

A Markdown Interpreter for T_EX

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

Version 3.11.0-0-ga9095584
2025-02-21

Contents

1	Introduction	1	3	Implementation	166
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	423
2	Interfaces	7	3.4	ConT _E Xt Implementation	463
2.1	Lua Interface	7			
2.2	Plain T _E X Interface	53			
2.3	L ^A T _E X Interface	153			
2.4	ConT _E Xt Interface	161			
				References	475
				Index	476

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	75
5	An example mindmap	76
6	An example UML sequence diagram	77
7	The banner of the Markdown package	78
8	A pushdown automaton that recognizes T _E X comments	260

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 Lua_{TEX} engine (though not necessarily in the LuaMeta_{TEX} engine).

LPeg \geq 0.10 A pattern-matching library for the writing of recursive descent parsers via the Parsing Expression Grammars (PEGs). It is used by the Lunamark library to parse the markdown input. LPeg \geq 0.10 is included in Lua_{TEX} \geq 0.72.0 (T_EX Live \geq 2013).

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

MD5 A library that provides MD5 crypto functions. It is used by the Lunamark library to compute the digest of the input for caching purposes. MD5 is included in all releases of Lua_{TEX} (T_EX Live \geq 2008).

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

Kpathsea A package that implements the loading of third-party Lua libraries and looking up files in the T_EX 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 \geq 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 \leq 2019. It is used to implement reflection capabilities that allow us to enumerate and inspect high-level concepts such as options, renderers, and renderer prototypes.

```
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^AT_EX Requirements

The L^AT_EX part of the package requires that the L^AT_EX 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^AT_EX 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 amsfonts`

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 $\text{T}_{\text{E}}\text{X}$ implementation of the package draws inspiration from several sources including the source code of $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\epsilon}$, the minted package by Geoffrey M. Poore, which likewise tackles the issue of interfacing with an external interpreter from $\text{T}_{\text{E}}\text{X}$, the filecontents package by Scott Pakin and others.

2 Interfaces

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

Since neither $\text{T}_{\text{E}}\text{X}$ nor Lua provide interfaces as a language construct, the separation to interfaces and implementations is a *gentlemen's agreement*. It serves as a means of structuring this documentation and as a promise to the user that if they only access the package through the interface, the future minor versions of the package should remain backwards compatible.

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

2.1 Lua Interface

The Lua interface provides the conversion from UTF-8 encoded markdown to plain $\text{T}_{\text{E}}\text{X}$. This interface is used by the plain $\text{T}_{\text{E}}\text{X}$ implementation (see Section 3.2) and will be of interest to the developers of other packages and Lua modules.

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

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

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

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

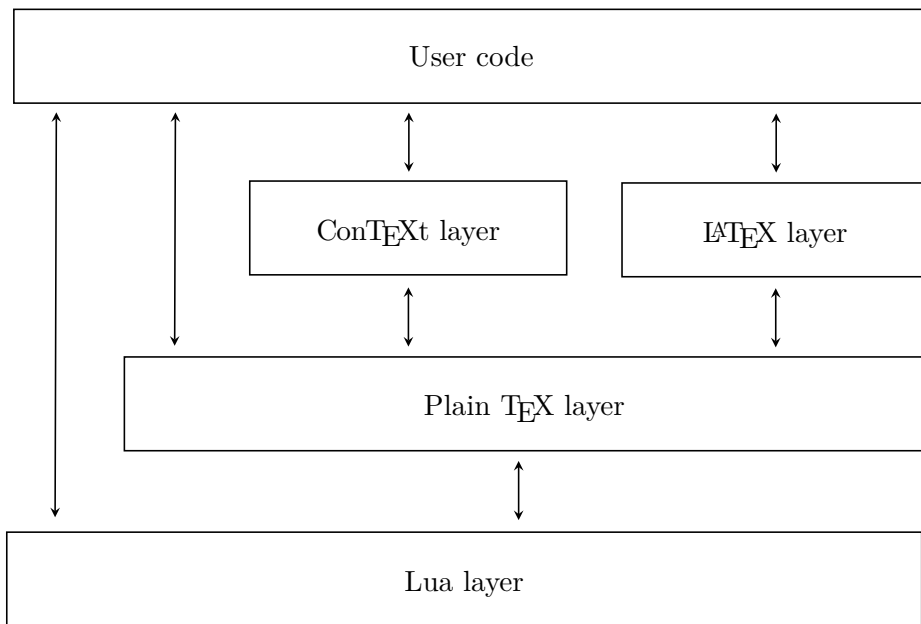


Figure 1: A block diagram of the Markdown package

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 \@@_add_lua_option:nnn
426   { cacheDir }
427   { path }
428   { \markdownOptionOutputDir / _markdown_ \jobname }
429 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.

```

430 \@@_add_lua_option:nnn
431   { contentBlocksLanguageMap }
432   { path }
433   { markdown-languages.json }
434 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.

```

435 \@@_add_lua_option:nnn
436   { debugExtensionsFileName }
437   { path }
438   { \markdownOptionOutputDir / \jobname .debug-extensions.json }
439 defaultOptions.debugExtensionsFileName = "debug-extensions.json"

```

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

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

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

```

440 \@@_add_lua_option:nnn
441   { frozenCacheFileName }
442   { path }
443   { \markdownOptionCacheDir / frozenCache.tex }
444 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`.

```

445 \@@_add_lua_option:nnn
446   { autoIdentifiers }
447   { boolean }
448   { false }
449 defaultOptions.autoIdentifiers = false

```

`blankBeforeBlockquote=true, false` default: false

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

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

```

450 \@@_add_lua_option:nnn
451   { blankBeforeBlockquote }
452   { boolean }
453   { false }
454 defaultOptions.blankBeforeBlockquote = false

```

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

`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.

```
455 \@@_add_lua_option:nnn
456   { blankBeforeCodeFence }
457   { boolean }
458   { false }
459 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.

```
460 \@@_add_lua_option:nnn
461   { blankBeforeDivFence }
462   { boolean }
463   { false }
464 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.

```
465 \@@_add_lua_option:nnn
466   { blankBeforeHeading }
467   { boolean }
468   { false }
469 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.

```
470 \@@_add_lua_option:nnn
471   { blankBeforeList }
472   { boolean }
473   { false }
474 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.

```
475 \@@_add_lua_option:nnn
476 { bracketedSpans }
477 { boolean }
478 { false }
479 defaultOptions.bracketedSpans = false
```

`breakableBlockquotes=true, false` default: true

true A blank line separates block quotes.

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

```
480 \@@_add_lua_option:nnn
481 { breakableBlockquotes }
482 { boolean }
483 { true }
484 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.

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

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

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

`false` Disable the Pandoc citation syntax extension.

```
490 \@@_add_lua_option:nnn
491 { citations }
492 { boolean }
493 { false }
494 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.``
```

```
495 \@@_add_lua_option:nnn
496 { codeSpans }
497 { boolean }
498 { true }
499 defaultOptions.codeSpans = true
```

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

`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.

```
500 \@@_add_lua_option:nnn
501   { contentBlocks }
502   { boolean }
503   { false }

504 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
```

```
505 \@@_add_lua_option:nnn
506   { contentLevel }
507   { string }
508   { block }

509 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.

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

514 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.

```
515 \@@_add_lua_option:nnn
516   { definitionLists }
517   { boolean }
518   { false }

519 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.

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

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

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

```
525 \@_add_lua_option:nnn
526 { expectJekyllData }
527 { boolean }
528 { false }

529 defaultOptions.expectJekyllData = false
```

`extensions=<filenames>`

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

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

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

```

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

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

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

return strike_through

```

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

```

530 metadata.user_extension_api_version = 2
531 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.

```

532 \cs_generate_variant:Nn
533 \@@_add_lua_option:nnn
534 { nnV }
535 \@@_add_lua_option:nnV
536 { extensions }
537 { clist }
538 \c_empty_clist
539 defaultOptions.extensions = {}

```

`fancyLists=true, false`

default: false

`true` Enable the Pandoc fancy list syntax extension¹¹:

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

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

false Disable the Pandoc fancy list syntax extension.

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

544 defaultOptions.fancyLists = false
```

fencedCode=true, false

default: true

true Enable the commonmark fenced code block extension:

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

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

false Disable the commonmark fenced code block extension.

```
545 \@@_add_lua_option:nnn
546 { fencedCode }
547 { boolean }
548 { true }

549 defaultOptions.fencedCode = true
```

`fencedCodeAttributes=true, false`

default: false

true Enable the Pandoc fenced code attribute syntax extension¹²:

```
~~~~ {#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.

```
550 \@@_add_lua_option:nnn
551   { fencedCodeAttributes }
552   { boolean }
553   { false }

554 defaultOptions.fencedCodeAttributes = false
```

`fencedDivs=true, false`

default: false

true Enable the Pandoc fenced div syntax extension¹³:

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

And another.
:::::
```

false Disable the Pandoc fenced div syntax extension.

```
555 \@@_add_lua_option:nnn
556   { fencedDivs }
557   { boolean }
558   { false }

559 defaultOptions.fencedDivs = false
```

¹²See https://pandoc.org/MANUAL.html#extension-fenced_code_attributes.

¹³See https://pandoc.org/MANUAL.html#extension-fenced_divs.

`finalizeCache=true, false` default: false

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

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

```
560 \@@_add_lua_option:nnn
561   { finalizeCache }
562   { boolean }
563   { false }

564 defaultOptions.finalizeCache = false
```

`frozenCacheCounter=<number>` default: 0

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

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

```
565 \@@_add_lua_option:nnn
566   { frozenCacheCounter }
567   { counter }
568   { 0 }

569 defaultOptions.frozenCacheCounter = 0
```

`gfmAutoIdentifiers=true, false` default: false

`true` Enable the Pandoc GitHub-flavored auto identifiers syntax extension¹⁴:

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

```
# 123 Sesame Street
```

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

¹⁴See https://pandoc.org/MANUAL.html#extension-gfm_auto_identifiers.

See also the option [autoIdentifiers](#).

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

574 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.

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

579 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.

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

584 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.

```
585 \@@_add_lua_option:nm
586   { html }
587   { boolean }
588   { true }

589 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 \TeX commands between dollar signs or backslash-escaped brackets:

```

$H_2 O$ is a liquid.

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

```

```

590 \@_add_lua_option:nmn
591   { hybrid }
592   { boolean }
593   { false }

594 defaultOptions.hybrid = false

```

`inlineCodeAttributes=true, false` default: false

`true` Enable the Pandoc inline code span attribute extension¹⁵:

```

`<$>`{.haskell}

```

¹⁵See https://pandoc.org/MANUAL.html#extension-inline_code_attributes.

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

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

599 defaultOptions.inlineCodeAttributes = false
```

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

`true` Enable the Pandoc inline note syntax extension¹⁶:

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

`false` Disable the Pandoc inline note syntax extension.

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

604 defaultOptions.inlineNotes = false
```

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

`true` Enable the Pandoc YAML metadata block syntax extension¹⁷ 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.
---
```

¹⁶See https://pandoc.org/MANUAL.html#extension-inline_notes.

¹⁷See https://pandoc.org/MANUAL.html#extension-yaml_metadata_block.

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

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

linkAttributes=true, false default: false

true Enable the Pandoc link and image attribute syntax extension¹⁸:

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

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

false Enable the Pandoc link and image attribute syntax extension.

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

lineBlocks=true, false default: false

true Enable the Pandoc line block syntax extension¹⁹:

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

false Disable the Pandoc line block syntax extension.

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

¹⁸See https://pandoc.org/MANUAL.html#extension-link_attributes.

¹⁹See https://pandoc.org/MANUAL.html#extension-line_blocks.

`mark=true, false` default: false

true Enable the Pandoc mark syntax extension²⁰:

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

false Disable the Pandoc mark syntax extension.

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

`notes=true, false` default: false

true Enable the Pandoc note syntax extension²¹:

```
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.

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

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

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

`pipeTables=true, false`

default: false

true Enable the PHP Markdown pipe table syntax extension:

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

false Disable the PHP Markdown pipe table syntax extension.

```
630 \@@_add_lua_option:nnn
631 { pipeTables }
632 { boolean }
633 { false }
634 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.

```
635 \@@_add_lua_option:nnn
636 { preserveTabs }
637 { boolean }
638 { true }
639 defaultOptions.preserveTabs = true
```

`rawAttribute=true, false`

default: false

true Enable the Pandoc raw attribute syntax extension²²:

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

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

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

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

```

 c & d
 \end{dcases}
\]

```

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

`false`      Disable the Pandoc raw attribute syntax extension.

```

640 \@@_add_lua_option:nmn
641 { rawAttribute }
642 { boolean }
643 { false }
644 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.

```

645 \@@_add_lua_option:nmn
646 { relativeReferences }
647 { boolean }
648 { false }
649 defaultOptions.relativeReferences = false
```

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



`shiftHeadings`=*<shift amount>* default: 0

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

```
650 \@@_add_lua_option:nnn
651 { shiftHeadings }
652 { number }
653 { 0 }

654 defaultOptions.shiftHeadings = 0
```

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

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

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

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

```
655 \@@_add_lua_option:nnn
656 { slice }
657 { slice }
658 { ^-$ }

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

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

- |                    |                                                                                            |
|--------------------|--------------------------------------------------------------------------------------------|
| <code>true</code>  | Convert any ellipses in the input to the <code>\markdownRendererEllipsis</code> TeX macro. |
| <code>false</code> | Preserve all ellipses in the input.                                                        |

```

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

664 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 `\markdownRendererO1ItemWithNumber` T<sub>E</sub>X macro.
- false**     Ignore the numbers in the ordered list items. Each item will only produce a `\markdownRendererO1Item` T<sub>E</sub>X macro.

```

665 \@@_add_lua_option:nnn
666 { startNumber }
667 { boolean }
668 { true }

669 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.

```

670 \@@_add_lua_option:nnn
671 { strikeThrough }
672 { boolean }
673 { false }

674 defaultOptions.strikeThrough = false

```

---

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

`stripIndent=true, false`

default: false

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

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

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

```
675 \@@_add_lua_option:nnn
676 { stripIndent }
677 { boolean }
678 { false }
679 defaultOptions.stripIndent = false
```

`subscripts=true, false`

default: false

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

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

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

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

---

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

`superscripts=true, false`

default: false

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

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

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

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

`tableAttributes=true, false`

default: false

**true**

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

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

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

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

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

---

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

`tableCaptions=true, false`

default: `false`

`true`

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

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

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

`false` Disable the Pandoc table caption syntax extension.

