
15396 \markdownSetup
15397 {
15398 rendererPrototypes = {
15399 headerAttributeContextBegin = {
15400 \markdownSetup
15401 {
15402 rendererPrototypes = {
15403 attributeIdentifier = {
15404 \seq_gput_right:Nn
15405 \g_@@_header_identifiers_seq

```

```

15406 { ##1 }
15407 },
15408 },
15409 }
15410 },
15411 headerAttributeContextEnd = {
15412 \seq_map_inline:Nn
15413 \g_@@_header_identifiers_seq
15414 { \label { ##1 } }
15415 \seq_gclear:N
15416 \g_@@_header_identifiers_seq
15417 },
15418 },
15419 }

```

If the `unnumbered` HTML class (or the `{-}` shorthand) appears after a heading the heading and all its subheadings will be unnumbered.

```

15420 \bool_new:N
15421 \l_@@_header_unnumbered_bool
15422 \markdownSetup
15423 {
15424 rendererPrototypes = {
15425 headerAttributeContextBegin += {
15426 \markdownSetup
15427 {
15428 rendererPrototypes = {
15429 attributeClassName = {
15430 \bool_if:nT
15431 {
15432 \str_if_eq_p:nn
15433 { ##1 }
15434 { unnumbered } &&
15435 ! \l_@@_header_unnumbered_bool
15436 }
15437 {
15438 \group_begin:
15439 \bool_set_true:N
15440 \l_@@_header_unnumbered_bool
15441 \c@secnumdepth = 0
15442 \markdownSetup
15443 {
15444 rendererPrototypes = {
15445 sectionBegin = {
15446 \group_begin:
15447 },
15448 sectionEnd = {
15449 \group_end:

```

```

15450 },
15451 },
15452 }
15453 }
15454 },
15455 },
15456 }
15457 },
15458 },
15459 }
15460 \ExplSyntaxOff
15461 \markdownSetup{rendererPrototypes={
15462 superscript = {#1},
15463 subscript = {\textsubscript{#1}},
15464 blockQuoteBegin = {\begin{quotation}},
15465 blockQuoteEnd = {\end{quotation}},
15466 inputVerbatim = {\VerbatimInput{#1}},
15467 thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
15468 note = {\footnote{#1}}}}

```

### 3.3.4.3 Fenced Code

When no `infostring` has been specified, default to the indented code block renderer.

```

15469 \RequirePackage{ltxcmds}
15470 \ExplSyntaxOn
15471 \cs_gset_protected:Npn
15472 \markdownRendererInputFencedCodePrototype#1#2#3
15473 {
15474 \tl_if_empty:nTF
15475 { #2 }
15476 { \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.

```

15477 {
15478 \regex_extract_once:nnN
15479 { \w* }
15480 { #2 }
15481 \l_tmpa_seq
15482 \seq_pop_left:NN
15483 \l_tmpa_seq
15484 \l_tmpa_tl

```

When the `minted` package is loaded, use it for syntax highlighting.

```

15485 \ltx@ifpackageloaded
15486 { minted }
15487 {
15488 \catcode`\%=14\relax

```

```

15489 \catcode`\#=6\relax
15490 \exp_args:NV
15491 \inputminted
15492 \l_tmpa_tl
15493 { #1 }
15494 \catcode`\%=12\relax
15495 \catcode`\#=12\relax
15496 }
15497 {

```

When the listings package is loaded, use it for syntax highlighting.

```

15498 \ltx@ifpackageloaded
15499 { listings }
15500 { \lstinputlisting [language = \l_tmpa_tl] { #1 } }

```

When neither the listings package nor the minted package is loaded, act as though no infostring were given.

```

15501 { \markdownRendererInputFencedCode { #1 } { } { } }
15502 }
15503 }
15504 }
15505 \ExplSyntaxOff

```

Support the nesting of strong emphasis.

```

15506 \ExplSyntaxOn
15507 \def\markdownLATEXStrongEmphasis#1{
15508 \str_if_in:NnTF
15509 \f@series
15510 { b }
15511 { \textnormal{#1} }
15512 { \textbf{#1} }
15513 }
15514 \ExplSyntaxOff
15515 \markdownSetup{rendererPrototypes={strongEmphasis={%
15516 \protect\markdownLATEXStrongEmphasis{#1}}}}

```

Support L<sup>A</sup>T<sub>E</sub>X document classes that do not provide chapters.

```

15517 \@ifundefined{chapter}{%
15518 \markdownSetup{rendererPrototypes = {
15519 headingOne = {\section{#1}},
15520 headingTwo = {\subsection{#1}},
15521 headingThree = {\subsubsection{#1}},
15522 headingFour = {\paragraph{#1}},
15523 headingFive = {\subparagraph{#1}}}}
15524 }{%
15525 \markdownSetup{rendererPrototypes = {
15526 headingOne = {\chapter{#1}},
15527 headingTwo = {\section{#1}},
15528 headingThree = {\subsection{#1}},

```

```

15529 headingFour = {\subsubsection{#1}},
15530 headingFive = {\paragraph{#1}},
15531 headingSix = {\subparagraph{#1}}}}
15532 }%

```

#### 3.3.4.4 Tickboxes

If the `taskLists` option is enabled, we will hide bullets in unordered list items with tickboxes.

```

15533 \markdownSetup{
15534 rendererPrototypes = {
15535 ulItem = {%
15536 \futurelet\markdownLaTeXCheckbox\markdownLaTeXUItem
15537 },
15538 },
15539 }
15540 \def\markdownLaTeXUItem{%
15541 \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
15542 \item[\markdownLaTeXCheckbox]%
15543 \expandafter\@gobble
15544 \else
15545 \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
15546 \item[\markdownLaTeXCheckbox]%
15547 \expandafter\expandafter\expandafter\@gobble
15548 \else
15549 \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
15550 \item[\markdownLaTeXCheckbox]%
15551 \expandafter\expandafter\expandafter\expandafter
15552 \expandafter\expandafter\expandafter\@gobble
15553 \else
15554 \item{}%
15555 \fi
15556 \fi
15557 \fi
15558 }

```

#### 3.3.4.5 HTML elements

If the `html` option is enabled and we are using  $\text{T}_{\text{E}}\text{X}_{4\text{ht}}$ <sup>36</sup>, we will pass HTML elements to the output HTML document unchanged.

```

15559 \@ifundefined{HCode}{}{
15560 \markdownSetup{
15561 rendererPrototypes = {
15562 inlineHtmlTag = {%
15563 \ifvmode
15564 \IgnorePar

```

---

<sup>36</sup>See <https://tug.org/tex4ht/>.

```

15565 \EndP
15566 \fi
15567 \HCode{#1}%
15568 },
15569 inputBlockHtmlElement = {%
15570 \ifvmode
15571 \IgnorePar
15572 \fi
15573 \EndP
15574 \special{t4ht*<#1}%
15575 \par
15576 \ShowPar
15577 },
15578 },
15579 }
15580 }

```

### 3.3.4.6 Citations

Here is a basic implementation for citations that uses the L<sup>A</sup>T<sub>E</sub>X `\cite` macro. There are also implementations that use the natbib `\citep`, and `\citet` macros, and the BibL<sup>A</sup>T<sub>E</sub>X `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```

15581 \newcount\markdownLaTeXCitationsCounter
15582
15583 % Basic implementation
15584 \long\def@gobblethree#1#2#3{%
15585 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
15586 \advance\markdownLaTeXCitationsCounter by 1\relax
15587 \ifx\relax#4\relax
15588 \ifx\relax#5\relax
15589 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15590 \relax
15591 \cite{#1#2#6}% No prenotes/postnotes, just accumulate cites
15592 \expandafter\expandafter\expandafter
15593 \expandafter\expandafter\expandafter\expandafter
15594 \@gobblethree
15595 \fi
15596 \else% Before a postnote (#5), dump the accumulator
15597 \ifx\relax#1\relax\else
15598 \cite{#1}%
15599 \fi
15600 \cite[#5]{#6}%
15601 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15602 \relax
15603 \else
15604 \expandafter\expandafter\expandafter

```



```

15605 \expandafter\expandafter\expandafter\expandafter
15606 \expandafter\expandafter\expandafter
15607 \expandafter\expandafter\expandafter\expandafter
15608 \markdownLaTeXBasicCitations
15609 \fi
15610 \expandafter\expandafter\expandafter
15611 \expandafter\expandafter\expandafter\expandafter{%
15612 \expandafter\expandafter\expandafter
15613 \expandafter\expandafter\expandafter\expandafter}%
15614 \expandafter\expandafter\expandafter
15615 \expandafter\expandafter\expandafter\expandafter{%
15616 \expandafter\expandafter\expandafter
15617 \expandafter\expandafter\expandafter\expandafter}%
15618 \expandafter\expandafter\expandafter
15619 \@gobblethree
15620 \fi
15621 \else% Before a prenote (#4), dump the accumulator
15622 \ifx\relax#1\relax\else
15623 \cite{#1}%
15624 \fi
15625 \ifnum\markdownLaTeXCitationsCounter>1\relax
15626 \space % Insert a space before the prenote in later citations
15627 \fi
15628 #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
15629 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15630 \relax
15631 \else
15632 \expandafter\expandafter\expandafter
15633 \expandafter\expandafter\expandafter\expandafter
15634 \markdownLaTeXBasicCitations
15635 \fi
15636 \expandafter\expandafter\expandafter{%
15637 \expandafter\expandafter\expandafter}%
15638 \expandafter\expandafter\expandafter{%
15639 \expandafter\expandafter\expandafter}%
15640 \expandafter
15641 \@gobblethree
15642 \fi\markdownLaTeXBasicCitations{#1#2#6},}
15643 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
15644
15645 % Natbib implementation
15646 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
15647 \advance\markdownLaTeXCitationsCounter by 1\relax
15648 \ifx\relax#3\relax
15649 \ifx\relax#4\relax
15650 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15651 \relax

```