```
695 \@@_add_lua_option:nnn
696 { tableCaptions }
697 { boolean }
698 { false }

699 defaultOptions.tableCaptions = false
```

`taskLists=true, false`

default: `false`

`true` Enable the Pandoc task list syntax extension²⁸:

```
- [ ] 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.

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

704 defaultOptions.taskLists = false
```

²⁷See https://pandoc.org/MANUAL.html#extension-table_captions.

²⁸See https://pandoc.org/MANUAL.html#extension-task_lists.

`texComments=true, false`

default: false

true Strip T_EX-style comments.

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

Always enabled when `hybrid` is enabled.

false Do not strip T_EX-style comments.

```
705 \@@_add_lua_option:nnn
706   { texComments }
707   { boolean }
708   { false }
709 defaultOptions.texComments = false
```

`texMathDollars=true, false`

default: false

true Enable the Pandoc dollar math syntax extension²⁹:

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

false Disable the Pandoc dollar math syntax extension.

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

²⁹See https://pandoc.org/MANUAL.html#extension-tex_math_dollars.

`texMathDoubleBackslash=true, false` default: false

true Enable the Pandoc double backslash math syntax extension³⁰:

```
inline math: \\(E=mc^2\\)
display math: \\[E=mc^2\\]
```

false Disable the Pandoc double backslash math syntax extension.

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

`texMathSingleBackslash=true, false` default: false

true Enable the Pandoc single backslash math syntax extension³¹:

```
inline math: \ (E=mc^2\ )
display math: \ [E=mc^2\ ]
```

false Disable the Pandoc single backslash math syntax extension.

```
720 \\@@_add_lua_option:nnn
721   { texMathSingleBackslash }
722   { boolean }
723   { false }
724 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:

³⁰See https://pandoc.org/MANUAL.html#extension-tex_math_double_backslash.

³¹See 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.

```

725 \@@_add_lua_option:nmn
726   { tightLists }
727   { boolean }
728   { true }

729 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.

```

730 \@@_add_lua_option:nmn
731   { underscores }
732   { boolean }
733   { true }
734 \ExplSyntaxOff

735 defaultOptions.underscores = true

```


2.1.7 Command-Line Interface

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

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

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

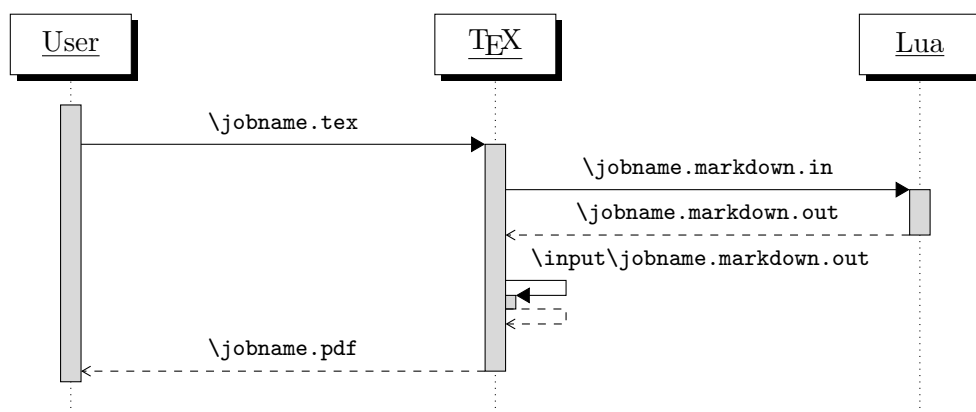


Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the T_EX interface

```

736 .TH MARKDOWN2TEX 1 "(((LASTMODIFIED)))"
737 .SH NAME
738 markdown2tex \- convert .md files to .tex
739 .SH SYNOPSIS
</lua-cli-manpage> <*lua-cli>
740 local HELP_STRING = "Usage: " .. [[
</lua-cli> <*lua-cli,lua-cli-manpage>
741 markdown2tex [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
742
</lua-cli,lua-cli-manpage> <*lua-cli-manpage>
743 .SH DESCRIPTION
  
```

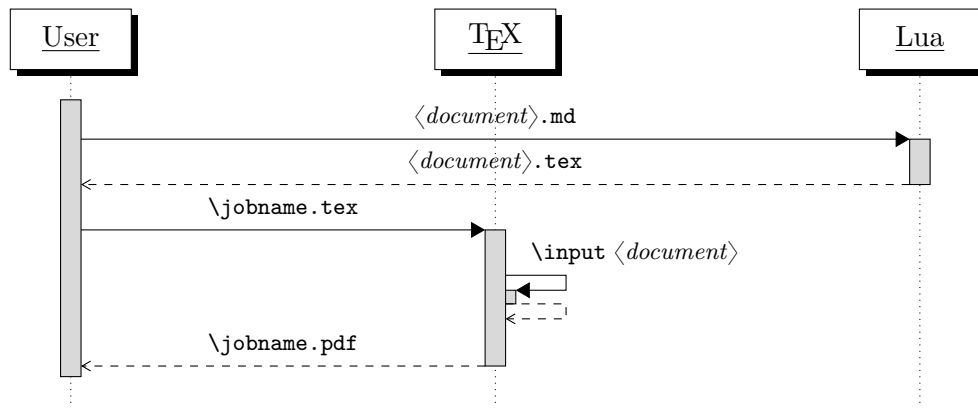


Figure 3: A sequence diagram of the Markdown package typesetting a markdown document using the Lua command-line interface

```

744 % \end{macrocode}
745 </lua-cli-manpage>
746 <*lua-cli, lua-cli-manpage>
747 % \begin{macrocode}
748 OPTIONS are documented in Section 2.2.1 of the Markdown Package User
749 Manual (https://ctan.org/pkg/markdown).
750
751 When OUTPUT_FILE is unspecified, the result of the conversion will be
752 written to the standard output. When INPUT_FILE is also unspecified, the
753 result of the conversion will be read from the standard input.
754 % \end{macrocode}
755 </lua-cli, lua-cli-manpage>
756 <*lua-cli>
757 % \begin{macrocode}
758
759 Report bugs to: witiko@mail.muni.cz
760 Markdown package home page: <https://github.com/witiko/markdown>]]
761
762 local VERSION_STRING = [[
763 markdown2tex (Markdown) ]] .. metadata.version .. [[
764
765 Copyright (C) ]] .. table.concat(metadata.copyright,
766                                     "\nCopyright (C) ") .. [[
767
768 License: ]] .. metadata.license
769
770 local function warn(s)
771   io.stderr:write("Warning: " .. s .. "\n")
772 end
  
```

```

773
774 local function error(s)
775   io.stderr:write("Error: " .. s .. "\n")
776   os.exit(1)
777 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.

```

778 local function camel_case(option_name)
779   local cased_option_name = option_name:gsub("_(%l)", function(match)
780     return match:sub(2, 2):upper()
781   end)
782   return cased_option_name
783 end
784
785 local function snake_case(option_name)
786   local cased_option_name = option_name:gsub("%l%u", function(match)
787     return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()
788   end)
789   return cased_option_name
790 end
791
792 local cases = {camel_case, snake_case}
793 local various_case_options = {}
794 for option_name, _ in pairs(defaultOptions) do
795   for _, case in ipairs(cases) do
796     various_case_options[case(option_name)] = option_name
797   end
798 end
799
800 local process_options = true
801 local options = {}
802 local input_filename
803 local output_filename
804 for i = 1, #arg do
805   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.

```

806     if arg[i] == "--" then
807       process_options = false
808       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.

```

809     elseif arg[i]:match("=") then
810         local key, value = arg[i]:match("(.-)=(.*)")
811         if defaultOptions[key] == nil and
812            various_case_options[key] ~= nil then
813             key = various_case_options[key]
814         end

```

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

```

815     local default_type = type(defaultOptions[key])
816     if default_type == "boolean" then
817         options[key] = (value == "true")
818     elseif default_type == "number" then
819         options[key] = tonumber(value)
820     elseif default_type == "table" then
821         options[key] = {}
822         for item in value:gmatch("[^,]+") do
823             table.insert(options[key], item)
824         end
825     else
826         if default_type ~= "string" then
827             if default_type == "nil" then
828                 warn('Option "' .. key .. '" not recognized.')
829             else
830                 warn('Option "' .. key .. '" type not recognized, ' ..
831                    'please file a report to the package maintainer.')
832             end
833             warn('Parsing the ' .. 'value "' .. value .. '" of option "' ..
834                key .. '" as a string.')
835         end
836         options[key] = value
837     end
838     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.

```

839     elseif arg[i] == "--help" or arg[i] == "-h" then
840         print(HELP_STRING)
841         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.

```

842     elseif arg[i] == "--version" or arg[i] == "-v" then
843         print(VERSION_STRING)
844         os.exit()
845     end
846 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_EX document.

```

847 if input_filename == nil then
848     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_EX document that will result from the conversion.

```

849 elseif output_filename == nil then
850     output_filename = arg[i]
851 else
852     error('Unexpected argument: "' .. arg[i] .. "'.')
853 end
854 ::continue::
855 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_EX document [hello.tex](#). After the Markdown package for our T_EX format has been loaded, the converted document can be typeset as follows:

```
\input hello
```

2.2 Plain T_EX Interface

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

```

856 \def\markdownLastModified{((LASTMODIFIED))}%
857 \def\markdownVersion{((VERSION))}%

```

The plain T_EX interface is implemented by the [markdown.tex](#) file that can be loaded as follows:

```
\input markdown
```

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

2.2.1 Typesetting Markdown 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.

```
858 \let\markdownBegin\relax
859 \let\markdownEnd\relax
```

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

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

Another limitation concerns spaces at the right end of an input line. In markdown, these are used to produce a forced line break. However, any such spaces are removed before the lines enter the input buffer of T_EX [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_EX code, the characters `c`, `e`, and `f` will not appear in the output.

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

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

The following example plain T_EX code showcases the usage of the `\markdownBegin` and `\markdownEnd` macros:

```
\input markdown
\markdownBegin
```

```
_Hello_ world ...  
\markdownEnd  
\bye
```

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

```
860 \let\uyamlBegin\u relax  
861 \def\uyamlEnd{\u markdownEnd\u endgroup}
```

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

The following example plain T_EX code showcases the usage of the `\markdownBegin` and `\markdownEnd` macros:

```
\input markdown  
\yamlBegin  
title: _Hello_ world ...  
author: John Doe  
\yamlEnd  
\bye
```

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

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

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

```
862 \let\u markinline\u relax
```

The following example plain T_EX 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:

```

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.

```
863 \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:

```



```

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.

```

864 \def\yamlInput#1{%
865   \begingroup
866   \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
867   \markdownInput{#1}%
868   \endgroup
869 }%

```

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.

```
870 \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.

```
871 \ExplSyntaxOn
872 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
873 \cs_generate_variant:Nn
874   \tl_const:Nn
875   { NV }
876 \tl_if_exist:NF
877   \c_@@_top_layer_tl
878   {
879     \tl_const:NV
880     \c_@@_top_layer_tl
881     \c_@@_option_layer_plain_tex_tl
882   }
```

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

```
883 \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.

```

884 \prop_new:N \g_@@_plain_tex_option_types_prop
885 \prop_new:N \g_@@_default_plain_tex_options_prop
886 \seq_gput_right:NV
887   \g_@@_option_layers_seq
888   \c_@@_option_layer_plain_tex_tl
889 \cs_new:Nn
890   \@@_add_plain_tex_option:nnn
891   {
892     \@@_add_option:Vnnn
893     \c_@@_option_layer_plain_tex_tl
894     { #1 }
895     { #2 }
896     { #3 }
897   }

```

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.

```

898 \cs_new:Nn
899   \@@_setup:n
900   {
901     \keys_set:nn
902     { markdown/options }
903     { #1 }
904   }
905 \cs_gset_eq:NN
906   \markdownSetup
907   \@@_setup:n

```

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

```

908 \cs_gset_eq:NN
909   \yamlSetup
910   \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.

```

911 \prg_new_conditional:Nnn
912   \@@_if_option:n
913   { TF, T, F }
914   {

```

```

915 \@@_get_option_type:nN
916   { #1 }
917   \l_tmpa_tl
918 \str_if_eq:NNF
919   \l_tmpa_tl
920   \c_@@_option_type_boolean_tl
921   {
922     \msg_error:nxxx
923     { markdown }
924     { expected-boolean-option }
925     { #1 }
926     { \l_tmpa_tl }
927   }
928 \@@_get_option_value:nN
929   { #1 }
930   \l_tmpa_tl
931 \str_if_eq:NNTF
932   \l_tmpa_tl
933   \c_@@_option_value_true_tl
934   { \prg_return_true: }
935   { \prg_return_false: }
936 }
937 \msg_new:nnn
938 { markdown }
939 { expected-boolean-option }
940 {
941   Option~#1~has~type~#2,~
942   but~a~boolean~was~expected.
943 }
944 \let
945 \markdownIfOption
946 \@@_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 \TeX document and their auxiliary files cached in the `cacheDir` directory.

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

```

947 \@@_add_plain_tex_option:nnn
948   { frozenCache }

```

```

949 { boolean }
950 { false }

```

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

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

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

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

```

951 \@@_add_plain_tex_option:nnn
952 { inputTempFileName }
953 { path }
954 { \jobname.markdown.in }

```

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

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

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

```

955 \@@_add_plain_tex_option:nnn
956 { outputDir }
957 { path }
958 { . }

```

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 \TeX formats such as \LaTeX and \ConTeXt . 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_EX formats should only use the plain T_EX default definitions or whether they should also use the format-specific default definitions. Whereas plain T_EX 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_EX 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^AT_EX document:

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

Here is how you would enable the macro in a ConT_EXt 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.

```
959 \@@_add_plain_tex_option:nnn  
960   { plain }  
961   { boolean }  
962   { 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^AT_EX document:

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

Here is how you would enable the macro in a ConT_EXt 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.

```
963 \@@_add_plain_tex_option:nnn  
964   { noDefaults }  
965   { boolean }  
966   { 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_EX package documentation using the Doc L^AT_EX package [9] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```
967 \seq_gput_right:Nn
968   \g_@@_plain_tex_options_seq
969   { stripPercentSigns }
970 \prop_gput:Nnn
971   \g_@@_plain_tex_option_types_prop
972   { stripPercentSigns }
973   { boolean }
974 \prop_gput:Nnx
975   \g_@@_default_plain_tex_options_prop
976   { stripPercentSigns }
977   { false }
```

2.2.2.5 Generating Plain T_EX Option Macros and Key-Values

We define the command `\@@_define_option_commands_and_keyvals:` that defines plain T_EX macros and the key–value interface of the `\markdownSetup` macro for the above plain T_EX 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_EX 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_EX formats such as L^AT_EX and ConT_EXt.

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.

```
978 \cs_new:Nn
979   \@@_define_option_commands_and_keyvals:
980   {
981     \seq_map_inline:Nn
982       \g_@@_option_layers_seq
983       {
984         \seq_map_inline:cn
985           { g_@@_ ##1 _options_seq }
986           {
987             \@@_define_option_command:n
988               { ####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.

```

989         \@@_with_various_cases:nn
990         { #####1 }
991         {
992             \@@_define_option_keyval:nnn
993             { ##1 }
994             { #####1 }
995             { #####1 }
996         }
997     }
998 }
999 }
1000 \cs_new:Nn
1001 \@@_define_option_command:n
1002 {

```

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

```

1003 \str_if_eq:nnTF
1004 { #1 }
1005 { outputDir }
1006 { \@@_define_option_command_output_dir: }
1007 {

```

Do not override options defined before loading the package.

```

1008 \@@_option_tl_to_csname:nN
1009 { #1 }
1010 \l_tmpa_tl
1011 \cs_if_exist:cF
1012 { \l_tmpa_tl }
1013 {
1014     \@@_get_default_option_value:nN
1015     { #1 }
1016     \l_tmpa_tl
1017     \@@_set_option_value:nV
1018     { #1 }
1019     \l_tmpa_tl
1020 }
1021 }
1022 }
1023 \ExplSyntaxOff
1024 \input lt3luabridge.tex

```

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

```

1025 \ExplSyntaxOn
1026 \cs_new:Nn
1027   \@@_define_option_command_output_dir:
1028   {
1029     \cs_if_free:NT
1030       \markdownOptionOutputDir
1031       {
1032         \bool_if:nTF
1033           {
1034             \cs_if_exist_p:N
1035               \luabridge_tl_set:Nn &&
1036             (
1037               \int_compare_p:nNn
1038                 { \g_luabridge_method_int }
1039                 =
1040                 { \c_luabridge_method_directlua_int } ||
1041               \sys_if_shell_unrestricted_p:
1042             )
1043           }
1044           {

```

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

```

1045         \group_begin:
1046         \cctab_select:N
1047         \c_str_cctab
1048         \luabridge_tl_set:Nn
1049         \l_tmpa_tl
1050         { print(os.getenv("TEXMF_OUTPUT_DIRECTORY") or ".") }
1051         \tl_gset:NV
1052         \markdownOptionOutputDir
1053         \l_tmpa_tl
1054         \group_end:
1055       }
1056     {
1057       \tl_gset:Nn
1058         \markdownOptionOutputDir
1059         { . }
1060     }
1061   }
1062 }
1063 \cs_new:Nn
1064   \@@_set_option_value:nn

```



```

1065 {
1066   \@@_define_option:n
1067   { #1 }
1068   \@@_get_option_type:nN
1069   { #1 }
1070   \l_tmpa_tl
1071   \str_if_eq:NNTF
1072   \c_@@_option_type_counter_tl
1073   \l_tmpa_tl
1074   {
1075     \@@_option_tl_to_csname:nN
1076     { #1 }
1077     \l_tmpa_tl
1078     \int_gset:cn
1079     { \l_tmpa_tl }
1080     { #2 }
1081   }
1082   {
1083     \@@_option_tl_to_csname:nN
1084     { #1 }
1085     \l_tmpa_tl
1086     \cs_set:cpn
1087     { \l_tmpa_tl }
1088     { #2 }
1089   }
1090 }
1091 \cs_generate_variant:Nn
1092 \@@_set_option_value:nm
1093 { nV }
1094 \cs_new:Nn
1095 \@@_define_option:n
1096 {
1097   \@@_option_tl_to_csname:nN
1098   { #1 }
1099   \l_tmpa_tl
1100   \cs_if_free:cT
1101   { \l_tmpa_tl }
1102   {
1103     \@@_get_option_type:nN
1104     { #1 }
1105     \l_tmpb_tl
1106     \str_if_eq:NNT
1107     \c_@@_option_type_counter_tl
1108     \l_tmpb_tl
1109     {
1110       \@@_option_tl_to_csname:nN
1111       { #1 }

```

```

1112         \l_tmpa_tl
1113     \int_new:c
1114         { \l_tmpa_tl }
1115     }
1116 }
1117 }
1118 \cs_new:Nn
1119 \@@_define_option_keyval:nnn
1120 {
1121     \prop_get:cnN
1122     { g_@@_ #1 _option_types_prop }
1123     { #2 }
1124     \l_tmpa_tl
1125     \str_if_eq:VVTF
1126     \l_tmpa_tl
1127     \c_@@_option_type_boolean_tl
1128     {
1129         \keys_define:nn
1130         { markdown/options }
1131         {

```

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

```

1132         #3 .code:n = {
1133             \tl_set:Nx
1134             \l_tmpa_tl
1135             {
1136                 \str_case:nnF
1137                 { ##1 }
1138                 {
1139                     { yes } { true }
1140                     { no } { false }
1141                 }
1142                 { ##1 }
1143             }
1144             \@@_set_option_value:nV
1145             { #2 }
1146             \l_tmpa_tl
1147         },
1148         #3 .default:n = { true },
1149     }
1150 }
1151 {
1152     \keys_define:nn
1153     { markdown/options }
1154     {
1155         #3 .code:n = {

```

```

1156             \@@_set_option_value:nn
1157             { #2 }
1158             { ##1 }
1159         },
1160     }
1161 }

```

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.

```

1162     \str_if_eq:VVT
1163     \l_tmpa_tl
1164     \c_@@_option_type_clist_tl
1165     {
1166         \tl_set:Nn
1167         \l_tmpa_tl
1168         { #3 }
1169         \tl_reverse:N
1170         \l_tmpa_tl
1171         \str_if_eq:enF
1172         {
1173             \tl_head:V
1174             \l_tmpa_tl
1175         }
1176         { s }
1177         {
1178             \msg_error:nnn
1179             { markdown }
1180             { malformed-name-for-clist-option }
1181             { #3 }
1182         }
1183         \tl_set:Nx
1184         \l_tmpa_tl
1185         {
1186             \tl_tail:V
1187             \l_tmpa_tl
1188         }
1189         \tl_reverse:N
1190         \l_tmpa_tl
1191         \tl_put_right:Nn
1192         \l_tmpa_tl
1193         {
1194             .code:n = {
1195                 \@@_get_option_value:nN
1196                 { #2 }

```

```

1197         \l_tmpa_tl
1198     \clist_set:NV
1199         \l_tmpa_clist
1200         { \l_tmpa_tl , { ##1 } }
1201     \@@_set_option_value:nV
1202         { #2 }
1203         \l_tmpa_clist
1204     }
1205 }
1206 \keys_define:nV
1207     { markdown/options }
1208     \l_tmpa_tl
1209 }
1210 }
1211 \cs_generate_variant:Nn
1212     \clist_set:Nn
1213     { NV }
1214 \cs_generate_variant:Nn
1215     \keys_define:nn
1216     { nV }
1217 \cs_generate_variant:Nn
1218     \@@_set_option_value:nn
1219     { nV }
1220 \prg_generate_conditional_variant:Nnn
1221     \str_if_eq:nn
1222     { en }
1223     { p, F }
1224 \msg_new:nnn
1225     { markdown }
1226     { malformed-name-for-clist-option }
1227     {
1228         Clist-option-name~#1~does~not~end~with~--s.
1229     }

```

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

```

1230 \str_if_eq:VVT
1231     \c_@@_top_layer_tl
1232     \c_@@_option_layer_plain_tex_tl
1233     {
1234         \@@_define_option_commands_and_keyvals:
1235     }
1236 \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 T_EX 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 L^AT_EX 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 T_EX directory structure. For example, loading a theme named `witiko/beamer/MU` would load a T_EX 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].

```
1237 \ExplSyntaxOn
1238 \keys_define:nn
1239   { markdown/options }
1240   {
1241     theme .code:n = {
1242       \@@_set_theme:n
1243         { #1 }
1244     },
1245     import .code:n = {
1246       \tl_set:Nn
1247         \l_tmpa_tl
1248         { #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.

```

1249     \tl_replace_all:NnV
1250     \l_tmpa_tl
1251     { / }
1252     \c_backslash_str
1253     \keys_set:nV
1254     { markdown/options/import }
1255     \l_tmpa_tl
1256   },
1257 }

```

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.

```

1258 \seq_new:N
1259   \g_@@_theme_names_seq
1260 \seq_new:N
1261   \g_@@_theme_versions_seq
1262 \tl_new:N
1263   \g_@@_current_theme_tl
1264 \tl_gset:Nn
1265   \g_@@_current_theme_tl
1266   { }
1267 \seq_gput_right:NV
1268   \g_@@_theme_names_seq
1269   \g_@@_current_theme_tl
1270 \cs_new:Npn
1271   \markdownThemeVersion
1272   { }
1273 \seq_gput_right:NV
1274   \g_@@_theme_versions_seq
1275   \g_@@_current_theme_tl
1276 \cs_new:Nn
1277   \@@_set_theme:n
1278   {

```

First, we validate the theme name.

```

1279   \str_if_in:nnF
1280     { #1 }
1281     { / }
1282     {
1283       \msg_error:nnn
1284         { markdown }
1285         { unqualified-theme-name }
1286         { #1 }
1287     }

```

```

1288   \str_if_in:nnT
1289     { #1 }
1290     { _ }
1291     {
1292       \msg_error:nnn
1293         { markdown }
1294         { underscores-in-theme-name }
1295         { #1 }
1296     }

```

Next, we extract the theme version.

```

1297   \str_if_in:nnTF
1298     { #1 }
1299     { @ }
1300     {
1301       \regex_extract_once:nnN
1302         { (.*?) @ (.*?) }
1303         { #1 }
1304         \l_tmpa_seq
1305       \seq_gpop_left:NN
1306         \l_tmpa_seq
1307         \l_tmpa_tl
1308       \seq_gpop_left:NN
1309         \l_tmpa_seq
1310         \l_tmpa_tl
1311       \tl_gset:NV
1312         \g_@@_current_theme_tl
1313         \l_tmpa_tl
1314       \seq_gpop_left:NN
1315         \l_tmpa_seq
1316         \l_tmpa_tl
1317       \cs_gset:Npe
1318         \markdownThemeVersion
1319         {
1320           \tl_use:N
1321             \l_tmpa_tl
1322         }
1323     }
1324     {
1325       \tl_gset:Nn
1326         \g_@@_current_theme_tl
1327         { #1 }
1328       \cs_gset:Npn
1329         \markdownThemeVersion
1330         { latest }
1331     }

```

Next, we munge the theme name.

```

1332 \str_set:NV
1333 \l_tmpa_str
1334 \g_@@_current_theme_tl
1335 \str_replace_all:Nnn
1336 \l_tmpa_str
1337 { / }
1338 { _ }

```

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

```

1339 \tl_set:NV
1340 \l_tmpa_tl
1341 \g_@@_current_theme_tl
1342 \tl_put_right:Nn
1343 \g_@@_current_theme_tl
1344 { / }
1345 \seq_gput_right:NV
1346 \g_@@_theme_names_seq
1347 \g_@@_current_theme_tl
1348 \seq_gput_right:NV
1349 \g_@@_theme_versions_seq
1350 \markdownThemeVersion
1351 \@@_load_theme:VeV
1352 \l_tmpa_tl
1353 { \markdownThemeVersion }
1354 \l_tmpa_str

```

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

```

1355 \seq_gpop_right:NN
1356 \g_@@_theme_names_seq
1357 \l_tmpa_tl
1358 \seq_get_right:NN
1359 \g_@@_theme_names_seq
1360 \l_tmpa_tl
1361 \tl_gset:NV
1362 \g_@@_current_theme_tl
1363 \l_tmpa_tl
1364 \seq_gpop_right:NN
1365 \g_@@_theme_versions_seq
1366 \l_tmpa_tl
1367 \seq_get_right:NN
1368 \g_@@_theme_versions_seq
1369 \l_tmpa_tl
1370 \cs_gset:Npe
1371 \markdownThemeVersion
1372 {
1373 \tl_use:N

```



```

1374         \l_tmpa_tl
1375     }
1376 }
1377 \msg_new:nnnn
1378 { markdown }
1379 { unqualified-theme-name }
1380 { Won't-load-theme-with-unqualified-name-#1 }
1381 { Theme-names-must-contain-at-least-one-forward-slash }
1382 \msg_new:nnnn
1383 { markdown }
1384 { underscores-in-theme-name }
1385 { Won't-load-theme-with-an-underscore-in-its-name-#1 }
1386 { Theme-names-must-not-contain-underscores-in-their-names }
1387 \cs_generate_variant:Nn
1388   \tl_replace_all:Nnn
1389   { NnV }
1390 \cs_generate_variant:Nn
1391   \cs_gset:Npn
1392   { 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.

```

1393 \prop_new:N
1394   \g_@@_plain_tex_built_in_themes_prop

```

Built-in plain T_EX 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
idea 1
 ((?))
 sub
idea 2
 ((?))
 sub
idea 3
 ((?))
 sub
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

```

```

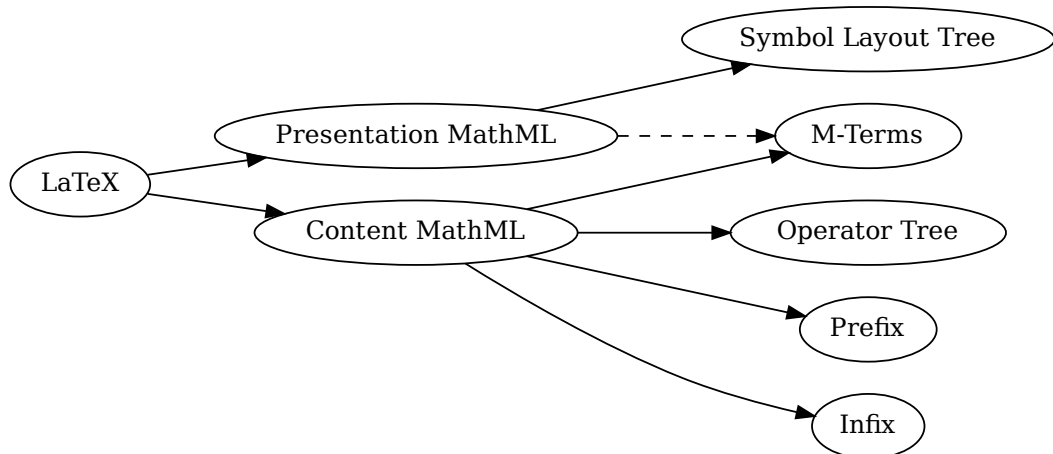
' Messages
C -> S: Send Request
note over S: Process request

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
end
@enduml
...