```

15652 \citep{#1,#5}% No prenotes/postnotes, just accumulate cites
15653 \expandafter\expandafter\expandafter
15654 \expandafter\expandafter\expandafter\expandafter
15655 \@gobbletwo
15656 \fi
15657 \else% Before a postnote (#4), dump the accumulator
15658 \ifx\relax#1\relax\else
15659 \citep{#1}%
15660 \fi
15661 \citep[] [#4]{#5}%
15662 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15663 \relax
15664 \else
15665 \expandafter\expandafter\expandafter
15666 \expandafter\expandafter\expandafter\expandafter
15667 \expandafter\expandafter\expandafter
15668 \expandafter\expandafter\expandafter\expandafter
15669 \markdownLaTeXNatbibCitations
15670 \fi
15671 \expandafter\expandafter\expandafter
15672 \expandafter\expandafter\expandafter\expandafter{%
15673 \expandafter\expandafter\expandafter
15674 \expandafter\expandafter\expandafter\expandafter}%
15675 \expandafter\expandafter\expandafter
15676 \@gobbletwo
15677 \fi
15678 \else% Before a prenote (#3), dump the accumulator
15679 \ifx\relax#1\relax\relax\else
15680 \citep{#1}%
15681 \fi
15682 \citep[#3] [#4]{#5}%
15683 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15684 \relax
15685 \else
15686 \expandafter\expandafter\expandafter
15687 \expandafter\expandafter\expandafter\expandafter
15688 \markdownLaTeXNatbibCitations
15689 \fi
15690 \expandafter\expandafter\expandafter{%
15691 \expandafter\expandafter\expandafter}%
15692 \expandafter
15693 \@gobbletwo
15694 \fi\markdownLaTeXNatbibCitations{#1,#5}}
15695 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
15696 \advance\markdownLaTeXCitationsCounter by 1\relax
15697 \ifx\relax#3\relax
15698 \ifx\relax#4\relax

```

```

15699 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15700 \relax
15701 \citet{#1,#5}% No prenotes/postnotes, just accumulate cites
15702 \expandafter\expandafter\expandafter
15703 \expandafter\expandafter\expandafter\expandafter
15704 \@gobbletwo
15705 \fi
15706 \else% After a prenote or a postnote, dump the accumulator
15707 \ifx\relax#1\relax\else
15708 \citet{#1}%
15709 \fi
15710 , \citet[#3][#4]{#5}%
15711 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
15712 \relax
15713 ,
15714 \else
15715 \ifnum
15716 \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
15717 \relax
15718 ,
15719 \fi
15720 \fi
15721 \expandafter\expandafter\expandafter
15722 \expandafter\expandafter\expandafter\expandafter
15723 \markdownLaTeXNatbibTextCitations
15724 \expandafter\expandafter\expandafter
15725 \expandafter\expandafter\expandafter\expandafter{%
15726 \expandafter\expandafter\expandafter
15727 \expandafter\expandafter\expandafter\expandafter}%
15728 \expandafter\expandafter\expandafter
15729 \@gobbletwo
15730 \fi
15731 \else% After a prenote or a postnote, dump the accumulator
15732 \ifx\relax#1\relax\relax\else
15733 \citet{#1}%
15734 \fi
15735 , \citet[#3][#4]{#5}%
15736 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
15737 \relax
15738 ,
15739 \else
15740 \ifnum
15741 \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
15742 \relax
15743 ,
15744 \fi
15745 \fi

```

```

15746 \expandafter\expandafter\expandafter
15747 \markdownLaTeXNatbibTextCitations
15748 \expandafter\expandafter\expandafter{%
15749 \expandafter\expandafter\expandafter}%
15750 \expandafter
15751 \@gobbletwo
15752 \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
15753
15754 % BibLaTeX implementation
15755 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
15756 \advance\markdownLaTeXCitationsCounter by 1\relax
15757 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15758 \relax
15759 \autocites#1[#3][#4]{#5}%
15760 \expandafter\@gobbletwo
15761 \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
15762 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
15763 \advance\markdownLaTeXCitationsCounter by 1\relax
15764 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15765 \relax
15766 \textcites#1[#3][#4]{#5}%
15767 \expandafter\@gobbletwo
15768 \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}}
15769
15770 \markdownSetup{rendererPrototypes = {
15771 cite = {%
15772 \markdownLaTeXCitationsCounter=1%
15773 \def\markdownLaTeXCitationsTotal{#1}%
15774 \@ifundefined{autocites}{%
15775 \@ifundefined{citep}{%
15776 \expandafter\expandafter\expandafter
15777 \markdownLaTeXBasicCitations
15778 \expandafter\expandafter\expandafter{%
15779 \expandafter\expandafter\expandafter}%
15780 \expandafter\expandafter\expandafter{%
15781 \expandafter\expandafter\expandafter}%
15782 }{%
15783 \expandafter\expandafter\expandafter
15784 \markdownLaTeXNatbibCitations
15785 \expandafter\expandafter\expandafter{%
15786 \expandafter\expandafter\expandafter}%
15787 }%
15788 }{%
15789 \expandafter\expandafter\expandafter
15790 \markdownLaTeXBibLaTeXCitations
15791 \expandafter{\expandafter}%
15792 }},

```

```

15793 textCite = {%
15794 \markdownLaTeXCitationsCounter=1%
15795 \def\markdownLaTeXCitationsTotal{#1}%
15796 \@ifundefined{autocites}{%
15797 \@ifundefined{citep}{%
15798 \expandafter\expandafter\expandafter
15799 \markdownLaTeXBasicTextCitations
15800 \expandafter\expandafter\expandafter{%
15801 \expandafter\expandafter\expandafter}%
15802 \expandafter\expandafter\expandafter{%
15803 \expandafter\expandafter\expandafter}%
15804 }{%
15805 \expandafter\expandafter\expandafter
15806 \markdownLaTeXNatbibTextCitations
15807 \expandafter\expandafter\expandafter{%
15808 \expandafter\expandafter\expandafter}%
15809 }%
15810 }{%
15811 \expandafter\expandafter\expandafter
15812 \markdownLaTeXBibLaTeXTextCitations
15813 \expandafter{\expandafter}%
15814 }}}

```

### 3.3.4.7 Links

Here is an implementation for hypertext links and relative references.

```

15815 \RequirePackage{url}
15816 \RequirePackage{expl3}
15817 \ExplSyntaxOn
15818 \cs_gset_protected:Npn
15819 \markdownRendererLinkPrototype
15820 #1#2#3#4
15821 {
15822 \tl_set:Nn \l_tmpa_tl { #1 }
15823 \tl_set:Nn \l_tmpb_tl { #2 }
15824 \bool_set:Nn
15825 \l_tmpa_bool
15826 {
15827 \tl_if_eq_p:NN
15828 \l_tmpa_tl
15829 \l_tmpb_tl
15830 }
15831 \tl_set:Nn \l_tmpa_tl { #4 }
15832 \bool_set:Nn
15833 \l_tmpb_bool
15834 {
15835 \tl_if_empty_p:N

```

```

15836 \l_tmpa_tl
15837 }

```

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.

```

15838 \bool_if:nTF
15839 {
15840 \l_tmpa_bool && \l_tmpb_bool
15841 }
15842 {
15843 \markdownLaTeXRendererAutolink { #2 } { #3 }
15844 }
15845 {
15846 \markdownLaTeXRendererDirectOrIndirectLink
15847 { #1 } { #2 } { #3 } { #4 }
15848 }
15849 }
15850 \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.

```

15851 \tl_set:Nn
15852 \l_tmpa_tl
15853 { #2 }
15854 \tl_trim_spaces:N
15855 \l_tmpa_tl
15856 \tl_set:Nx
15857 \l_tmpb_tl
15858 {
15859 \tl_range:Nnn
15860 \l_tmpa_tl
15861 { 1 }
15862 { 1 }
15863 }
15864 \str_if_eq:NNTF
15865 \l_tmpb_tl
15866 \c_hash_str
15867 {
15868 \tl_set:Nx
15869 \l_tmpb_tl
15870 {
15871 \tl_range:Nnn
15872 \l_tmpa_tl
15873 { 2 }
15874 { -1 }
15875 }
15876 \exp_args:NV

```

```

15877 \ref
15878 \l_tmpb_tl
15879 }
15880 {
15881 \url { #2 }
15882 }
15883 }
15884 \ExplSyntaxOff
15885 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
15886 #1\footnote{\ifx\empty#4\empty\else#4: \fi\url{#3}}