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

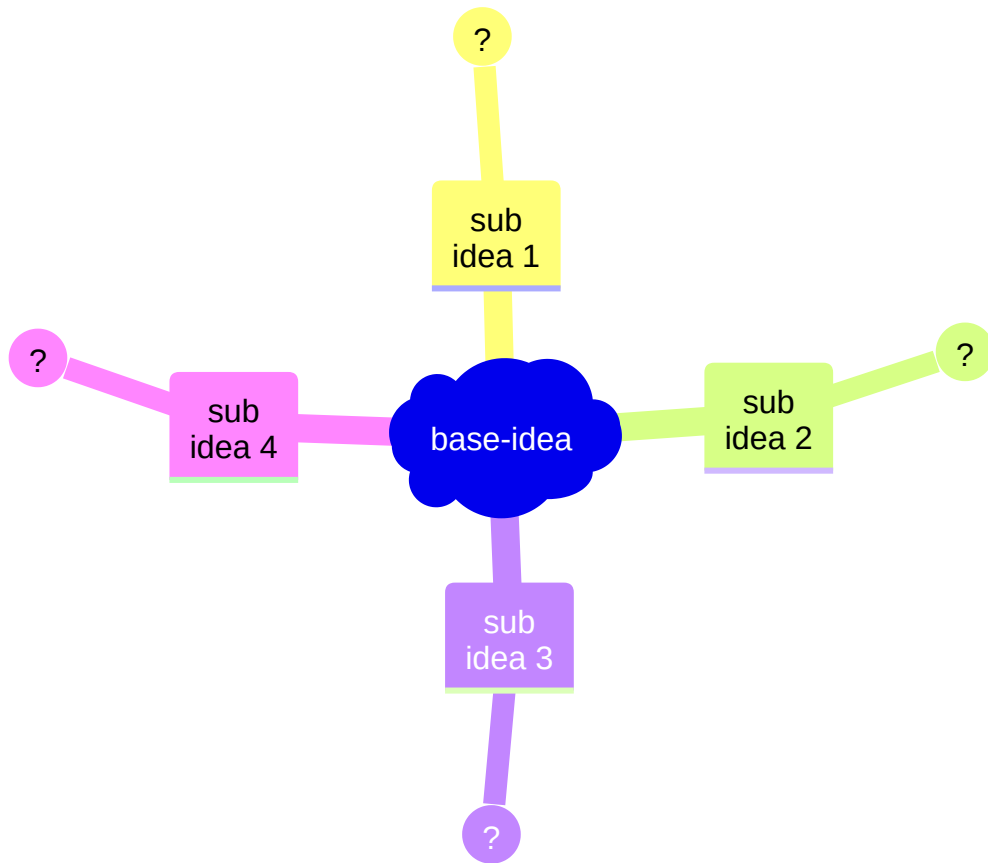
```

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



**Figure 4: An example directed graph**

The theme requires a Unix-like operating system with GNU Diffutils, Graphviz, the npm package [@mermaid-js/mermaid-cli](#), and PlantUML installed. All these packages are already included in the Docker image [witiko/markdown](#);



**Figure 5: An example mindmap**

## Simple Request-Response Flow

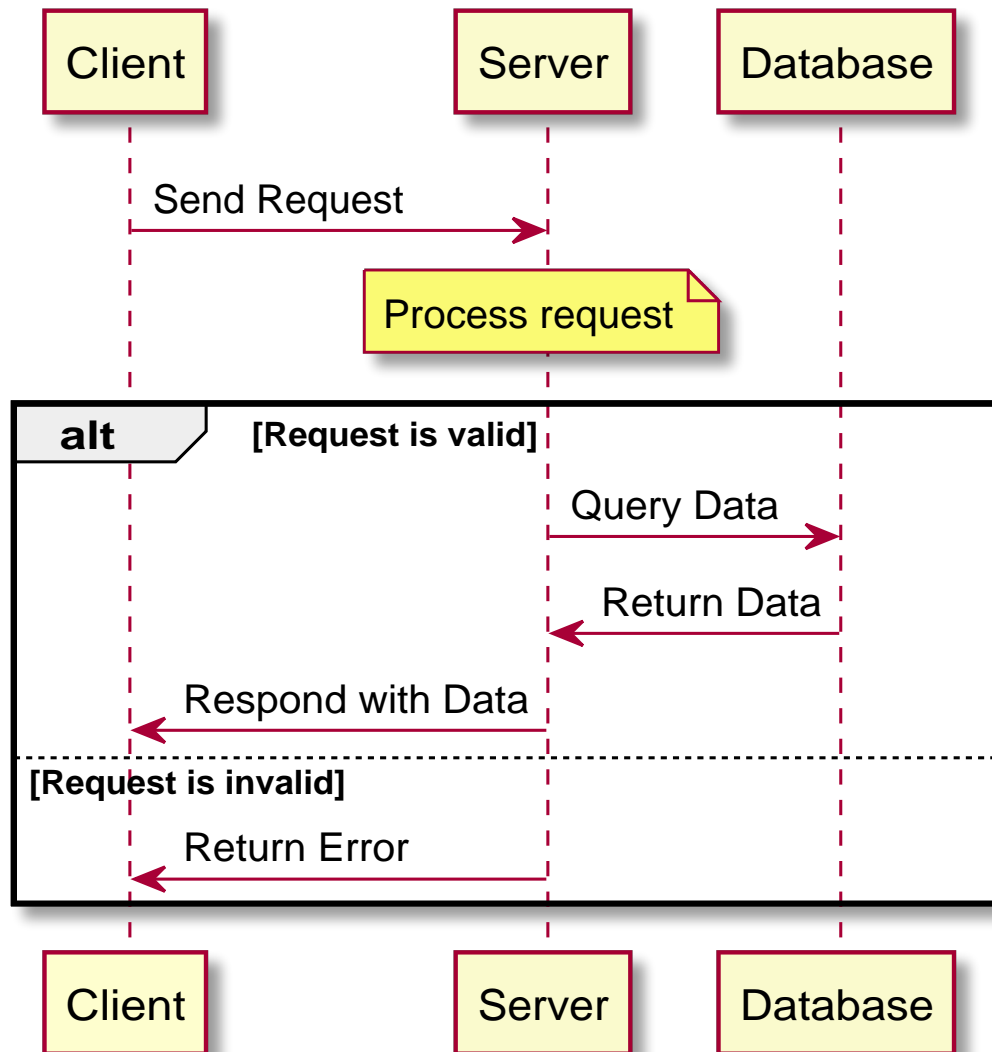


Figure 6: An example UML sequence diagram

consult `Dockerfile` to see how they are installed. The theme also requires shell access unless the `frozenCache` plain  $\TeX$  option is enabled.

**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}


\end{markdown}
\end{document}
```

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

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

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

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



## Chapter 1

### Introduction

#### 1.1 Section

##### 1.1.1 Subsection

Hello *Markdown*!

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

Table 1.1: Table

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

theme requires the catchfile  $\LaTeX$  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
\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.

```
1395 \prop_new:N
1396 \g_@@_snippets_prop
1397 \cs_new:Nn
1398 \@@_setup_snippet:nn
1399 {
1400 \tl_if_empty:nT
1401 { #1 }
1402 {
1403 \msg_error:nnn
1404 { markdown }
1405 { empty-snippet-name }
1406 { #1 }
1407 }
1408 \tl_set:NV
1409 \l_tmpa_tl
1410 \g_@@_current_theme_tl
1411 \tl_put_right:Nn
1412 \l_tmpa_tl
1413 { #1 }
1414 \@@_if_snippet_exists:nT
```

```

1415 { #1 }
1416 {
1417 \msg_warning:nnV
1418 { markdown }
1419 { redefined-snippet }
1420 \l_tmpa_tl
1421 }
1422 \keys_precompile:nnN
1423 { markdown/options }
1424 { #2 }
1425 \l_tmpb_tl
1426 \prop_gput:NVV
1427 \g_@@_snippets_prop
1428 \l_tmpa_tl
1429 \l_tmpb_tl
1430 }
1431 \cs_gset_eq:NN
1432 \markdownSetupSnippet
1433 \@@_setup_snippet:nn
1434 \msg_new:nnnn
1435 { markdown }
1436 { empty-snippet-name }
1437 { Empty-snippet-name~#1 }
1438 { Pick-a~non-empty~name~for~your~snippet }
1439 \msg_new:nnn
1440 { markdown }
1441 { redefined-snippet }
1442 { Redefined~snippet~#1 }

```

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

```

1443 \tl_new:N
1444 \l_@@_current_snippet_tl
1445 \prg_new_conditional:Nnn
1446 \@@_if_snippet_exists:n
1447 { TF, T, F }
1448 {
1449 \tl_set:NV
1450 \l_@@_current_snippet_tl
1451 \g_@@_current_theme_tl
1452 \tl_put_right:Nn
1453 \l_@@_current_snippet_tl
1454 { #1 }
1455 \prop_if_in:NVTF
1456 \g_@@_snippets_prop
1457 \l_@@_current_snippet_tl
1458 { \prg_return_true: }
1459 { \prg_return_false: }

```



```

1460 }
1461 \cs_gset_eq:NN
1462 \markdownIfSnippetExists
1463 \@@_if_snippet_exists:nTF

```

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

```

1464 \keys_define:nn
1465 { markdown/options }
1466 {
1467 snippet .code:n = {
1468 \tl_set:NV
1469 \l_tmpa_tl
1470 \g_@@_current_theme_tl
1471 \tl_put_right:Nn
1472 \l_tmpa_tl
1473 { #1 }
1474 \@@_if_snippet_exists:nTF
1475 { #1 }
1476 {
1477 \prop_get:NVN
1478 \g_@@_snippets_prop
1479 \l_tmpa_tl
1480 \l_tmpb_tl
1481 \tl_use:N
1482 \l_tmpb_tl
1483 }
1484 {
1485 \msg_error:nnV
1486 { markdown }
1487 { undefined-snippet }
1488 \l_tmpa_tl
1489 }
1490 }
1491 }
1492 \msg_new:nnn
1493 { markdown }
1494 { undefined-snippet }
1495 { Can't~invoke~undefined~snippet~#1 }
1496 \ExplSyntaxOff

```

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

```

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

```

The following ordered list will be preceded by arabic numerals:

1. wahid
2. aithnayn

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

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```
\end{markdown}
```

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

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

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```
\end{markdown}
```

Alternatively, we can use the extended variant of the `import`  $\LaTeX$  option that allows us to import the `romanNumerals` snippet to the current namespace for easier access:

```
\markdownSetup{
 import = {
 jdoue/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 `jdooe/lists` theme. For example, we can make the snippet `jdooe/lists/romanNumerals` available under the name `roman`.

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

The following ordered list will be preceded by roman numerals:

```
3. tres
4. quattuor

\end{markdown}
```

Several themes and/or snippets can be loaded at once using the extended variant of the `import`  $\LaTeX$  option:

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

```

1497 \ExplSyntaxOn
1498 \tl_new:N
1499 \l_@@_import_current_theme_tl
1500 \keys_define:nn
1501 { markdown/options/import }
1502 {

```

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

```

1503 unknown .default:n = {},
1504 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.

```

1505 \tl_set_eq:NN
1506 \l_@@_import_current_theme_tl
1507 \l_keys_key_str
1508 \tl_replace_all:NVN
1509 \l_@@_import_current_theme_tl
1510 \c_backslash_str
1511 { / }

```

Here, we import the snippets.

```

1512 \clist_map_inline:nn
1513 { #1 }
1514 {
1515 \regex_extract_once:nnNTF
1516 { ^(.*)\s+as\s+(.*)$ }
1517 { ##1 }
1518 \l_tmpa_seq
1519 {
1520 \seq_pop:NN
1521 \l_tmpa_seq
1522 \l_tmpa_tl
1523 \seq_pop:NN
1524 \l_tmpa_seq
1525 \l_tmpa_tl
1526 \seq_pop:NN
1527 \l_tmpa_seq
1528 \l_tmpb_tl
1529 }
1530 }
1531 \tl_set:Nn
1532 \l_tmpa_tl

```

```

1533 { ##1 }
1534 \tl_set:Nn
1535 \l_tmpb_tl
1536 { ##1 }
1537 }
1538 \tl_put_left:Nn
1539 \l_tmpa_tl
1540 { / }
1541 \tl_put_left:NV
1542 \l_tmpa_tl
1543 \l_@@_import_current_theme_tl
1544 \@@_setup_snippet:Vx
1545 \l_tmpb_tl
1546 { snippet = { \l_tmpa_tl } }
1547 }

```

Here, we load the theme.

```

1548 \@@_set_theme:V
1549 \l_@@_import_current_theme_tl
1550 },
1551 }
1552 \cs_generate_variant:Nn
1553 \tl_replace_all:Nnn
1554 { NVn }
1555 \cs_generate_variant:Nn
1556 \@@_set_theme:n
1557 { V }
1558 \cs_generate_variant:Nn
1559 \@@_setup_snippet:nn
1560 { Vx }

```

## 2.2.5 Token Renderers

The following T<sub>E</sub>X 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.

```

1561 \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.

```

1562 \prop_new:N \g_@@_renderer_arities_prop
1563 \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.

```

1564 \ExplSyntaxOn
1565 \cs_gset_protected:Npn
1566 \markdownRendererAttributeIdentifier
1567 {
1568 \markdownRendererAttributeIdentifierPrototype
1569 }
1570 \seq_gput_right:Nn
1571 \g_@@_renderers_seq
1572 { attributeIdentifier }
1573 \prop_gput:Nnn
1574 \g_@@_renderer_arities_prop
1575 { attributeIdentifier }
1576 { 1 }
1577 \cs_gset_protected:Npn
1578 \markdownRendererAttributeClassName
1579 {
1580 \markdownRendererAttributeClassNamePrototype
1581 }
1582 \seq_gput_right:Nn
1583 \g_@@_renderers_seq
1584 { attributeClassName }
1585 \prop_gput:Nnn
1586 \g_@@_renderer_arities_prop
1587 { attributeClassName }
1588 { 1 }

```

```

1589 \cs_gset_protected:Npn
1590 \markdownRendererAttributeKeyValue
1591 {
1592 \markdownRendererAttributeKeyValuePrototype
1593 }
1594 \seq_gput_right:Nn
1595 \g_@@_renderers_seq
1596 { attributeKeyValue }
1597 \prop_gput:Nnn
1598 \g_@@_renderer_arities_prop
1599 { attributeKeyValue }
1600 { 2 }
1601 \ExplSyntaxOff

```

### 2.2.5.2 Block Quote Renderers

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

```

1602 \ExplSyntaxOn
1603 \cs_gset_protected:Npn
1604 \markdownRendererBlockQuoteBegin
1605 {
1606 \markdownRendererBlockQuoteBeginPrototype
1607 }
1608 \seq_gput_right:Nn
1609 \g_@@_renderers_seq
1610 { blockQuoteBegin }
1611 \prop_gput:Nnn
1612 \g_@@_renderer_arities_prop
1613 { blockQuoteBegin }
1614 { 0 }
1615 \ExplSyntaxOff

```

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

```

1616 \ExplSyntaxOn
1617 \cs_gset_protected:Npn
1618 \markdownRendererBlockQuoteEnd
1619 {
1620 \markdownRendererBlockQuoteEndPrototype
1621 }
1622 \seq_gput_right:Nn
1623 \g_@@_renderers_seq
1624 { blockQuoteEnd }
1625 \prop_gput:Nnn
1626 \g_@@_renderer_arities_prop
1627 { blockQuoteEnd }

```

```

1628 { 0 }
1629 \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.

```

1630 \ExplSyntaxOn
1631 \cs_gset_protected:Npn
1632 \markdownRendererBracketedSpanAttributeContextBegin
1633 {
1634 \markdownRendererBracketedSpanAttributeContextBeginPrototype
1635 }
1636 \seq_gput_right:Nn
1637 \g_@@_renderers_seq
1638 { bracketedSpanAttributeContextBegin }
1639 \prop_gput:Nnn
1640 \g_@@_renderer_arities_prop
1641 { bracketedSpanAttributeContextBegin }
1642 { 0 }
1643 \cs_gset_protected:Npn
1644 \markdownRendererBracketedSpanAttributeContextEnd
1645 {
1646 \markdownRendererBracketedSpanAttributeContextEndPrototype
1647 }
1648 \seq_gput_right:Nn
1649 \g_@@_renderers_seq
1650 { bracketedSpanAttributeContextEnd }
1651 \prop_gput:Nnn
1652 \g_@@_renderer_arities_prop
1653 { bracketedSpanAttributeContextEnd }
1654 { 0 }
1655 \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.

```

1656 \ExplSyntaxOn
1657 \cs_gset_protected:Npn
1658 \markdownRendererUlBegin
1659 {

```



```

1660 \markdownRendererUlBeginPrototype
1661 }
1662 \seq_gput_right:Nn
1663 \g_@@_renderers_seq
1664 { ulBegin }
1665 \prop_gput:Nnn
1666 \g_@@_renderer_arities_prop
1667 { ulBegin }
1668 { 0 }
1669 \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.

```

1670 \ExplSyntaxOn
1671 \cs_gset_protected:Npn
1672 \markdownRendererUlBeginTight
1673 {
1674 \markdownRendererUlBeginTightPrototype
1675 }
1676 \seq_gput_right:Nn
1677 \g_@@_renderers_seq
1678 { ulBeginTight }
1679 \prop_gput:Nnn
1680 \g_@@_renderer_arities_prop
1681 { ulBeginTight }
1682 { 0 }
1683 \ExplSyntaxOff

```

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

```

1684 \ExplSyntaxOn
1685 \cs_gset_protected:Npn
1686 \markdownRendererUlItem
1687 {
1688 \markdownRendererUlItemPrototype
1689 }
1690 \seq_gput_right:Nn
1691 \g_@@_renderers_seq
1692 { ulItem }
1693 \prop_gput:Nnn
1694 \g_@@_renderer_arities_prop
1695 { ulItem }
1696 { 0 }
1697 \ExplSyntaxOff

```

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

```
1698 \ExplSyntaxOn
1699 \cs_gset_protected:Npn
1700 \markdownRendererUListItemEnd
1701 {
1702 \markdownRendererUListItemEndPrototype
1703 }
1704 \seq_gput_right:Nn
1705 \g_@@_renderers_seq
1706 { ulItemEnd }
1707 \prop_gput:Nnn
1708 \g_@@_renderer_arities_prop
1709 { ulItemEnd }
1710 { 0 }
1711 \ExplSyntaxOff
```

The `\markdownRendererUListEnd` 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.

```
1712 \ExplSyntaxOn
1713 \cs_gset_protected:Npn
1714 \markdownRendererUListEnd
1715 {
1716 \markdownRendererUListEndPrototype
1717 }
1718 \seq_gput_right:Nn
1719 \g_@@_renderers_seq
1720 { ulEnd }
1721 \prop_gput:Nnn
1722 \g_@@_renderer_arities_prop
1723 { ulEnd }
1724 { 0 }
1725 \ExplSyntaxOff
```

The `\markdownRendererUListEndTight` 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.

```
1726 \ExplSyntaxOn
1727 \cs_gset_protected:Npn
1728 \markdownRendererUListEndTight
1729 {
1730 \markdownRendererUListEndTightPrototype
1731 }
1732 \seq_gput_right:Nn
```

```

1733 \g_@@_renderers_seq
1734 { ulEndTight }
1735 \prop_gput:Nnn
1736 \g_@@_renderer_arities_prop
1737 { ulEndTight }
1738 { 0 }
1739 \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.

```

1740 \ExplSyntaxOn
1741 \cs_gset_protected:Npn
1742 \markdownRendererCite
1743 {
1744 \markdownRendererCitePrototype
1745 }
1746 \seq_gput_right:Nn
1747 \g_@@_renderers_seq
1748 { cite }
1749 \prop_gput:Nnn
1750 \g_@@_renderer_arities_prop
1751 { cite }
1752 { 1 }
1753 \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.

```

1754 \ExplSyntaxOn
1755 \cs_gset_protected:Npn
1756 \markdownRendererTextCite
1757 {
1758 \markdownRendererTextCitePrototype
1759 }
1760 \seq_gput_right:Nn
1761 \g_@@_renderers_seq
1762 { textCite }
1763 \prop_gput:Nnn
1764 \g_@@_renderer_arities_prop

```

```

1765 { textCite }
1766 { 1 }
1767 \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.

```

1768 \ExplSyntaxOn
1769 \cs_gset_protected:Npn
1770 \markdownRendererInputVerbatim
1771 {
1772 \markdownRendererInputVerbatimPrototype
1773 }
1774 \seq_gput_right:Nn
1775 \g_@@_renderers_seq
1776 { inputVerbatim }
1777 \prop_gput:Nnn
1778 \g_@@_renderer_arities_prop
1779 { inputVerbatim }
1780 { 1 }
1781 \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.

```

1782 \ExplSyntaxOn
1783 \cs_gset_protected:Npn
1784 \markdownRendererInputFencedCode
1785 {
1786 \markdownRendererInputFencedCodePrototype
1787 }
1788 \seq_gput_right:Nn
1789 \g_@@_renderers_seq
1790 { inputFencedCode }
1791 \prop_gput:Nnn
1792 \g_@@_renderer_arities_prop
1793 { inputFencedCode }
1794 { 3 }
1795 \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.

```

1796 \ExplSyntaxOn
1797 \cs_gset_protected:Npn
1798 \markdownRendererCodeSpan
1799 {
1800 \markdownRendererCodeSpanPrototype
1801 }
1802 \seq_gput_right:Nn
1803 \g_@@_renderers_seq
1804 { codeSpan }
1805 \prop_gput:Nnn
1806 \g_@@_renderer_arities_prop
1807 { codeSpan }
1808 { 1 }
1809 \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.

```

1810 \ExplSyntaxOn
1811 \cs_gset_protected:Npn
1812 \markdownRendererCodeSpanAttributeContextBegin
1813 {
1814 \markdownRendererCodeSpanAttributeContextBeginPrototype
1815 }
1816 \seq_gput_right:Nn
1817 \g_@@_renderers_seq
1818 { codeSpanAttributeContextBegin }
1819 \prop_gput:Nnn
1820 \g_@@_renderer_arities_prop
1821 { codeSpanAttributeContextBegin }
1822 { 0 }
1823 \cs_gset_protected:Npn
1824 \markdownRendererCodeSpanAttributeContextEnd
1825 {
1826 \markdownRendererCodeSpanAttributeContextEndPrototype
1827 }
1828 \seq_gput_right:Nn
1829 \g_@@_renderers_seq
1830 { codeSpanAttributeContextEnd }
1831 \prop_gput:Nnn

```

```

1832 \g_@@_renderer_arities_prop
1833 { codeSpanAttributeContextEnd }
1834 { 0 }
1835 \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.

```

1836 \ExplSyntaxOn
1837 \cs_gset_protected:Npn
1838 \markdownRendererContentBlock
1839 {
1840 \markdownRendererContentBlockPrototype
1841 }
1842 \seq_gput_right:Nn
1843 \g_@@_renderers_seq
1844 { contentBlock }
1845 \prop_gput:Nnn
1846 \g_@@_renderer_arities_prop
1847 { contentBlock }
1848 { 4 }
1849 \ExplSyntaxOff

```

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

```

1850 \ExplSyntaxOn
1851 \cs_gset_protected:Npn
1852 \markdownRendererContentBlockOnlineImage
1853 {
1854 \markdownRendererContentBlockOnlineImagePrototype
1855 }
1856 \seq_gput_right:Nn
1857 \g_@@_renderers_seq
1858 { contentBlockOnlineImage }
1859 \prop_gput:Nnn
1860 \g_@@_renderer_arities_prop
1861 { contentBlockOnlineImage }
1862 { 4 }
1863 \ExplSyntaxOff

```

The `\markdownRendererContentBlockCode` macro represents an iA Writer content block that was recognized as a file in a known programming language by its

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

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

```

1864 \ExplSyntaxOn
1865 \cs_gset_protected:Npn
1866 \markdownRendererContentBlockCode
1867 {
1868 \markdownRendererContentBlockCodePrototype
1869 }
1870 \seq_gput_right:Nn
1871 \g_@@_renderers_seq
1872 { contentBlockCode }
1873 \prop_gput:Nnn
1874 \g_@@_renderer_arities_prop
1875 { contentBlockCode }
1876 { 5 }
1877 \ExplSyntaxOff

```

### 2.2.5.10 Definition List Renderers

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

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

```

1878 \ExplSyntaxOn
1879 \cs_gset_protected:Npn
1880 \markdownRendererDlBegin
1881 {
1882 \markdownRendererDlBeginPrototype
1883 }
1884 \seq_gput_right:Nn
1885 \g_@@_renderers_seq

```

---

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

```

1886 { dlBegin }
1887 \prop_gput:Nnn
1888 \g_@@_renderer_arities_prop
1889 { dlBegin }
1890 { 0 }
1891 \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.

```

1892 \ExplSyntaxOn
1893 \cs_gset_protected:Npn
1894 \markdownRendererDlBeginTight
1895 {
1896 \markdownRendererDlBeginTightPrototype
1897 }
1898 \seq_gput_right:Nn
1899 \g_@@_renderers_seq
1900 { dlBeginTight }
1901 \prop_gput:Nnn
1902 \g_@@_renderer_arities_prop
1903 { dlBeginTight }
1904 { 0 }
1905 \ExplSyntaxOff

```

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

```

1906 \ExplSyntaxOn
1907 \cs_gset_protected:Npn
1908 \markdownRendererDlItem
1909 {
1910 \markdownRendererDlItemPrototype
1911 }
1912 \seq_gput_right:Nn
1913 \g_@@_renderers_seq
1914 { dlItem }
1915 \prop_gput:Nnn
1916 \g_@@_renderer_arities_prop
1917 { dlItem }
1918 { 1 }
1919 \ExplSyntaxOff

```

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

```

1920 \ExplSyntaxOn

```



```

1921 \cs_gset_protected:Npn
1922 \markdownRendererDlItemEnd
1923 {
1924 \markdownRendererDlItemEndPrototype
1925 }
1926 \seq_gput_right:Nn
1927 \g_@@_renderers_seq
1928 { dlItemEnd }
1929 \prop_gput:Nnn
1930 \g_@@_renderer_arities_prop
1931 { dlItemEnd }
1932 { 0 }
1933 \ExplSyntaxOff

```

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

```

1934 \ExplSyntaxOn
1935 \cs_gset_protected:Npn
1936 \markdownRendererDlDefinitionBegin
1937 {
1938 \markdownRendererDlDefinitionBeginPrototype
1939 }
1940 \seq_gput_right:Nn
1941 \g_@@_renderers_seq
1942 { dlDefinitionBegin }
1943 \prop_gput:Nnn
1944 \g_@@_renderer_arities_prop
1945 { dlDefinitionBegin }
1946 { 0 }
1947 \ExplSyntaxOff

```

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

```

1948 \ExplSyntaxOn
1949 \cs_gset_protected:Npn
1950 \markdownRendererDlDefinitionEnd
1951 {
1952 \markdownRendererDlDefinitionEndPrototype
1953 }
1954 \seq_gput_right:Nn
1955 \g_@@_renderers_seq
1956 { dlDefinitionEnd }
1957 \prop_gput:Nnn
1958 \g_@@_renderer_arities_prop
1959 { dlDefinitionEnd }
1960 { 0 }
1961 \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.

```

1962 \ExplSyntaxOn
1963 \cs_gset_protected:Npn
1964 \markdownRendererDlEnd
1965 {
1966 \markdownRendererDlEndPrototype
1967 }
1968 \seq_gput_right:Nn
1969 \g_@@_renderers_seq
1970 { dlEnd }
1971 \prop_gput:Nnn
1972 \g_@@_renderer_arities_prop
1973 { dlEnd }
1974 { 0 }
1975 \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.

```

1976 \ExplSyntaxOn
1977 \cs_gset_protected:Npn
1978 \markdownRendererDlEndTight
1979 {
1980 \markdownRendererDlEndTightPrototype
1981 }
1982 \seq_gput_right:Nn
1983 \g_@@_renderers_seq
1984 { dlEndTight }
1985 \prop_gput:Nnn
1986 \g_@@_renderer_arities_prop
1987 { dlEndTight }
1988 { 0 }
1989 \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.

```

1990 \ExplSyntaxOn
1991 \cs_gset_protected:Npn
1992 \markdownRendererEllipsis
1993 {

```

```

1994 \markdownRendererEllipsisPrototype
1995 }
1996 \seq_gput_right:Nn
1997 \g_@@_renderers_seq
1998 { ellipsis }
1999 \prop_gput:Nnn
2000 \g_@@_renderer_arities_prop
2001 { ellipsis }
2002 { 0 }
2003 \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.