```

### 3.3.4.8 Tables

Here is a basic implementation of tables. If the booktabs package is loaded, then it is used to produce horizontal lines.

```

15887 \newcount\markdownLaTeXRowCount
15888 \newcount\markdownLaTeXRowTotal
15889 \newcount\markdownLaTeXColumnCounter
15890 \newcount\markdownLaTeXColumnTotal
15891 \newtoks\markdownLaTeXTable
15892 \newtoks\markdownLaTeXTableAlignment
15893 \newtoks\markdownLaTeXTableEnd
15894 \AtBeginDocument{%
15895 \@ifpackageloaded{booktabs}{%
15896 \def\markdownLaTeXTopRule{\toprule}%
15897 \def\markdownLaTeXMidRule{\midrule}%
15898 \def\markdownLaTeXBottomRule{\bottomrule}%
15899 }{%
15900 \def\markdownLaTeXTopRule{\hline}%
15901 \def\markdownLaTeXMidRule{\hline}%
15902 \def\markdownLaTeXBottomRule{\hline}%
15903 }%
15904 }
15905 \markdownSetup{rendererPrototypes={
15906 table = {%
15907 \markdownLaTeXTable={}%
15908 \markdownLaTeXTableAlignment={}%
15909 \markdownLaTeXTableEnd={%
15910 \markdownLaTeXBottomRule
15911 \end{tabular}}}%
15912 \ifx\empty#1\empty\else
15913 \addto@hook\markdownLaTeXTable{%
15914 \begin{table}
15915 \centering}%
15916 \addto@hook\markdownLaTeXTableEnd{%
15917 \caption{#1}}}%
15918 \fi

```

```

15919 }
15920 }}

```

If the `tableAttributes` option is enabled, we will register any identifiers, so that they can be used as L<sup>A</sup>T<sub>E</sub>X labels for referencing tables.

```

15921 \ExplSyntaxOn
15922 \seq_new:N
15923 \l_@@_table_identifiers_seq
15924 \markdownSetup {
15925 rendererPrototypes = {
15926 table += {
15927 \seq_map_inline:Nn
15928 \l_@@_table_identifiers_seq
15929 {
15930 \addto@hook
15931 \markdownLaTeXTableEnd
15932 { \label { ##1 } }
15933 }
15934 },
15935 }
15936 }
15937 \markdownSetup {
15938 rendererPrototypes = {
15939 tableAttributeContextBegin = {
15940 \group_begin:
15941 \markdownSetup {
15942 rendererPrototypes = {
15943 attributeIdentifier = {
15944 \seq_put_right:Nn
15945 \l_@@_table_identifiers_seq
15946 { ##1 }
15947 },
15948 },
15949 }
15950 },
15951 tableAttributeContextEnd = {
15952 \group_end:
15953 },
15954 },
15955 }
15956 \ExplSyntaxOff
15957 \markdownSetup{rendererPrototypes={
15958 table += {%
15959 \ifx\empty#1\empty\else
15960 \addto@hook\markdownLaTeXTableEnd{%
15961 \end{table}}}%
15962 \fi

```



```

15963 \addto@hook\markdownLaTeXTable{\begin{tabular}}%
15964 \markdownLaTeXRowCount=0%
15965 \markdownLaTeXRowTotal=#2%
15966 \markdownLaTeXColumnTotal=#3%
15967 \markdownLaTeXRenderTableRow
15968 }
15969 }}
15970 \def\markdownLaTeXRenderTableRow#1{%
15971 \markdownLaTeXColumnCounter=0%
15972 \ifnum\markdownLaTeXRowCount=0\relax
15973 \markdownLaTeXReadAlignments#1%
15974 \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
15975 \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
15976 \the\markdownLaTeXTableAlignment}}%
15977 \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
15978 \else
15979 \markdownLaTeXRenderTableCell#1%
15980 \fi
15981 \ifnum\markdownLaTeXRowCount=1\relax
15982 \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
15983 \fi
15984 \advance\markdownLaTeXRowCount by 1\relax
15985 \ifnum\markdownLaTeXRowCount>\markdownLaTeXRowTotal\relax
15986 \the\markdownLaTeXTable
15987 \the\markdownLaTeXTableEnd
15988 \expandafter\@gobble
15989 \fi\markdownLaTeXRenderTableRow}
15990 \def\markdownLaTeXReadAlignments#1{%
15991 \advance\markdownLaTeXColumnCounter by 1\relax
15992 \if#1d%
15993 \addto@hook\markdownLaTeXTableAlignment{1}%
15994 \else
15995 \addto@hook\markdownLaTeXTableAlignment{#1}%
15996 \fi
15997 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
15998 \expandafter\@gobble
15999 \fi\markdownLaTeXReadAlignments}
16000 \def\markdownLaTeXRenderTableCell#1{%
16001 \advance\markdownLaTeXColumnCounter by 1\relax
16002 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
16003 \addto@hook\markdownLaTeXTable{#1&}%
16004 \else
16005 \addto@hook\markdownLaTeXTable{#1\\}%
16006 \expandafter\@gobble
16007 \fi\markdownLaTeXRenderTableCell}

```

### 3.3.4.9 Line Blocks

Here is a basic implementation of line blocks. If the `verse` package is loaded, then it is used to produce the verses.

```

16008
16009 \markdownIfOption{lineBlocks}{%
16010 \RequirePackage{verse}
16011 \markdownSetup{rendererPrototypes={
16012 lineBlockBegin = {%
16013 \begingroup
16014 \def\markdownRendererHardLineBreak{\\}%
16015 \begin{verse}%
16016 },
16017 lineBlockEnd = {%
16018 \end{verse}%
16019 \endgroup
16020 },
16021 }}
16022 }{}
16023

```

#### 3.3.4.10 YAML Metadata

The default setup of YAML metadata will invoke the `\title`, `\author`, and `\date` macros when scalar values for keys that correspond to the `title`, `author`, and `date` relative wildcards are encountered, respectively.

```

16024 \ExplSyntaxOn
16025 \keys_define:nn
16026 { markdown / jekyllData }
16027 {
16028 author .code:n = {
16029 \author
16030 { #1 }
16031 },
16032 date .code:n = {
16033 \date
16034 { #1 }
16035 },
16036 title .code:n = {
16037 \title
16038 { #1 }

```

To complement the default setup of our key-values, we will use the `\maketitle` macro to typeset the title page of a document at the end of YAML metadata. If we are in the preamble, we will wait macro until after the beginning of the document. Otherwise, we will use the `\maketitle` macro straight away.

```

16039 \AddToHook
16040 { begindocument / end }
16041 { \maketitle }

```

```

16042 },
16043 }

```

#### 3.3.4.11 Marked Text

If the `mark` option is enabled, we will load either the `soul` package or the `lua-ul` package and use it to implement marked text.

```

16044 \@@_if_option:nT
16045 { mark }
16046 {
16047 \sys_if_engine luatex:TF
16048 {
16049 \RequirePackage
16050 { luacolor }
16051 \RequirePackage
16052 { lua-ul }
16053 \markdownSetup
16054 {
16055 rendererPrototypes = {
16056 mark = {
16057 \highLight
16058 { #1 }
16059 },
16060 }
16061 }
16062 }
16063 {
16064 \RequirePackage
16065 { xcolor }
16066 \RequirePackage
16067 { soul }
16068 \markdownSetup
16069 {
16070 rendererPrototypes = {
16071 mark = {
16072 \hl
16073 { #1 }
16074 },
16075 }
16076 }
16077 }
16078 }

```

#### 3.3.4.12 Strike-Through

If the `strikeThrough` option is enabled, we will load either the `soul` package or the `lua-ul` package and use it to implement strike-throughs.

```

16079 \@@_if_option:nT
16080 { strikeThrough }
16081 {
16082 \sys_if_engine luatex:TF
16083 {
16084 \RequirePackage
16085 { lua-ul }
16086 \markdownSetup
16087 {
16088 rendererPrototypes = {
16089 strikeThrough = {
16090 \strikeThrough
16091 { #1 }
16092 },
16093 }
16094 }
16095 }
16096 {
16097 \RequirePackage
16098 { soul }
16099 \markdownSetup
16100 {
16101 rendererPrototypes = {
16102 strikeThrough = {
16103 \st
16104 { #1 }
16105 },
16106 }
16107 }
16108 }
16109 }

```

### 3.3.4.13 Images and their attributes

We define images to be rendered as floating figures using the command `\includegraphics`, where the image label is the alt text and the image title is the caption of the figure.