```

2004 \ExplSyntaxOn
2005 \cs_gset_protected:Npn
2006 \markdownRendererEmphasis
2007 {
2008 \markdownRendererEmphasisPrototype
2009 }
2010 \seq_gput_right:Nn
2011 \g_@@_renderers_seq
2012 { emphasis }
2013 \prop_gput:Nnn
2014 \g_@@_renderer_arities_prop
2015 { emphasis }
2016 { 1 }
2017 \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.

```

2018 \ExplSyntaxOn
2019 \cs_gset_protected:Npn
2020 \markdownRendererStrongEmphasis
2021 {
2022 \markdownRendererStrongEmphasisPrototype
2023 }
2024 \seq_gput_right:Nn
2025 \g_@@_renderers_seq
2026 { strongEmphasis }
2027 \prop_gput:Nnn
2028 \g_@@_renderer_arities_prop
2029 { strongEmphasis }
2030 { 1 }

```

2031 \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.

```
2032 \ExplSyntaxOn
2033 \cs_gset_protected:Npn
2034 \markdownRendererFencedCodeAttributeContextBegin
2035 {
2036 \markdownRendererFencedCodeAttributeContextBeginPrototype
2037 }
2038 \seq_gput_right:Nn
2039 \g_@@_renderers_seq
2040 { fencedCodeAttributeContextBegin }
2041 \prop_gput:Nnn
2042 \g_@@_renderer_arities_prop
2043 { fencedCodeAttributeContextBegin }
2044 { 0 }
2045 \cs_gset_protected:Npn
2046 \markdownRendererFencedCodeAttributeContextEnd
2047 {
2048 \markdownRendererFencedCodeAttributeContextEndPrototype
2049 }
2050 \seq_gput_right:Nn
2051 \g_@@_renderers_seq
2052 { fencedCodeAttributeContextEnd }
2053 \prop_gput:Nnn
2054 \g_@@_renderer_arities_prop
2055 { fencedCodeAttributeContextEnd }
2056 { 0 }
2057 \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.

```
2058 \ExplSyntaxOn
2059 \cs_gset_protected:Npn
2060 \markdownRendererFencedDivAttributeContextBegin
2061 {
```

```

2062 \markdownRendererFencedDivAttributeContextBeginPrototype
2063 }
2064 \seq_gput_right:Nn
2065 \g_@@_renderers_seq
2066 { fencedDivAttributeContextBegin }
2067 \prop_gput:Nnn
2068 \g_@@_renderer_arities_prop
2069 { fencedDivAttributeContextBegin }
2070 { 0 }
2071 \cs_gset_protected:Npn
2072 \markdownRendererFencedDivAttributeContextEnd
2073 {
2074 \markdownRendererFencedDivAttributeContextEndPrototype
2075 }
2076 \seq_gput_right:Nn
2077 \g_@@_renderers_seq
2078 { fencedDivAttributeContextEnd }
2079 \prop_gput:Nnn
2080 \g_@@_renderer_arities_prop
2081 { fencedDivAttributeContextEnd }
2082 { 0 }
2083 \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.

```

2084 \ExplSyntaxOn
2085 \cs_gset_protected:Npn
2086 \markdownRendererHeaderAttributeContextBegin
2087 {
2088 \markdownRendererHeaderAttributeContextBeginPrototype
2089 }
2090 \seq_gput_right:Nn
2091 \g_@@_renderers_seq
2092 { headerAttributeContextBegin }
2093 \prop_gput:Nnn
2094 \g_@@_renderer_arities_prop
2095 { headerAttributeContextBegin }
2096 { 0 }
2097 \cs_gset_protected:Npn
2098 \markdownRendererHeaderAttributeContextEnd
2099 {
2100 \markdownRendererHeaderAttributeContextEndPrototype

```

```

2101 }
2102 \seq_gput_right:Nn
2103 \g_@@_renderers_seq
2104 { headerAttributeContextEnd }
2105 \prop_gput:Nnn
2106 \g_@@_renderer_arities_prop
2107 { headerAttributeContextEnd }
2108 { 0 }
2109 \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.

```

2110 \ExplSyntaxOn
2111 \cs_gset_protected:Npn
2112 \markdownRendererHeadingOne
2113 {
2114 \markdownRendererHeadingOnePrototype
2115 }
2116 \seq_gput_right:Nn
2117 \g_@@_renderers_seq
2118 { headingOne }
2119 \prop_gput:Nnn
2120 \g_@@_renderer_arities_prop
2121 { headingOne }
2122 { 1 }
2123 \ExplSyntaxOff

```

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

```

2124 \ExplSyntaxOn
2125 \cs_gset_protected:Npn
2126 \markdownRendererHeadingTwo
2127 {
2128 \markdownRendererHeadingTwoPrototype
2129 }
2130 \seq_gput_right:Nn
2131 \g_@@_renderers_seq
2132 { headingTwo }
2133 \prop_gput:Nnn
2134 \g_@@_renderer_arities_prop
2135 { headingTwo }
2136 { 1 }
2137 \ExplSyntaxOff

```

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

```
2138 \ExplSyntaxOn
2139 \cs_gset_protected:Npn
2140 \markdownRendererHeadingThree
2141 {
2142 \markdownRendererHeadingThreePrototype
2143 }
2144 \seq_gput_right:Nn
2145 \g_@@_renderers_seq
2146 { headingThree }
2147 \prop_gput:Nnn
2148 \g_@@_renderer_arities_prop
2149 { headingThree }
2150 { 1 }
2151 \ExplSyntaxOff
```

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

```
2152 \ExplSyntaxOn
2153 \cs_gset_protected:Npn
2154 \markdownRendererHeadingFour
2155 {
2156 \markdownRendererHeadingFourPrototype
2157 }
2158 \seq_gput_right:Nn
2159 \g_@@_renderers_seq
2160 { headingFour }
2161 \prop_gput:Nnn
2162 \g_@@_renderer_arities_prop
2163 { headingFour }
2164 { 1 }
2165 \ExplSyntaxOff
```

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

```
2166 \ExplSyntaxOn
2167 \cs_gset_protected:Npn
2168 \markdownRendererHeadingFive
2169 {
2170 \markdownRendererHeadingFivePrototype
2171 }
2172 \seq_gput_right:Nn
2173 \g_@@_renderers_seq
2174 { headingFive }
2175 \prop_gput:Nnn
```

```

2176 \g_@@_renderer_arities_prop
2177 { headingFive }
2178 { 1 }
2179 \ExplSyntaxOff

```

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

```

2180 \ExplSyntaxOn
2181 \cs_gset_protected:Npn
2182 \markdownRendererHeadingSix
2183 {
2184 \markdownRendererHeadingSixPrototype
2185 }
2186 \seq_gput_right:Nn
2187 \g_@@_renderers_seq
2188 { headingSix }
2189 \prop_gput:Nnn
2190 \g_@@_renderer_arities_prop
2191 { headingSix }
2192 { 1 }
2193 \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.

```

2194 \ExplSyntaxOn
2195 \cs_gset_protected:Npn
2196 \markdownRendererInlineHtmlComment
2197 {
2198 \markdownRendererInlineHtmlCommentPrototype
2199 }
2200 \seq_gput_right:Nn
2201 \g_@@_renderers_seq
2202 { inlineHtmlComment }
2203 \prop_gput:Nnn
2204 \g_@@_renderer_arities_prop
2205 { inlineHtmlComment }
2206 { 1 }
2207 \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.

```

2208 \ExplSyntaxOn
2209 \cs_gset_protected:Npn
2210 \markdownRendererInlineHtmlTag
2211 {
2212 \markdownRendererInlineHtmlTagPrototype
2213 }
2214 \seq_gput_right:Nn
2215 \g_@@_renderers_seq
2216 { inlineHtmlTag }
2217 \prop_gput:Nnn
2218 \g_@@_renderer_arities_prop
2219 { inlineHtmlTag }
2220 { 1 }
2221 \cs_gset_protected:Npn
2222 \markdownRendererInputBlockHtmlElement
2223 {
2224 \markdownRendererInputBlockHtmlElementPrototype
2225 }
2226 \seq_gput_right:Nn
2227 \g_@@_renderers_seq
2228 { inputBlockHtmlElement }
2229 \prop_gput:Nnn
2230 \g_@@_renderer_arities_prop
2231 { inputBlockHtmlElement }
2232 { 1 }
2233 \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.

```

2234 \ExplSyntaxOn
2235 \cs_gset_protected:Npn
2236 \markdownRendererImage
2237 {
2238 \markdownRendererImagePrototype
2239 }
2240 \seq_gput_right:Nn
2241 \g_@@_renderers_seq

```

```

2242 { image }
2243 \prop_gput:Nnn
2244 \g_@@_renderer_arities_prop
2245 { image }
2246 { 4 }
2247 \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.

```

2248 \ExplSyntaxOn
2249 \cs_gset_protected:Npn
2250 \markdownRendererImageAttributeContextBegin
2251 {
2252 \markdownRendererImageAttributeContextBeginPrototype
2253 }
2254 \seq_gput_right:Nn
2255 \g_@@_renderers_seq
2256 { imageAttributeContextBegin }
2257 \prop_gput:Nnn
2258 \g_@@_renderer_arities_prop
2259 { imageAttributeContextBegin }
2260 { 0 }
2261 \cs_gset_protected:Npn
2262 \markdownRendererImageAttributeContextEnd
2263 {
2264 \markdownRendererImageAttributeContextEndPrototype
2265 }
2266 \seq_gput_right:Nn
2267 \g_@@_renderers_seq
2268 { imageAttributeContextEnd }
2269 \prop_gput:Nnn
2270 \g_@@_renderer_arities_prop
2271 { imageAttributeContextEnd }
2272 { 0 }
2273 \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.

```

2274 \ExplSyntaxOn

```

```

2275 \cs_gset_protected:Npn
2276 \markdownRendererInterblockSeparator
2277 {
2278 \markdownRendererInterblockSeparatorPrototype
2279 }
2280 \seq_gput_right:Nn
2281 \g_@@_renderers_seq
2282 { interblockSeparator }
2283 \prop_gput:Nnn
2284 \g_@@_renderer_arities_prop
2285 { interblockSeparator }
2286 { 0 }
2287 \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.

```

2288 \ExplSyntaxOn
2289 \cs_gset_protected:Npn
2290 \markdownRendererParagraphSeparator
2291 {
2292 \markdownRendererParagraphSeparatorPrototype
2293 }
2294 \seq_gput_right:Nn
2295 \g_@@_renderers_seq
2296 { paragraphSeparator }
2297 \prop_gput:Nnn
2298 \g_@@_renderer_arities_prop
2299 { paragraphSeparator }
2300 { 0 }
2301 \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.

```

2302 \ExplSyntaxOn
2303 \cs_gset_protected:Npn
2304 \markdownRendererLineBlockBegin
2305 {
2306 \markdownRendererLineBlockBeginPrototype

```

```

2307 }
2308 \seq_gput_right:Nn
2309 \g_@@_renderers_seq
2310 { lineBlockBegin }
2311 \prop_gput:Nnn
2312 \g_@@_renderer_arities_prop
2313 { lineBlockBegin }
2314 { 0 }
2315 \cs_gset_protected:Npn
2316 \markdownRendererLineBlockEnd
2317 {
2318 \markdownRendererLineBlockEndPrototype
2319 }
2320 \seq_gput_right:Nn
2321 \g_@@_renderers_seq
2322 { lineBlockEnd }
2323 \prop_gput:Nnn
2324 \g_@@_renderer_arities_prop
2325 { lineBlockEnd }
2326 { 0 }
2327 \ExplSyntaxOff

```

### 2.2.5.23 Line Break Renderers

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

```

2328 \ExplSyntaxOn
2329 \cs_gset_protected:Npn
2330 \markdownRendererSoftLineBreak
2331 {
2332 \markdownRendererSoftLineBreakPrototype
2333 }
2334 \seq_gput_right:Nn
2335 \g_@@_renderers_seq
2336 { softLineBreak }
2337 \prop_gput:Nnn
2338 \g_@@_renderer_arities_prop
2339 { softLineBreak }
2340 { 0 }
2341 \ExplSyntaxOff

```

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

```

2342 \ExplSyntaxOn
2343 \cs_gset_protected:Npn
2344 \markdownRendererHardLineBreak
2345 {

```

```

2346 \markdownRendererHardLineBreakPrototype
2347 }
2348 \seq_gput_right:Nn
2349 \g_@@_renderers_seq
2350 { hardLineBreak }
2351 \prop_gput:Nnn
2352 \g_@@_renderer_arities_prop
2353 { hardLineBreak }
2354 { 0 }
2355 \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.

```

2356 \ExplSyntaxOn
2357 \cs_gset_protected:Npn
2358 \markdownRendererLink
2359 {
2360 \markdownRendererLinkPrototype
2361 }
2362 \seq_gput_right:Nn
2363 \g_@@_renderers_seq
2364 { link }
2365 \prop_gput:Nnn
2366 \g_@@_renderer_arities_prop
2367 { link }
2368 { 4 }
2369 \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.

```

2370 \ExplSyntaxOn
2371 \cs_gset_protected:Npn
2372 \markdownRendererLinkAttributeContextBegin
2373 {
2374 \markdownRendererLinkAttributeContextBeginPrototype
2375 }
2376 \seq_gput_right:Nn
2377 \g_@@_renderers_seq

```

```

2378 { linkAttributeContextBegin }
2379 \prop_gput:Nnn
2380 \g_@@_renderer_arities_prop
2381 { linkAttributeContextBegin }
2382 { 0 }
2383 \cs_gset_protected:Npn
2384 \markdownRendererLinkAttributeContextEnd
2385 {
2386 \markdownRendererLinkAttributeContextEndPrototype
2387 }
2388 \seq_gput_right:Nn
2389 \g_@@_renderers_seq
2390 { linkAttributeContextEnd }
2391 \prop_gput:Nnn
2392 \g_@@_renderer_arities_prop
2393 { linkAttributeContextEnd }
2394 { 0 }
2395 \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.

```

2396 \ExplSyntaxOn
2397 \cs_gset_protected:Npn
2398 \markdownRendererMark
2399 {
2400 \markdownRendererMarkPrototype
2401 }
2402 \seq_gput_right:Nn
2403 \g_@@_renderers_seq
2404 { mark }
2405 \prop_gput:Nnn
2406 \g_@@_renderer_arities_prop
2407 { mark }
2408 { 1 }
2409 \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.

```
2410 \ExplSyntaxOn
2411 \cs_gset_protected:Npn
2412 \markdownRendererDocumentBegin
2413 {
2414 \markdownRendererDocumentBeginPrototype
2415 }
2416 \seq_gput_right:Nn
2417 \g_@@_renderers_seq
2418 { documentBegin }
2419 \prop_gput:Nnn
2420 \g_@@_renderer_arities_prop
2421 { documentBegin }
2422 { 0 }
2423 \cs_gset_protected:Npn
2424 \markdownRendererDocumentEnd
2425 {
2426 \markdownRendererDocumentEndPrototype
2427 }
2428 \seq_gput_right:Nn
2429 \g_@@_renderers_seq
2430 { documentEnd }
2431 \prop_gput:Nnn
2432 \g_@@_renderer_arities_prop
2433 { documentEnd }
2434 { 0 }
2435 \ExplSyntaxOff
```

### 2.2.5.28 Non-Breaking Space Renderer

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

```
2436 \ExplSyntaxOn
2437 \cs_gset_protected:Npn
2438 \markdownRendererNbsp
2439 {
2440 \markdownRendererNbspPrototype
2441 }
2442 \seq_gput_right:Nn
2443 \g_@@_renderers_seq
2444 { nbsp }
2445 \prop_gput:Nnn
2446 \g_@@_renderer_arities_prop
2447 { nbsp }
2448 { 0 }
2449 \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.

```
2450 \def\markdownRendererNote{%
2451 \markdownRendererNotePrototype}%
2452 \ExplSyntaxOn
2453 \seq_gput_right:Nn
2454 \g_@@_renderers_seq
2455 { note }
2456 \prop_gput:Nnn
2457 \g_@@_renderer_arities_prop
2458 { note }
2459 { 1 }
2460 \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.

```
2461 \ExplSyntaxOn
2462 \cs_gset_protected:Npn
2463 \markdownRendererOlBegin
2464 {
2465 \markdownRendererOlBeginPrototype
2466 }
2467 \seq_gput_right:Nn
2468 \g_@@_renderers_seq
2469 { olBegin }
2470 \prop_gput:Nnn
2471 \g_@@_renderer_arities_prop
2472 { olBegin }
2473 { 0 }
2474 \ExplSyntaxOff
```

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

```
2475 \ExplSyntaxOn
2476 \cs_gset_protected:Npn
2477 \markdownRendererOlBeginTight
2478 {
```



```

2479 \markdownRendererOlBeginTightPrototype
2480 }
2481 \seq_gput_right:Nn
2482 \g_@@_renderers_seq
2483 { olBeginTight }
2484 \prop_gput:Nnn
2485 \g_@@_renderer_arities_prop
2486 { olBeginTight }
2487 { 0 }
2488 \ExplSyntaxOff

```

The `\markdownRendererFancyOlBegin` macro represents the beginning of a fancy ordered list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the `fancyLists` option is enabled. The macro receives two arguments: the style of the list item labels (`Decimal`, `LowerRoman`, `UpperRoman`, `LowerAlpha`, and `UpperAlpha`), and the style of delimiters between list item labels and texts (`Default`, `OneParen`, and `Period`).

```

2489 \ExplSyntaxOn
2490 \cs_gset_protected:Npn
2491 \markdownRendererFancyOlBegin
2492 {
2493 \markdownRendererFancyOlBeginPrototype
2494 }
2495 \seq_gput_right:Nn
2496 \g_@@_renderers_seq
2497 { fancyOlBegin }
2498 \prop_gput:Nnn
2499 \g_@@_renderer_arities_prop
2500 { fancyOlBegin }
2501 { 2 }
2502 \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.

```

2503 \ExplSyntaxOn
2504 \cs_gset_protected:Npn
2505 \markdownRendererFancyOlBeginTight
2506 {
2507 \markdownRendererFancyOlBeginTightPrototype
2508 }
2509 \seq_gput_right:Nn
2510 \g_@@_renderers_seq

```

```

2511 { fancyOlBeginTight }
2512 \prop_gput:Nnn
2513 \g_@@_renderer_arities_prop
2514 { fancyOlBeginTight }
2515 { 2 }
2516 \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.

```

2517 \ExplSyntaxOn
2518 \cs_gset_protected:Npn
2519 \markdownRendererOlItem
2520 {
2521 \markdownRendererOlItemPrototype
2522 }
2523 \seq_gput_right:Nn
2524 \g_@@_renderers_seq
2525 { olItem }
2526 \prop_gput:Nnn
2527 \g_@@_renderer_arities_prop
2528 { olItem }
2529 { 0 }
2530 \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.

```

2531 \ExplSyntaxOn
2532 \cs_gset_protected:Npn
2533 \markdownRendererOlItemEnd
2534 {
2535 \markdownRendererOlItemEndPrototype
2536 }
2537 \seq_gput_right:Nn
2538 \g_@@_renderers_seq
2539 { olItemEnd }
2540 \prop_gput:Nnn
2541 \g_@@_renderer_arities_prop
2542 { olItemEnd }
2543 { 0 }
2544 \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.

```

2545 \ExplSyntaxOn
2546 \cs_gset_protected:Npn
2547 \markdownRendererOlItemWithNumber
2548 {
2549 \markdownRendererOlItemWithNumberPrototype
2550 }
2551 \seq_gput_right:Nn
2552 \g_@@_renderers_seq
2553 { olItemWithNumber }
2554 \prop_gput:Nnn
2555 \g_@@_renderer_arities_prop
2556 { olItemWithNumber }
2557 { 1 }
2558 \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.

```

2559 \ExplSyntaxOn
2560 \cs_gset_protected:Npn
2561 \markdownRendererFancyOlItem
2562 {
2563 \markdownRendererFancyOlItemPrototype
2564 }
2565 \seq_gput_right:Nn
2566 \g_@@_renderers_seq
2567 { fancyOlItem }
2568 \prop_gput:Nnn
2569 \g_@@_renderer_arities_prop
2570 { fancyOlItem }
2571 { 0 }
2572 \ExplSyntaxOff

```

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

```

2573 \ExplSyntaxOn
2574 \cs_gset_protected:Npn
2575 \markdownRendererFancyOlItemEnd
2576 {
2577 \markdownRendererFancyOlItemEndPrototype
2578 }
2579 \seq_gput_right:Nn
2580 \g_@@_renderers_seq

```

```

2581 { fancyO1ItemEnd }
2582 \prop_gput:Nnn
2583 \g_@@_renderer_arities_prop
2584 { fancyO1ItemEnd }
2585 { 0 }
2586 \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.

```

2587 \ExplSyntaxOn
2588 \cs_gset_protected:Npn
2589 \markdownRendererFancyO1ItemWithNumber
2590 {
2591 \markdownRendererFancyO1ItemWithNumberPrototype
2592 }
2593 \seq_gput_right:Nn
2594 \g_@@_renderers_seq
2595 { fancyO1ItemWithNumber }
2596 \prop_gput:Nnn
2597 \g_@@_renderer_arities_prop
2598 { fancyO1ItemWithNumber }
2599 { 1 }
2600 \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.

```

2601 \ExplSyntaxOn
2602 \cs_gset_protected:Npn
2603 \markdownRendererO1End
2604 {
2605 \markdownRendererO1EndPrototype
2606 }
2607 \seq_gput_right:Nn
2608 \g_@@_renderers_seq
2609 { olEnd }
2610 \prop_gput:Nnn
2611 \g_@@_renderer_arities_prop
2612 { olEnd }
2613 { 0 }
2614 \ExplSyntaxOff

```

The `\markdownRendererO1EndTight` macro represents the end of an ordered list that contains no item with several paragraphs of text (the list is tight). This macro

will only be produced, when the `tightLists` option is enabled and the `fancyLists` option is disabled. The macro receives no arguments.

```

2615 \ExplSyntaxOn
2616 \cs_gset_protected:Npn
2617 \markdownRendererOlEndTight
2618 {
2619 \markdownRendererOlEndTightPrototype
2620 }
2621 \seq_gput_right:Nn
2622 \g_@@_renderers_seq
2623 { olEndTight }
2624 \prop_gput:Nnn
2625 \g_@@_renderer_arities_prop
2626 { olEndTight }
2627 { 0 }
2628 \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.

```

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

```

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

```

2643 \ExplSyntaxOn
2644 \cs_gset_protected:Npn
2645 \markdownRendererFancyOlEndTight
2646 {
2647 \markdownRendererFancyOlEndTightPrototype
2648 }

```

```

2649 \seq_gput_right:Nn
2650 \g_@@_renderers_seq
2651 { fancy01EndTight }
2652 \prop_gput:Nnn
2653 \g_@@_renderer_arities_prop
2654 { fancy01EndTight }
2655 { 0 }
2656 \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.

```

2657 \ExplSyntaxOn
2658 \cs_gset_protected:Npn
2659 \markdownRendererInputRawInline
2660 {
2661 \markdownRendererInputRawInlinePrototype
2662 }
2663 \seq_gput_right:Nn
2664 \g_@@_renderers_seq
2665 { inputRawInline }
2666 \prop_gput:Nnn
2667 \g_@@_renderer_arities_prop
2668 { inputRawInline }
2669 { 2 }
2670 \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.

```

2671 \ExplSyntaxOn
2672 \cs_gset_protected:Npn
2673 \markdownRendererInputRawBlock
2674 {
2675 \markdownRendererInputRawBlockPrototype
2676 }
2677 \seq_gput_right:Nn
2678 \g_@@_renderers_seq
2679 { inputRawBlock }
2680 \prop_gput:Nnn
2681 \g_@@_renderer_arities_prop
2682 { inputRawBlock }

```

```

2683 { 2 }
2684 \ExplSyntaxOff

```

### 2.2.5.32 Section Renderers

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

```

2685 \ExplSyntaxOn
2686 \cs_gset_protected:Npn
2687 \markdownRendererSectionBegin
2688 {
2689 \markdownRendererSectionBeginPrototype
2690 }
2691 \seq_gput_right:Nn
2692 \g_@@_renderers_seq
2693 { sectionBegin }
2694 \prop_gput:Nnn
2695 \g_@@_renderer_arities_prop
2696 { sectionBegin }
2697 { 0 }
2698 \cs_gset_protected:Npn
2699 \markdownRendererSectionEnd
2700 {
2701 \markdownRendererSectionEndPrototype
2702 }
2703 \seq_gput_right:Nn
2704 \g_@@_renderers_seq
2705 { sectionEnd }
2706 \prop_gput:Nnn
2707 \g_@@_renderer_arities_prop
2708 { sectionEnd }
2709 { 0 }
2710 \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.

```

2711 \ExplSyntaxOn
2712 \cs_gset_protected:Npn
2713 \markdownRendererReplacementCharacter
2714 {
2715 \markdownRendererReplacementCharacterPrototype
2716 }
2717 \seq_gput_right:Nn
2718 \g_@@_renderers_seq
2719 { replacementCharacter }

```

```

2720 \prop_gput:Nnn
2721 \g_@@_renderer_arities_prop
2722 { replacementCharacter }
2723 { 0 }
2724 \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`.

```

2725 \ExplSyntaxOn
2726 \cs_gset_protected:Npn
2727 \markdownRendererLeftBrace
2728 {
2729 \markdownRendererLeftBracePrototype
2730 }
2731 \seq_gput_right:Nn
2732 \g_@@_renderers_seq
2733 { leftBrace }
2734 \prop_gput:Nnn
2735 \g_@@_renderer_arities_prop
2736 { leftBrace }
2737 { 0 }
2738 \cs_gset_protected:Npn
2739 \markdownRendererRightBrace
2740 {
2741 \markdownRendererRightBracePrototype
2742 }
2743 \seq_gput_right:Nn
2744 \g_@@_renderers_seq
2745 { rightBrace }
2746 \prop_gput:Nnn
2747 \g_@@_renderer_arities_prop
2748 { rightBrace }
2749 { 0 }
2750 \cs_gset_protected:Npn
2751 \markdownRendererDollarSign
2752 {
2753 \markdownRendererDollarSignPrototype
2754 }
2755 \seq_gput_right:Nn
2756 \g_@@_renderers_seq
2757 { dollarSign }
2758 \prop_gput:Nnn
2759 \g_@@_renderer_arities_prop
2760 { dollarSign }

```



```

2761 { 0 }
2762 \cs_gset_protected:Npn
2763 \markdownRendererPercentSign
2764 {
2765 \markdownRendererPercentSignPrototype
2766 }
2767 \seq_gput_right:Nn
2768 \g_@@_renderers_seq
2769 { percentSign }
2770 \prop_gput:Nnn
2771 \g_@@_renderer_arities_prop
2772 { percentSign }
2773 { 0 }
2774 \cs_gset_protected:Npn
2775 \markdownRendererAmpersand
2776 {
2777 \markdownRendererAmpersandPrototype
2778 }
2779 \seq_gput_right:Nn
2780 \g_@@_renderers_seq
2781 { ampersand }
2782 \prop_gput:Nnn
2783 \g_@@_renderer_arities_prop
2784 { ampersand }
2785 { 0 }
2786 \cs_gset_protected:Npn
2787 \markdownRendererUnderscore
2788 {
2789 \markdownRendererUnderscorePrototype
2790 }
2791 \seq_gput_right:Nn
2792 \g_@@_renderers_seq
2793 { underscore }
2794 \prop_gput:Nnn
2795 \g_@@_renderer_arities_prop
2796 { underscore }
2797 { 0 }
2798 \cs_gset_protected:Npn
2799 \markdownRendererHash
2800 {
2801 \markdownRendererHashPrototype
2802 }
2803 \seq_gput_right:Nn
2804 \g_@@_renderers_seq
2805 { hash }
2806 \prop_gput:Nnn
2807 \g_@@_renderer_arities_prop

```

```

2808 { hash }
2809 { 0 }
2810 \cs_gset_protected:Npn
2811 \markdownRendererCircumflex
2812 {
2813 \markdownRendererCircumflexPrototype
2814 }
2815 \seq_gput_right:Nn
2816 \g_@@_renderers_seq
2817 { circumflex }
2818 \prop_gput:Nnn
2819 \g_@@_renderer_arities_prop
2820 { circumflex }
2821 { 0 }
2822 \cs_gset_protected:Npn
2823 \markdownRendererBackslash
2824 {
2825 \markdownRendererBackslashPrototype
2826 }
2827 \seq_gput_right:Nn
2828 \g_@@_renderers_seq
2829 { backslash }
2830 \prop_gput:Nnn
2831 \g_@@_renderer_arities_prop
2832 { backslash }
2833 { 0 }
2834 \cs_gset_protected:Npn
2835 \markdownRendererTilde
2836 {
2837 \markdownRendererTildePrototype
2838 }
2839 \seq_gput_right:Nn
2840 \g_@@_renderers_seq
2841 { tilde }
2842 \prop_gput:Nnn
2843 \g_@@_renderer_arities_prop
2844 { tilde }
2845 { 0 }
2846 \cs_gset_protected:Npn
2847 \markdownRendererPipe
2848 {
2849 \markdownRendererPipePrototype
2850 }
2851 \seq_gput_right:Nn
2852 \g_@@_renderers_seq
2853 { pipe }
2854 \prop_gput:Nnn

```

```

2855 \g_@@_renderer_arities_prop
2856 { pipe }
2857 { 0 }
2858 \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.