If the `linkAttributes` option is enabled, we will make attributes in the form  $\langle key \rangle = \langle value \rangle$  set the corresponding keys of the `graphicx` package to the corresponding values and we will register any identifiers, so that they can be used as  $\text{\LaTeX}$  labels for referencing figures.

```

16110 \seq_new:N
16111 \l_@@_image_identifiers_seq
16112 \markdownSetup {
16113 rendererPrototypes = {
16114 image = {
16115 \tl_if_empty:nTF

```

```

16116 { #4 }
16117 {
16118 \begin { center }
16119 \includegraphics
16120 [alt = { #1 }]
16121 { #3 }
16122 \end { center }
16123 }
16124 {
16125 \begin { figure }
16126 \begin { center }
16127 \includegraphics
16128 [alt = { #1 }]
16129 { #3 }
16130 \caption { #4 }
16131 \seq_map_inline:Nn
16132 \l_@@_image_identifiers_seq
16133 { \label { ##1 } }
16134 \end { center }
16135 \end { figure }
16136 }
16137 },
16138 }
16139 }
16140 \@@_if_option:nT
16141 { linkAttributes }
16142 {
16143 \RequirePackage { graphicx }
16144 }
16145 \markdownSetup {
16146 rendererPrototypes = {
16147 imageAttributeContextBegin = {
16148 \group_begin:
16149 \markdownSetup {
16150 rendererPrototypes = {
16151 attributeIdentifier = {
16152 \seq_put_right:Nn
16153 \l_@@_image_identifiers_seq
16154 { ##1 }
16155 },
16156 attributeKeyValue = {
16157 \setkeys
16158 { Gin }
16159 { { ##1 } = { ##2 } }
16160 },
16161 },
16162 }

```

```

16163 },
16164 imageAttributeContextEnd = {
16165 \group_end:
16166 },
16167 },
16168 }
16169 \ExplSyntaxOff

```

### 3.3.4.14 Raw Attributes

In the raw block and inline raw span renderer prototypes, display the content using the default templates of the package `luaxml` when the raw attribute is `html` and default to the plain TeX renderer prototypes otherwise, translating raw attribute `latex` to `tex`.

```

16170 \ExplSyntaxOn
16171 \cs_new:Nn
16172 \@@_luaxml_print_html:n
16173 {
16174 \luabridge_now:n
16175 {
16176 local~input_file = assert(io.open(" #1 ", "r"))
16177 local~input = assert(input_file:read("*a"))
16178 assert(input_file:close())
16179 input = "<body>" .. input .. "</body>"
16180 local~dom = require("luaxml-domobject").html_parse(input)
16181 local~output = require("luaxml-htmltemplates"):process_dom(dom)
16182 print(output)
16183 }
16184 }
16185 \cs_gset_protected:Npn
16186 \markdownRendererInputRawInlinePrototype#1#2
16187 {
16188 \str_case:nnF
16189 { #2 }
16190 {
16191 { latex }
16192 {
16193 \@@_plain_tex_default_input_raw_inline:nn
16194 { #1 }
16195 { tex }
16196 }
16197 { html }
16198 {

```

If we are using  $\text{\TeX}$ 4ht<sup>37</sup>, we will pass HTML elements to the output HTML document unchanged.

---

<sup>37</sup>See <https://tug.org/tex4ht/>.

```

16199 \cs_if_exist:NTF
16200 \HCode
16201 {
16202 \if_mode_vertical:
16203 \IgnorePar
16204 \EndP
16205 \fi:
16206 \special
16207 { t4ht* < #1 }
16208 }
16209 {
16210 \@@_luaxml_print_html:n
16211 { #1 }
16212 }
16213 }
16214 }
16215 {
16216 \@@_plain_tex_default_input_raw_inline:nn
16217 { #1 }
16218 { #2 }
16219 }
16220 }
16221 \cs_gset_protected:Npn
16222 \markdownRendererInputRawBlockPrototype#1#2
16223 {
16224 \str_case:nnF
16225 { #2 }
16226 {
16227 { latex }
16228 {
16229 \@@_plain_tex_default_input_raw_block:nn
16230 { #1 }
16231 { tex }
16232 }
16233 { html }
16234 {

```

If we are using  $\text{\TeX}4\text{ht}$ <sup>38</sup>, we will pass HTML elements to the output HTML document unchanged.

```

16235 \cs_if_exist:NTF
16236 \HCode
16237 {
16238 \if_mode_vertical:
16239 \IgnorePar
16240 \fi:
16241 \EndP

```

---

<sup>38</sup>See <https://tug.org/tex4ht/>.

```

16242 \special
16243 { t4ht* < #1 }
16244 \par
16245 \ShowPar
16246 }
16247 {
16248 \@@_luaxml_print_html:n
16249 { #1 }
16250 }
16251 }
16252 }
16253 {
16254 \@@_plain_tex_default_input_raw_block:nn
16255 { #1 }
16256 { #2 }
16257 }
16258 }

```

### 3.3.4.15 Bracketed spans

If the `bracketedSpans` option is enabled, we will register any identifiers, so that they can be used as  $\text{\LaTeX}$  labels for referencing the last  $\text{\LaTeX}$  counter that has been incremented in e.g. ordered lists.

```

16259 \seq_new:N
16260 \l_@@_bracketed_span_identifiers_seq
16261 \markdownSetup {
16262 rendererPrototypes = {
16263 bracketedSpanAttributeContextBegin = {
16264 \group_begin:
16265 \markdownSetup {
16266 rendererPrototypes = {
16267 attributeIdentifier = {
16268 \seq_put_right:Nn
16269 \l_@@_bracketed_span_identifiers_seq
16270 { ##1 }
16271 },
16272 },
16273 }
16274 },
16275 bracketedSpanAttributeContextEnd = {
16276 \seq_map_inline:Nn
16277 \l_@@_bracketed_span_identifiers_seq
16278 { \label { ##1 } }
16279 \group_end:
16280 },
16281 },
16282 }

```



```

16283 \ExplSyntaxOff
16284 \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.

```

16285 \newcommand\markdownMakeOther{%
16286 \count0=128\relax
16287 \loop
16288 \catcode\count0=11\relax
16289 \advance\count0 by 1\relax
16290 \ifnum\count0<256\repeat}%

```

## 3.4 ConT<sub>E</sub>Xt Implementation

The ConT<sub>E</sub>Xt implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV ConT<sub>E</sub>Xt formats *seem* to implement (the documentation is scarce) the majority of the plain T<sub>E</sub>X format required by the plain T<sub>E</sub>X implementation. As a consequence, we can directly reuse the existing plain T<sub>E</sub>X implementation after supplying the missing plain T<sub>E</sub>X macros.

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `\enableregime` macro. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the filecontents L<sup>A</sup>T<sub>E</sub>X package.

```

16291 \def\markdownMakeOther{%
16292 \count0=128\relax
16293 \loop
16294 \catcode\count0=11\relax
16295 \advance\count0 by 1\relax
16296 \ifnum\count0<256\repeat

```

On top of that, make the pipe character (`|`) inactive during the scanning. This is necessary, since the character is active in ConT<sub>E</sub>Xt.

```

16297 \catcode`|=12}%

```

### 3.4.1 Typesetting Markdown

The `\inputmarkdown` and `\inputyaml` macros are defined to accept an optional argument with options recognized by the ConT<sub>E</sub>Xt interface (see Section 2.4.2).

```

16298 \long\def\inputmarkdown{%
16299 \dosingleempty
16300 \doinputmarkdown}%

```

```

16301 \long\def\doinputmarkdown[#1]#2{%
16302 \begingroup
16303 \iffirstargument
16304 \setupmarkdown[#1]%
16305 \fi
16306 \markdownInput{#2}%
16307 \endgroup}%
16308 \long\def\inputyaml{%
16309 \dosingleempty
16310 \doinputyaml}%
16311 \long\def\doinputyaml[#1]#2{%
16312 \doinputmarkdown
16313 [jekyllData, expectJekyllData, ensureJekyllData, #1]{#2}}%

```

The `\startmarkdown`, `\stopmarkdown`, `\startyaml`, and `\stopyaml` macros are implemented using the `\markdownReadAndConvert` macro.

In Knuth’s  $\text{\TeX}$ , trailing spaces are removed very early on when a line is being put to the input buffer. [18, sec. 31]. According to Eijkhout [19, sec. 2.2], this is because “these spaces are hard to see in an editor”. At the moment, there is no option to suppress this behavior in (Lua) $\text{\TeX}$ , but Con $\text{\TeX}$ t MkIV funnels all input through its own input handler. This makes it possible to suppress the removal of trailing spaces in Con $\text{\TeX}$ t MkIV and therefore to insert hard line breaks into markdown text.

```

16314 \startluacode
16315 document.markdown_buffering = false
16316 local function preserve_trailing_spaces(line)
16317 if document.markdown_buffering then
16318 line = line:gsub("[\t][\t]$", "\t\t")
16319 end
16320 return line
16321 end
16322 resolvers.installinputlinehandler(preserve_trailing_spaces)
16323 \stopluacode
16324 \begingroup
16325 \catcode`\|=0%
16326 \catcode`\|=12%
16327 |gdef|startmarkdown{%
16328 |ctxlua{document.markdown_buffering = true}%
16329 |markdownReadAndConvert{\stopmarkdown}%
16330 {|stopmarkdown}}%
16331 |gdef|stopmarkdown{%
16332 |ctxlua{document.markdown_buffering = false}%
16333 |markdownEnd}%
16334 |gdef|startyaml{%
16335 |begingroup
16336 |ctxlua{document.markdown_buffering = true}%
16337 |setupyaml[jekyllData, expectJekyllData, ensureJekyllData]%
16338 |markdownReadAndConvert{\stopyaml}%

```

```

16339 {\stopyaml}}}%
16340 |gdef|stopyaml{%
16341 |ctxlua{document.markdown_buffering = false}%
16342 |yamlEnd}%
16343 |endgroup

```

### 3.4.2 Themes

This section overrides the plain T<sub>E</sub>X implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in ConT<sub>E</sub>Xt themes provided with the Markdown package.

```

16344 \ExplSyntaxOn
16345 \prop_new:N \g_@@_context_loaded_themes_linenos_prop
16346 \prop_new:N \g_@@_context_loaded_themes_versions_prop
16347 \cs_gset:Nn
16348 \@@_load_theme:nnn
16349 {

```

Determine whether either this is a built-in theme according to the prop `\g_@@_context_built_in_themes_prop` or a file named `t-markdowntheme<munged theme name>.tex` exists. If it does, load it. Otherwise, try loading a plain T<sub>E</sub>X theme instead.

```

16350 \bool_if:nTF
16351 {
16352 \bool_lazy_or_p:nn
16353 {
16354 \prop_if_in_p:Nn
16355 \g_@@_context_built_in_themes_prop
16356 { #1 }
16357 }
16358 {
16359 \file_if_exist_p:n
16360 { t - markdown theme #3.tex }
16361 }
16362 }
16363 {
16364 \prop_get:NnNTF
16365 \g_@@_context_loaded_themes_linenos_prop
16366 { #1 }
16367 \l_tmpa_tl
16368 {
16369 \prop_get:NnN
16370 \g_@@_context_loaded_themes_versions_prop
16371 { #1 }
16372 \l_tmpb_tl
16373 \str_if_eq:nVTF

```

```

16374 { #2 }
16375 \l_tmpb_tl
16376 {
16377 \msg_warning:nnnVn
16378 { markdown }
16379 { repeatedly-loaded-context-theme }
16380 { #1 }
16381 \l_tmpa_tl
16382 { #2 }
16383 }
16384 {
16385 \msg_error:nnnnVV
16386 { markdown }
16387 { different-versions-of-context-theme }
16388 { #1 }
16389 { #2 }
16390 \l_tmpb_tl
16391 \l_tmpa_tl
16392 }
16393 }
16394 {
16395 \prop_gput:Nnx
16396 \g_@@_context_loaded_themes_linenos_prop
16397 { #1 }
16398 { \tex_the:D \tex_inputlineno:D } % noqa: W200
16399 \prop_gput:Nnn
16400 \g_@@_context_loaded_themes_versions_prop
16401 { #1 }
16402 { #2 }

```

Load built-in plain TeX themes from the prop `\g_@@_context_built_in_themes_prop` and from the filesystem otherwise.

```

16403 \prop_if_in:NnTF
16404 \g_@@_context_built_in_themes_prop
16405 { #1 }
16406 {
16407 \msg_info:nnnn
16408 { markdown }
16409 { loading-built-in-context-theme }
16410 { #1 }
16411 { #2 }
16412 \prop_item:Nn
16413 \g_@@_context_built_in_themes_prop
16414 { #1 }
16415 }
16416 {
16417 \msg_info:nnnn

```

```

16418 { markdown }
16419 { loading-context-theme }
16420 { #1 }
16421 { #2 }
16422 \usemodule
16423 [t]
16424 [markdown theme #3]
16425 }
16426 }
16427 }
16428 {
16429 \@@_plain_tex_load_theme:nnn
16430 { #1 }
16431 { #2 }
16432 { #3 }
16433 }
16434 }
16435 \msg_new:nnn
16436 { markdown }
16437 { loading-built-in-context-theme }
16438 { Loading~version~#2~of~built-in~ConTeXt~Markdown~theme~#1 }
16439 \msg_new:nnn
16440 { markdown }
16441 { loading-context-theme }
16442 { Loading~version~#2~of~ConTeXt~Markdown~theme~#1 }
16443 \msg_new:nnn
16444 { markdown }
16445 { repeatedly-loaded-context-theme }
16446 {
16447 Version~#3~of~ConTeXt~Markdown~theme~#1~was~previously~
16448 loaded~on~line~#2,~not~loading~it~again
16449 }
16450 \msg_new:nnn
16451 { markdown }
16452 { different-versions-of-context-theme }
16453 {
16454 Tried~to~load~version~#2~of~ConTeXt~Markdown~theme~#1~
16455 but~version~#3~has~already~been~loaded~on~line~#4
16456 }
16457 \ExplSyntaxOff

```

The [witiko/markdown/defaults](#) ConTeXt theme provides default definitions for token renderer prototypes. First, the ConTeXt theme loads the plain T<sub>E</sub>X theme with the default definitions for plain T<sub>E</sub>X:

```
16458 \markdownLoadPlainTeXTheme
```

Next, the ConTeXt theme overrides some of the plain T<sub>E</sub>X definitions. See Section 3.4.3 for the actual definitions.

### 3.4.3 Token Renderer Prototypes

The following configuration should be considered placeholder. If the option `plain` has been enabled (see Section 2.2.2.3), none of the definitions will take effect.

```
16459 \markdownIfOption{plain}{\iffalse}{\iftrue}
16460 \def\markdownRendererHardLineBreakPrototype{\blank}%
16461 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
16462 \def\markdownRendererRightBracePrototype{\textbraceright}%
16463 \def\markdownRendererDollarSignPrototype{\textdollar}%
16464 \def\markdownRendererPercentSignPrototype{\percent}%
16465 \def\markdownRendererUnderscorePrototype{\textunderscore}%
16466 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
16467 \def\markdownRendererBackslashPrototype{\textbackslash}%
16468 \def\markdownRendererTildePrototype{\textasciitilde}%
16469 \def\markdownRendererPipePrototype{\char`|}%
16470 \def\markdownRendererLinkPrototype#1#2#3#4{%
16471 \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
16472 \fi\texttt<\hyphenatedurl{#3}>}}%
16473 \usemodule[database]
16474 \defineseparatedlist
16475 [MarkdownConTeXtCSV]
16476 [separator={,},
16477 before=\bTABLE,after=\eTABLE,
16478 first=\bTR,last=\eTR,
16479 left=\bTD,right=\eTD]
16480 \def\markdownConTeXtCSV{csv}
16481 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
16482 \def\markdownConTeXtCSV@arg{#1}%
16483 \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
16484 \placetable[] [tab:#1]{#4}{%
16485 \processseparatedfile[MarkdownConTeXtCSV][#3]}%
16486 \else
16487 \markdownInput{#3}%
16488 \fi}%
16489 \def\markdownRendererImagePrototype#1#2#3#4{%
16490 \placefigure[] []{#4}{\externalfigure[#3]}}%
16491 \def\markdownRendererUlBeginPrototype{\startitemize}%
16492 \def\markdownRendererUlBeginTightPrototype{\startitemize[packed]}%
16493 \def\markdownRendererUlItemPrototype{\item}%
16494 \def\markdownRendererUlEndPrototype{\stopitemize}%
16495 \def\markdownRendererUlEndTightPrototype{\stopitemize}%
16496 \def\markdownRendererOlBeginPrototype{\startitemize[n]}%
16497 \def\markdownRendererOlBeginTightPrototype{\startitemize[packed,n]}%
16498 \def\markdownRendererOlItemPrototype{\item}%
16499 \def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
16500 \def\markdownRendererOlEndPrototype{\stopitemize}%
16501 \def\markdownRendererOlEndTightPrototype{\stopitemize}%
```

```

16502 \definedescription
16503 [MarkdownConTeXtDlItemPrototype]
16504 [location=hanging,
16505 margin=standard,
16506 headstyle=bold]%
16507 \definestartstop
16508 [MarkdownConTeXtDlPrototype]
16509 [before=\blank,
16510 after=\blank]%
16511 \definestartstop
16512 [MarkdownConTeXtDlTightPrototype]
16513 [before=\blank\startpacked,
16514 after=\stoppacked\blank]%
16515 \def\markdownRendererDlBeginPrototype{%
16516 \startMarkdownConTeXtDlPrototype}%
16517 \def\markdownRendererDlBeginTightPrototype{%
16518 \startMarkdownConTeXtDlTightPrototype}%
16519 \def\markdownRendererDlItemPrototype#1{%
16520 \startMarkdownConTeXtDlItemPrototype{#1}}%
16521 \def\markdownRendererDlItemEndPrototype{%
16522 \stopMarkdownConTeXtDlItemPrototype}%
16523 \def\markdownRendererDlEndPrototype{%
16524 \stopMarkdownConTeXtDlPrototype}%
16525 \def\markdownRendererDlEndTightPrototype{%
16526 \stopMarkdownConTeXtDlTightPrototype}%
16527 \def\markdownRendererEmphasisPrototype#1{{\em#1}}%
16528 \def\markdownRendererStrongEmphasisPrototype#1{{\bf#1}}%
16529 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
16530 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
16531 \def\markdownRendererLineBlockBeginPrototype{%
16532 \begingroup
16533 \def\markdownRendererHardLineBreak{
16534 }%
16535 \startlines
16536 }%
16537 \def\markdownRendererLineBlockEndPrototype{%
16538 \stoplines
16539 \endgroup
16540 }%
16541 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%

```

#### 3.4.3.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

16542 \ExplSyntaxOn
16543 \cs_gset:Npn
16544 \markdownRendererInputFencedCodePrototype#1#2#3

```

```

16545 {
16546 \tl_if_empty:nTF
16547 { #2 }
16548 { \markdownRendererInputVerbatim{#1} }