```

2859 \ExplSyntaxOn
2860 \cs_gset_protected:Npn
2861 \markdownRendererStrikeThrough
2862 {
2863 \markdownRendererStrikeThroughPrototype
2864 }
2865 \seq_gput_right:Nn
2866 \g_@@_renderers_seq
2867 { strikeThrough }
2868 \prop_gput:Nnn
2869 \g_@@_renderer_arities_prop
2870 { strikeThrough }
2871 { 1 }
2872 \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.

```

2873 \ExplSyntaxOn
2874 \cs_gset_protected:Npn
2875 \markdownRendererSubscript
2876 {
2877 \markdownRendererSubscriptPrototype
2878 }
2879 \seq_gput_right:Nn
2880 \g_@@_renderers_seq
2881 { subscript }
2882 \prop_gput:Nnn
2883 \g_@@_renderer_arities_prop
2884 { subscript }
2885 { 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.

```

2886 \cs_gset_protected:Npn
2887 \markdownRendererSuperscript
2888 {
2889 \markdownRendererSuperscriptPrototype
2890 }
2891 \seq_gput_right:Nn
2892 \g_@@_renderers_seq
2893 { superscript }
2894 \prop_gput:Nnn
2895 \g_@@_renderer_arities_prop
2896 { superscript }
2897 { 1 }
2898 \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.

```

2899 \ExplSyntaxOn
2900 \cs_gset_protected:Npn
2901 \markdownRendererTableAttributeContextBegin
2902 {
2903 \markdownRendererTableAttributeContextBeginPrototype
2904 }
2905 \seq_gput_right:Nn
2906 \g_@@_renderers_seq
2907 { tableAttributeContextBegin }
2908 \prop_gput:Nnn
2909 \g_@@_renderer_arities_prop
2910 { tableAttributeContextBegin }
2911 { 0 }
2912 \cs_gset_protected:Npn
2913 \markdownRendererTableAttributeContextEnd
2914 {
2915 \markdownRendererTableAttributeContextEndPrototype
2916 }
2917 \seq_gput_right:Nn
2918 \g_@@_renderers_seq
2919 { tableAttributeContextEnd }
2920 \prop_gput:Nnn

```

```

2921 \g_@@_renderer_arities_prop
2922 { tableAttributeContextEnd }
2923 { 0 }
2924 \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.

```

2925 \ExplSyntaxOn
2926 \cs_gset_protected:Npn
2927 \markdownRendererTable
2928 {
2929 \markdownRendererTablePrototype
2930 }
2931 \seq_gput_right:Nn
2932 \g_@@_renderers_seq
2933 { table }
2934 \prop_gput:Nnn
2935 \g_@@_renderer_arities_prop
2936 { table }
2937 { 3 }
2938 \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.

```

2939 \ExplSyntaxOn
2940 \cs_gset_protected:Npn
2941 \markdownRendererInlineMath
2942 {
2943 \markdownRendererInlineMathPrototype

```

```

2944 }
2945 \seq_gput_right:Nn
2946 \g_@@_renderers_seq
2947 { inlineMath }
2948 \prop_gput:Nnn
2949 \g_@@_renderer_arities_prop
2950 { inlineMath }
2951 { 1 }
2952 \cs_gset_protected:Npn
2953 \markdownRendererDisplayMath
2954 {
2955 \markdownRendererDisplayMathPrototype
2956 }
2957 \seq_gput_right:Nn
2958 \g_@@_renderers_seq
2959 { displayMath }
2960 \prop_gput:Nnn
2961 \g_@@_renderer_arities_prop
2962 { displayMath }
2963 { 1 }
2964 \ExplSyntaxOff

```

#### 2.2.5.41 Thematic Break Renderer

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

```

2965 \ExplSyntaxOn
2966 \cs_gset_protected:Npn
2967 \markdownRendererThematicBreak
2968 {
2969 \markdownRendererThematicBreakPrototype
2970 }
2971 \seq_gput_right:Nn
2972 \g_@@_renderers_seq
2973 { thematicBreak }
2974 \prop_gput:Nnn
2975 \g_@@_renderer_arities_prop
2976 { thematicBreak }
2977 { 0 }
2978 \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.

```
2979 \ExplSyntaxOn
2980 \cs_gset_protected:Npn
2981 \markdownRendererTickedBox
2982 {
2983 \markdownRendererTickedBoxPrototype
2984 }
2985 \seq_gput_right:Nn
2986 \g_@@_renderers_seq
2987 { tickedBox }
2988 \prop_gput:Nnn
2989 \g_@@_renderer_arities_prop
2990 { tickedBox }
2991 { 0 }
2992 \cs_gset_protected:Npn
2993 \markdownRendererHalfTickedBox
2994 {
2995 \markdownRendererHalfTickedBoxPrototype
2996 }
2997 \seq_gput_right:Nn
2998 \g_@@_renderers_seq
2999 { halfTickedBox }
3000 \prop_gput:Nnn
3001 \g_@@_renderer_arities_prop
3002 { halfTickedBox }
3003 { 0 }
3004 \cs_gset_protected:Npn
3005 \markdownRendererUntickedBox
3006 {
3007 \markdownRendererUntickedBoxPrototype
3008 }
3009 \seq_gput_right:Nn
3010 \g_@@_renderers_seq
3011 { untickedBox }
3012 \prop_gput:Nnn
3013 \g_@@_renderer_arities_prop
3014 { untickedBox }
3015 { 0 }
3016 \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.

```

3017 \ExplSyntaxOn
3018 \cs_gset_protected:Npn
3019 \markdownRendererWarning
3020 {
3021 \markdownRendererWarningPrototype
3022 }
3023 \cs_gset_protected:Npn
3024 \markdownRendererError
3025 {
3026 \markdownRendererErrorPrototype
3027 }
3028 \seq_gput_right:Nn
3029 \g_@@_renderers_seq
3030 { warning }
3031 \prop_gput:Nnn
3032 \g_@@_renderer_arities_prop
3033 { warning }
3034 { 4 }
3035 \seq_gput_right:Nn
3036 \g_@@_renderers_seq
3037 { error }
3038 \prop_gput:Nnn
3039 \g_@@_renderer_arities_prop
3040 { error }
3041 { 4 }
3042 \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.

```

3043 \ExplSyntaxOn
3044 \cs_gset_protected:Npn
3045 \markdownRendererJekyllDataBegin
3046 {
3047 \markdownRendererJekyllDataBeginPrototype

```



```

3048 }
3049 \seq_gput_right:Nn
3050 \g_@@_renderers_seq
3051 { jekyllDataBegin }
3052 \prop_gput:Nnn
3053 \g_@@_renderer_arities_prop
3054 { jekyllDataBegin }
3055 { 0 }
3056 \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.

```

3057 \ExplSyntaxOn
3058 \cs_gset_protected:Npn
3059 \markdownRendererJekyllDataEnd
3060 {
3061 \markdownRendererJekyllDataEndPrototype
3062 }
3063 \seq_gput_right:Nn
3064 \g_@@_renderers_seq
3065 { jekyllDataEnd }
3066 \prop_gput:Nnn
3067 \g_@@_renderer_arities_prop
3068 { jekyllDataEnd }
3069 { 0 }
3070 \ExplSyntaxOff

```

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

```

3071 \ExplSyntaxOn
3072 \cs_gset_protected:Npn
3073 \markdownRendererJekyllDataMappingBegin
3074 {
3075 \markdownRendererJekyllDataMappingBeginPrototype
3076 }
3077 \seq_gput_right:Nn
3078 \g_@@_renderers_seq
3079 { jekyllDataMappingBegin }
3080 \prop_gput:Nnn
3081 \g_@@_renderer_arities_prop
3082 { jekyllDataMappingBegin }
3083 { 2 }

```

```
3084 \ExplSyntaxOff
```

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

```
3085 \ExplSyntaxOn
3086 \cs_gset_protected:Npn
3087 \markdownRendererJekyllDataMappingEnd
3088 {
3089 \markdownRendererJekyllDataMappingEndPrototype
3090 }
3091 \seq_gput_right:Nn
3092 \g_@@_renderers_seq
3093 { jekyllDataMappingEnd }
3094 \prop_gput:Nnn
3095 \g_@@_renderer_arities_prop
3096 { jekyllDataMappingEnd }
3097 { 0 }
3098 \ExplSyntaxOff
```

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

```
3099 \ExplSyntaxOn
3100 \cs_gset_protected:Npn
3101 \markdownRendererJekyllDataSequenceBegin
3102 {
3103 \markdownRendererJekyllDataSequenceBeginPrototype
3104 }
3105 \seq_gput_right:Nn
3106 \g_@@_renderers_seq
3107 { jekyllDataSequenceBegin }
3108 \prop_gput:Nnn
3109 \g_@@_renderer_arities_prop
3110 { jekyllDataSequenceBegin }
3111 { 2 }
3112 \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.

```
3113 \ExplSyntaxOn
3114 \cs_gset_protected:Npn
3115 \markdownRendererJekyllDataSequenceEnd
```

```

3116 {
3117 \markdownRendererJekyllDataSequenceEndPrototype
3118 }
3119 \seq_gput_right:Nn
3120 \g_@@_renderers_seq
3121 { jekyllDataSequenceEnd }
3122 \prop_gput:Nnn
3123 \g_@@_renderer_arities_prop
3124 { jekyllDataSequenceEnd }
3125 { 0 }
3126 \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.

```

3127 \ExplSyntaxOn
3128 \cs_gset_protected:Npn
3129 \markdownRendererJekyllDataBoolean
3130 {
3131 \markdownRendererJekyllDataBooleanPrototype
3132 }
3133 \seq_gput_right:Nn
3134 \g_@@_renderers_seq
3135 { jekyllDataBoolean }
3136 \prop_gput:Nnn
3137 \g_@@_renderer_arities_prop
3138 { jekyllDataBoolean }
3139 { 2 }
3140 \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.

```

3141 \ExplSyntaxOn
3142 \cs_gset_protected:Npn
3143 \markdownRendererJekyllDataNumber
3144 {
3145 \markdownRendererJekyllDataNumberPrototype
3146 }
3147 \seq_gput_right:Nn
3148 \g_@@_renderers_seq
3149 { jekyllDataNumber }

```

```

3150 \prop_gput:Nnn
3151 \g_@@_renderer_arities_prop
3152 { jekyllDataNumber }
3153 { 2 }
3154 \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 `\markdownRendererJekyllDataProgrammaticString` receives the scalar value after all markdown markup and special `TEX` characters in the string have been replaced by `TEX` macros, `\markdownRendererJekyllDataTypographicString` receives the raw scalar value. Therefore, whereas the `\markdownRendererJekyllDataTypographicString` macro is more appropriate for texts that are supposed to be typeset with `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 `TEX`.

```

3155 \ExplSyntaxOn
3156 \cs_gset_protected:Npn
3157 \markdownRendererJekyllDataTypographicString
3158 {
3159 \markdownRendererJekyllDataTypographicStringPrototype
3160 }
3161 \cs_gset_protected:Npn
3162 \markdownRendererJekyllDataProgrammaticString
3163 {
3164 \markdownRendererJekyllDataProgrammaticStringPrototype
3165 }
3166 \seq_gput_right:Nn
3167 \g_@@_renderers_seq
3168 { jekyllDataTypographicString }
3169 \prop_gput:Nnn
3170 \g_@@_renderer_arities_prop
3171 { jekyllDataTypographicString }
3172 { 2 }
3173 \seq_gput_right:Nn
3174 \g_@@_renderers_seq
3175 { jekyllDataProgrammaticString }
3176 \prop_gput:Nnn
3177 \g_@@_renderer_arities_prop
3178 { jekyllDataProgrammaticString }
3179 { 2 }
3180 \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.

```

3181 \ExplSyntaxOn
3182 \cs_gset:Npn
3183 \markdownRendererJekyllDataTypographicString
3184 {
3185 \cs_if_exist:NTF
3186 \markdownRendererJekyllDataString
3187 {
3188 \@@_if_option:nTF
3189 { experimental }
3190 {
3191 \markdownError
3192 {
3193 The~jekyllDataString~renderer~has~been~deprecated,~
3194 to~be~removed~in~Markdown~4.0.0
3195 }
3196 }
3197 {
3198 \markdownWarning
3199 {
3200 The~jekyllDataString~renderer~has~been~deprecated,~
3201 to~be~removed~in~Markdown~4.0.0
3202 }
3203 }
3204 }
3205 }
3206 {
3207 \cs_if_exist:NTF
3208 \markdownRendererJekyllDataStringPrototype
3209 {
3210 \@@_if_option:nTF
3211 { experimental }
3212 {
3213 \markdownError
3214 {
3215 The~jekyllDataString~renderer~prototype~
3216 has~been~deprecated,~
3217 to~be~removed~in~Markdown~4.0.0
3218 }
3219 }
3220 {
3221 \markdownWarning
3222 {

```

```

3223 The~jekyllDataString~renderer~prototype~
3224 has~been~deprecated,~
3225 to~be~removed~in~Markdown~4.0.0
3226 }
3227 \markdownRendererJekyllDataStringPrototype
3228 }
3229 }
3230 {
3231 \markdownRendererJekyllDataTypographicStringPrototype
3232 }
3233 }
3234 }
3235 \seq_gput_right:Nn
3236 \g_@@_renderers_seq
3237 { jekyllDataString }
3238 \prop_gput:Nnn
3239 \g_@@_renderer_arities_prop
3240 { jekyllDataString }
3241 { 2 }
3242 \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.

```

3243 \ExplSyntaxOn
3244 \cs_gset_protected:Npn
3245 \markdownRendererJekyllDataEmpty
3246 {
3247 \markdownRendererJekyllDataEmptyPrototype
3248 }
3249 \seq_gput_right:Nn
3250 \g_@@_renderers_seq
3251 { jekyllDataEmpty }
3252 \prop_gput:Nnn
3253 \g_@@_renderer_arities_prop
3254 { jekyllDataEmpty }
3255 { 1 }
3256 \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.

```

3257 \ExplSyntaxOn
3258 \cs_new:Nn \@@_define_renderers:
3259 {
3260 \seq_map_inline:