```

Otherwise, extract the first word of the infostring and treat it as the name of the programming language in which the code block is written. This name is then used in the ConTeXt `\definetying` macro, which allows the user to set up code highlighting mapping as follows:

```

\definetying [latex]
\setuptyping [latex] [option=TEX]

\starttext
 \startmarkdown
~~~ latex
\documentclass{article}
\begin{document}
  Hello world!
\end{document}
~~~
 \stopmarkdown
\stoptext

```

```

16549 {
16550 \regex_extract_once:nnN
16551 { \w* }
16552 { #2 }
16553 \l_tmpa_seq
16554 \seq_pop_left:NN
16555 \l_tmpa_seq
16556 \l_tmpa_tl
16557 \typefile[\l_tmpa_tl][] {#1}
16558 }
16559 }
16560 \ExplSyntaxOff
16561 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
16562 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
16563 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
16564 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
16565 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
16566 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
16567 \def\markdownRendererThematicBreakPrototype{%
16568 \blackrule[height=1pt, width=\hsize]}%
16569 \def\markdownRendererNotePrototype#1{\footnote{#1}}%
16570 \def\markdownRendererTickedBoxPrototype{${\boxtimes}$}

```



```

16571 \def\markdownRendererHalfTickedBoxPrototype{\boxdot$}
16572 \def\markdownRendererUntickedBoxPrototype{\square$}
16573 \def\markdownRendererStrikeThroughPrototype#1{\overstrikes{#1}}
16574 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
16575 \def\markdownRendererSubscriptPrototype#1{\low{#1}}
16576 \def\markdownRendererDisplayMathPrototype#1{%
16577 \startformula#1\stopformula}%

```

### 3.4.3.2 Tables

There is a basic implementation of tables.

```

16578 \newcount\markdownConTeXtRowCounter
16579 \newcount\markdownConTeXtRowTotal
16580 \newcount\markdownConTeXtColumnCounter
16581 \newcount\markdownConTeXtColumnTotal
16582 \newtoks\markdownConTeXtTable
16583 \newtoks\markdownConTeXtTableFloat
16584 \def\markdownRendererTablePrototype#1#2#3{%
16585 \markdownConTeXtTable={}%
16586 \ifx\empty#1\empty
16587 \markdownConTeXtTableFloat={%
16588 \the\markdownConTeXtTable}%
16589 \else
16590 \markdownConTeXtTableFloat={%
16591 \placetable{#1}{\the\markdownConTeXtTable}}%
16592 \fi
16593 \begingroup
16594 \setupTABLE[r][each][topframe=off, bottomframe=off,
16595 leftframe=off, rightframe=off]
16596 \setupTABLE[c][each][topframe=off, bottomframe=off,
16597 leftframe=off, rightframe=off]
16598 \setupTABLE[r][1][topframe=on, bottomframe=on]
16599 \setupTABLE[r][#1][bottomframe=on]
16600 \markdownConTeXtRowCounter=0%
16601 \markdownConTeXtRowTotal=#2%
16602 \markdownConTeXtColumnTotal=#3%
16603 \markdownConTeXtRenderTableRow}
16604 \def\markdownConTeXtRenderTableRow#1{%
16605 \markdownConTeXtColumnCounter=0%
16606 \ifnum\markdownConTeXtRowCounter=0\relax
16607 \markdownConTeXtReadAlignments#1%
16608 \markdownConTeXtTable={\bTABLE}%
16609 \else
16610 \markdownConTeXtTable=\expandafter{%
16611 \the\markdownConTeXtTable\bTR}%
16612 \markdownConTeXtRenderTableCell#1%
16613 \markdownConTeXtTable=\expandafter{%

```

```

16614 \the\markdownConTeXtTable\eTR}%
16615 \fi
16616 \advance\markdownConTeXtRowCounter by 1\relax
16617 \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
16618 \markdownConTeXtTable=\expandafter{%
16619 \the\markdownConTeXtTable\eTABLE}%
16620 \the\markdownConTeXtTableFloat
16621 \endgroup
16622 \expandafter\gobbleoneargument
16623 \fi\markdownConTeXtRenderTableRow}
16624 \def\markdownConTeXtReadAlignments#1{%
16625 \advance\markdownConTeXtColumnCounter by 1\relax
16626 \if#1d%
16627 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
16628 \fi\if#1l%
16629 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
16630 \fi\if#1c%
16631 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
16632 \fi\if#1r%
16633 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
16634 \fi
16635 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
16636 \else
16637 \expandafter\gobbleoneargument
16638 \fi\markdownConTeXtReadAlignments}
16639 \def\markdownConTeXtRenderTableCell#1{%
16640 \advance\markdownConTeXtColumnCounter by 1\relax
16641 \markdownConTeXtTable=\expandafter{%
16642 \the\markdownConTeXtTable\bTD#1\eTD}%
16643 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
16644 \else
16645 \expandafter\gobbleoneargument
16646 \fi\markdownConTeXtRenderTableCell}

```

### 3.4.3.3 Raw Attributes

In the raw block and inline raw span renderer prototypes, default to the plain TeX renderer prototypes, translating raw attribute `context` to `tex`.

```

16647 \ExplSyntaxOn
16648 \cs_gset:Npn
16649 \markdownRendererInputRawInlinePrototype#1#2
16650 {
16651 \str_case:nnF
16652 { #2 }
16653 {
16654 { latex }
16655 {

```

```

16656 \@@_plain_tex_default_input_raw_inline:nn
16657 { #1 }
16658 { context }
16659 }
16660 }
16661 {
16662 \@@_plain_tex_default_input_raw_inline:nn
16663 { #1 }
16664 { #2 }
16665 }
16666 }
16667 \cs_gset:Npn
16668 \markdownRendererInputRawBlockPrototype#1#2
16669 {
16670 \str_case:nnF
16671 { #2 }
16672 {
16673 { context }
16674 {
16675 \@@_plain_tex_default_input_raw_block:nn
16676 { #1 }
16677 { tex }
16678 }
16679 }
16680 {
16681 \@@_plain_tex_default_input_raw_block:nn
16682 { #1 }
16683 { #2 }
16684 }
16685 }
16686 \cs_gset_eq:NN
16687 \markdownRendererInputRawBlockPrototype
16688 \markdownRendererInputRawInlinePrototype
16689 \fi % Closes ` \markdownIfOption{plain}{\iffalse}{\iftrue}`
16690 \ExplSyntaxOff
16691 \stopmodule
16692 \protect

```

At the end of the ConT<sub>E</sub>Xt module, we load the [witiko/markdown/defaults](#) ConT<sub>E</sub>Xt theme with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```

16693 \ExplSyntaxOn
16694 \str_if_eq:VVT
16695 \c_@@_top_layer_tl
16696 \c_@@_option_layer_context_tl
16697 {
16698 \use:c

```

```

16699 { ExplSyntaxOff }
16700 \@@_if_option:nF
16701 { noDefaults }
16702 {
16703 \@@_if_option:nTF
16704 { experimental }
16705 {
16706 \@@_setup:n
16707 { theme = witiko/markdown/defaults@experimental }
16708 }
16709 {
16710 \@@_setup:n
16711 { theme = witiko/markdown/defaults }
16712 }
16713 }
16714 \use:c
16715 { ExplSyntaxOn }
16716 }
16717 \ExplSyntaxOff
16718 \stopmodule
16719 \protect

```

## References

- [1] LuaTeX development team. *LuaTeX reference manual*. Version 1.21. Feb. 1, 2025. URL: <http://mirrors.ctan.org/systems/doc/luatex/luatex.pdf> (visited on 05/12/2025).
- [2] L<sup>A</sup>T<sub>E</sub>X Project. *l3kernel. L<sup>A</sup>T<sub>E</sub>X3 programming conventions*. Dec. 25, 2024. URL: <https://ctan.org/pkg/l3kernel> (visited on 01/06/2025).
- [3] Frank Mittelbach, Ulrike Fischer, and L<sup>A</sup>T<sub>E</sub>X Project. *The documentmetadata-support code*. June 1, 2024. URL: <https://mirrors.ctan.org/macros/latex/required/latex-lab/documentmetadata-support-code.pdf> (visited on 10/21/2024).
- [4] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: <https://www.muni.cz/en/research/projects/32984> (visited on 02/19/2018).
- [5] Anton Sotkov. *File transclusion syntax for Markdown*. Jan. 19, 2017. URL: <https://github.com/iainc/Markdown-Content-Blocks> (visited on 01/08/2018).
- [6] John MacFarlane. *Pandoc. a universal document converter*. 2022. URL: <https://pandoc.org/> (visited on 10/05/2022).

- [7] 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).
- [8] Donald Ervin Knuth. *The T<sub>E</sub>Xbook*. 3rd ed. Vol. A. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. ix, 479. ISBN: 0-201-13447-0.
- [9] 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).
- [10] Till Tantau, Joseph Wright, and Vedran Miletic. *The Beamer class*. Feb. 10, 2021. URL: <https://mirrors.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf> (visited on 02/11/2021).
- [11] Vít Starý Novotný. *Versioned Themes*. Markdown Enhancement Proposal. Oct. 13, 2024. URL: <https://github.com/Witiko/markdown/discussions/514> (visited on 10/21/2024).
- [12] Vít Starý Novotný et al. *Convert control sequence with a variable number of undelimited parameters into a token list*. URL: <https://tex.stackexchange.com/q/716362/70941> (visited on 04/28/2024).
- [13] Vít Starý Novotný. *Routing YAML metadata to expl3 key-values*. Markdown Enhancement Proposal. Oct. 14, 2024. URL: <https://github.com/witiko/markdown/discussions/517> (visited on 01/06/2025).
- [14] Frank Mittelbach. *L<sup>A</sup>T<sub>E</sub>X’s hook management*. June 26, 2024. URL: <https://mirrors.ctan.org/macros/latex/base/lthooks-code.pdf> (visited on 10/02/2024).
- [15] Geoffrey M. Poore. *The minted Package. Highlighted source code in L<sup>A</sup>T<sub>E</sub>X*. July 19, 2017. URL: <https://mirrors.ctan.org/macros/latex/contrib/minted/minted.pdf> (visited on 09/01/2020).
- [16] Roberto Ierusalimsky. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.
- [17] Johannes Braams et al. *The L<sup>A</sup>T<sub>E</sub>X<sub>2<sub>ε</sub></sub> Sources*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/source2e.pdf> (visited on 01/08/2018).
- [18] Donald Ervin Knuth. *T<sub>E</sub>X: The Program*. Vol. B. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. xvi, 594. ISBN: 978-0-201-13437-7.
- [19] Victor Eijkhout. *T<sub>E</sub>X by Topic. A T<sub>E</sub>Xnician’s Reference*. Wokingham, England: Addison-Wesley, Feb. 1, 1992. 307 pp. ISBN: 978-0-201-56882-0.

## Index

autoIdentifiers 21, 33, 86, 101

blankBeforeBlockquote	22
blankBeforeCodeFence	22
blankBeforeDivFence	22
blankBeforeHeading	23
blankBeforeList	23
bracketedSpans	23, 88, 464
breakableBlockquotes	24
cacheDir	4, 17, 19, 59, 60, 160, 172, 375, 396, 415
citationNbsps	24
citations	24, 91, 92
codeSpans	25
contentBlocks	20, 25, 35
contentBlocksLanguageMap	20
contentLevel	26
debugExtensions	9, 20, 26, 318
debugExtensionsFileName	20, 26
defaultOptions	10, 52, 373, 374
definitionLists	27, 96
eagerCache	17, 372
ensureJekyllData	27
entities.char_entity	219
entities.dec_entity	218
entities.hex_entity	218
entities.hex_entity_with_x_char	218
escape_minimal	223
escape_programmatic_text	223
escape_typographic_text	223
expandtabs	280
expectJekyllData	27, 28
experimental	5, 17, 434
extensions	29, 167, 323
extensions.bracketed_spans	324
extensions.citations	325
extensions.content_blocks	329
extensions.definition_lists	332
extensions.fancy_lists	334
extensions.fenced_code	340
extensions.fenced_divs	346
extensions.header_attributes	350
extensions.inline_code_attributes	352

extensions.jekyll_data	368
extensions.line_blocks	352
extensions.link_attributes	354
extensions.mark	353
extensions.notes	355
extensions.pipe_table	358
extensions.raw_inline	362
extensions.strike_through	363
extensions.subscripts	363
extensions.superscripts	364
extensions.tex_math	365
fancyLists	30, 112–118, 434
fencedCode	31, 40, 92, 100, 119, 386, 389
fencedCodeAttributes	31, 86, 100, 389
fencedDiv	101
fencedDivs	32, 42
finalizeCache	17, 21, 32, 33, 59, 60, 159, 372, 374
frozenCache	21, 32, 60, 75, 76, 159, 387, 395
frozenCacheCounter	33, 374, 423, 424
frozenCacheFileName	21, 32, 59, 374
\g_markdown_diagrams_infostrings_prop	391
gfmAutoIdentifiers	21, 33, 86, 101
hashEnumerators	33
headerAttributes	34, 42, 86, 101
html	34, 104, 105, 447
hybrid	35, 35, 40, 46, 48, 62, 76, 120, 160, 223, 281, 423
inlineCodeAttributes	36, 86, 93
inlineNotes	36
\input	56, 374
\inputmarkdown	162, 164, 165, 465
inputTempFileName	60, 62, 417, 418, 420, 421
\inputyaml	162, 164, 465
iterlines	280
jekyllData	3, 27, 28, 37, 129–132, 134
\l_file_search_path_seq	422
languages_json	329, 329
lineBlocks	38, 108

linkAttributes	37, 86, 106, 110, 299, 460
mark	38, 110, 459
\markdown	155, 156, 427
markdown	154, 154, 155, 425, 426
markdown*	154, 154, 159, 425
\markdownBegin	54, 54–56, 152, 154, 155, 162, 163
\markdownCleanup	416
\markdownConvert	416
\markdownEnd	54, 54–56, 152, 154–156, 162, 163
\markdownError	152, 152
\markdownEscape	54, 57, 424
\markdownIfOption	58
\markdownIfSnippetExists	80
\markdownInfo	152, 152
\markdownInput	54, 56, 154, 156, 159, 164, 422, 425
\markdownInputFilename	415
\markdownInputFileStream	416
\markdownInputPlainTeX	425
\markdownLoadPlainTeXTheme	160, 166, 385
\markdownLuaExecute	419, 422
\markdownLuaOptions	412, 416
\markdownMakeOther	152, 465
\markdownOptionFinalizeCache	59
\markdownOptionFrozenCache	59
\markdownOptionHybrid	62
\markdownOptionInputTempFileName	60
\markdownOptionNoDefaults	61
\markdownOptionOutputDir	60, 60, 63, 64
\markdownOptionPlain	61
\markdownOptionStripPercentSigns	62
\markdownOutputFileStream	416
\markdownPrepare	415
\markdownPrepareInputFilename	415
\markdownPrepareLuaOptions	412
\markdownReadAndConvert	152, 416, 425–427, 466
\markdownReadAndConvertProcessLine	418, 418
\markdownReadAndConvertStripPercentSigns	417
\markdownReadAndConvertTab	416
\markdownRendererAttributeClassName	86
\markdownRendererAttributeIdentifier	86
\markdownRendererAttributeKeyValue	86



<code>\markdownRendererBlockQuoteBegin</code>	87
<code>\markdownRendererBlockQuoteEnd</code>	88
<code>\markdownRendererBracketedSpanAttributeContextBegin</code>	88
<code>\markdownRendererBracketedSpanAttributeContextEnd</code>	88
<code>\markdownRendererCite</code>	91, 92
<code>\markdownRendererCodeSpan</code>	93
<code>\markdownRendererCodeSpanAttributeContextBegin</code>	93
<code>\markdownRendererCodeSpanAttributeContextEnd</code>	93
<code>\markdownRendererContentBlock</code>	94, 94
<code>\markdownRendererContentBlockCode</code>	95
<code>\markdownRendererContentBlockOnlineImage</code>	94
<code>\markdownRendererDisplayMath</code>	126
<code>\markdownRendererDlBegin</code>	96
<code>\markdownRendererDlBeginTight</code>	96
<code>\markdownRendererDlDefinitionBegin</code>	97
<code>\markdownRendererDlDefinitionEnd</code>	98
<code>\markdownRendererDlEnd</code>	98
<code>\markdownRendererDlEndTight</code>	98
<code>\markdownRendererDlItem</code>	96
<code>\markdownRendererDlItemEnd</code>	97
<code>\markdownRendererDocumentBegin</code>	111
<code>\markdownRendererDocumentEnd</code>	111
<code>\markdownRendererEllipsis</code>	42, 99
<code>\markdownRendererEmphasis</code>	99, 139
<code>\markdownRendererError</code>	128
<code>\markdownRendererFancyOlBegin</code>	113, 114
<code>\markdownRendererFancyOlBeginTight</code>	114
<code>\markdownRendererFancyOlEnd</code>	117
<code>\markdownRendererFancyOlEndTight</code>	118
<code>\markdownRendererFancyOlItem</code>	115
<code>\markdownRendererFancyOlItemEnd</code>	116
<code>\markdownRendererFancyOlItemWithNumber</code>	116
<code>\markdownRendererFencedCodeAttributeContextBegin</code>	100
<code>\markdownRendererFencedCodeAttributeContextEnd</code>	100
<code>\markdownRendererFencedDivAttributeContextBegin</code>	101
<code>\markdownRendererFencedDivAttributeContextEnd</code>	101
<code>\markdownRendererHalfTickedBox</code>	127
<code>\markdownRendererHardLineBreak</code>	109
<code>\markdownRendererHeaderAttributeContextBegin</code>	101
<code>\markdownRendererHeaderAttributeContextEnd</code>	101
<code>\markdownRendererHeadingFive</code>	104
<code>\markdownRendererHeadingFour</code>	103

<code>\markdownRendererHeadingOne</code>	102
<code>\markdownRendererHeadingSix</code>	104
<code>\markdownRendererHeadingThree</code>	103
<code>\markdownRendererHeadingTwo</code>	103
<code>\markdownRendererImage</code>	106
<code>\markdownRendererImageAttributeContextBegin</code>	106
<code>\markdownRendererImageAttributeContextEnd</code>	106
<code>\markdownRendererInlineHtmlComment</code>	104
<code>\markdownRendererInlineHtmlTag</code>	105
<code>\markdownRendererInlineMath</code>	126
<code>\markdownRendererInputBlockHtmlElement</code>	105
<code>\markdownRendererInputFencedCode</code>	92
<code>\markdownRendererInputRawBlock</code>	119
<code>\markdownRendererInputRawInline</code>	118
<code>\markdownRendererInputVerbatim</code>	92
<code>\markdownRendererInterblockSeparator</code>	107
<code>\markdownRendererJekyllDataBegin</code>	129
<code>\markdownRendererJekyllDataBoolean</code>	131
<code>\markdownRendererJekyllDataEmpty</code>	134
<code>\markdownRendererJekyllDataEnd</code>	129
<code>\markdownRendererJekyllDataMappingBegin</code>	129
<code>\markdownRendererJekyllDataMappingEnd</code>	130
<code>\markdownRendererJekyllDataNumber</code>	132
<code>\markdownRendererJekyllDataProgrammaticString</code>	132, 132, 133
<code>\markdownRendererJekyllDataSequenceBegin</code>	130
<code>\markdownRendererJekyllDataSequenceEnd</code>	131
<code>\markdownRendererJekyllDataString</code>	133, 137
<code>\markdownRendererJekyllDataStringPrototype</code>	147
<code>\markdownRendererJekyllDataTypographicString</code>	132, 132, 133, 369
<code>\markdownRendererLineBlockBegin</code>	108
<code>\markdownRendererLineBlockEnd</code>	108
<code>\markdownRendererLink</code>	109, 139
<code>\markdownRendererLinkAttributeContextBegin</code>	110
<code>\markdownRendererLinkAttributeContextEnd</code>	110
<code>\markdownRendererMark</code>	110
<code>\markdownRendererNbsp</code>	112
<code>\markdownRendererNote</code>	112
<code>\markdownRendererOlBegin</code>	112
<code>\markdownRendererOlBeginTight</code>	113
<code>\markdownRendererOlEnd</code>	116
<code>\markdownRendererOlEndTight</code>	117
<code>\markdownRendererOlItem</code>	43, 114

<code>\markdownRendererOliItemEnd</code>	114
<code>\markdownRendererOliItemWithNumber</code>	42, 115
<code>\markdownRendererParagraphSeparator</code>	107
<code>\markdownRendererReplacementCharacter</code>	120
<code>\markdownRendererSectionBegin</code>	119
<code>\markdownRendererSectionEnd</code>	119
<code>\markdownRendererSoftLineBreak</code>	108
<code>\markdownRendererStrikeThrough</code>	123
<code>\markdownRendererStrongEmphasis</code>	100
<code>\markdownRendererSubscript</code>	124
<code>\markdownRendererSuperscript</code>	124
<code>\markdownRendererTable</code>	125
<code>\markdownRendererTableAttributeContextBegin</code>	124
<code>\markdownRendererTableAttributeContextEnd</code>	124
<code>\markdownRendererTextCite</code>	92
<code>\markdownRendererThematicBreak</code>	126
<code>\markdownRendererTickedBox</code>	127
<code>\markdownRendererUlBegin</code>	89
<code>\markdownRendererUlBeginTight</code>	89
<code>\markdownRendererUlEnd</code>	90
<code>\markdownRendererUlEndTight</code>	91
<code>\markdownRendererUliItem</code>	90
<code>\markdownRendererUliItemEnd</code>	90
<code>\markdownRendererUntickedBox</code>	127
<code>\markdownRendererWarning</code>	128
<code>\markdownSetup</code>	58, 58, 62, 158, 159, 165, 426, 433
<code>\markdownSetupSnippet</code>	79, 79
<code>\markdownThemeVersion</code>	69, 69, 70
<code>\markdownWarning</code>	152, 152
<code>\markinline</code>	54, 55, 56, 154, 156, 420, 424
<code>\markinlinePlainTeX</code>	424
<code>new</code>	7, 18, 372, 374
<code>notes</code>	39, 112
<code>parsers</code>	239, 279
<code>parsers.commented_line</code>	260
<code>parsers.unicode_data</code>	240
<code>pipeTables</code>	7, 39, 45, 125
<code>preserveTabs</code>	40, 43, 280
<code>rawAttribute</code>	35, 40, 40, 118, 119
<code>reader</code>	8, 30, 167, 239, 279, 324

reader->add_special_character	8, 9, 30, 318
reader->auto_link_email	307
reader->auto_link_url	307
reader->create_parser	280
reader->finalize_grammar	313, 379
reader->initialize_named_group	318
reader->insert_pattern	8, 9, 30, 314, 320
reader->lookup_note_reference	292
reader->lookup_reference	292
reader->normalize_tag	279
reader->options	279
reader->parser_functions	280
reader->parser_functions.name	280
reader->parsers	279, 279
reader->register_link	292
reader->update_rule	314, 317, 320
reader->writer	279
reader.new	279, 279, 379
relativeReferences	41
\setupmarkdown	165, 165
\setupyaml	165
shiftHeadings	7, 41
singletonCache	18
slice	7, 42, 220, 232, 233
smartEllipses	42, 99, 160
\startmarkdown	162, 162, 163, 466
startNumber	42, 114–116
\startyaml	162, 163, 466
\stopmarkdown	162, 162, 163, 466
\stopyaml	162, 163, 466
strikeThrough	43, 123, 459
stripIndent	43, 280
stripPercentSigns	417
subscripts	44, 124
superscripts	44, 124
syntax	315, 319
tableAttributes	44, 124, 456
tableCaptions	7, 44, 45, 124
taskLists	45, 127, 447
texComments	46, 281

<code>texMathDollars</code>	36, 46, 126
<code>texMathDoubleBackslash</code>	36, 47, 126
<code>texMathSingleBackslash</code>	36, 47, 126
<code>tightLists</code>	47, 89, 91, 96, 98, 113, 114, 117, 118, 434
<code>underscores</code>	48
<code>unicodeNormalization</code>	18, 19
<code>unicodeNormalizationForm</code>	18, 19
<code>util.cache</code>	168, 168
<code>util.cache_verbatim</code>	168
<code>util.encode_json_string</code>	168
<code>util.err</code>	168
<code>util.escaper</code>	171
<code>util.expand_tabs_in_line</code>	169
<code>util.flatten</code>	169
<code>util.intersperse</code>	170
<code>util.map</code>	171
<code>util.pathname</code>	172
<code>util.rope_last</code>	170
<code>util.rope_to_string</code>	170
<code>util.salt</code>	172
<code>util.table_copy</code>	168
<code>util.walk</code>	169, 170
<code>util.warning</code>	172
<code>walkable_syntax</code>	8, 20, 26, 313, 314, 317–319
<code>writer</code>	167, 167, 219, 324
<code>writer-&gt;active_attributes</code>	231, 231, 232
<code>writer-&gt;attribute_type_levels</code>	231
<code>writer-&gt;attributes</code>	229
<code>writer-&gt;block_html_element</code>	227
<code>writer-&gt;blockquote</code>	228
<code>writer-&gt;bulletitem</code>	226
<code>writer-&gt;bulletlist</code>	225
<code>writer-&gt;citations</code>	325
<code>writer-&gt;code</code>	224
<code>writer-&gt;contentblock</code>	330
<code>writer-&gt;defer_call</code>	239, 239
<code>writer-&gt;definitionlist</code>	332
<code>writer-&gt;display_math</code>	365
<code>writer-&gt;div_begin</code>	346
<code>writer-&gt;div_end</code>	346

writer->document	228
writer->ellipsis	221
writer->emphasis	227
writer->error	224
writer->escape	223
writer->escaped_chars	222, 223
writer->escaped_minimal_strings	222, 223
writer->escaped_strings	222
writer->escaped_uri_chars	222, 223
writer->fancyitem	336
writer->fancylist	335
writer->fencedCode	341
writer->flatten_inlines	220, 220
writer->get_state	238
writer->hard_line_break	221
writer->heading	236
writer->identifier	223
writer->image	225
writer->infostring	223
writer->inline_html_comment	227
writer->inline_html_tag	227
writer->inline_math	365
writer->interblocksep	221
writer->is_writing	220, 220
writer->jekyllData	369
writer->lineblock	352
writer->link	224
writer->mark	353
writer->math	223
writer->nbsp	220
writer->note	356
writer->options	219
writer->ordereditem	226
writer->orderedlist	226
writer->paragraph	221
writer->paragraphsep	221
writer->plain	220
writer->pop_attributes	231, 232
writer->push_attributes	231, 231, 232
writer->rawBlock	341
writer->rawInline	362
writer->set_state	238

writer->slice_begin	220
writer->slice_end	220
writer->soft_line_break	221
writer->space	220
writer->span	324
writer->strike_through	363
writer->string	223
writer->strong	227
writer->subscript	363
writer->superscript	364
writer->table	359
writer->thematic_break	222
writer->textbox	227
writer->undosep	221, 322
writer->uri	223
writer->verbatim	228
writer->warning	172, 223
writer.new	219, 219, 379
\yaml	156
yaml	154, 155, 156, 425
\yamlBegin	54, 55, 152, 155, 163
\yamlEnd	54, 55, 152, 155, 156, 163
\yamlInput	54, 56, 154, 157, 164, 425
\yamlSetup	58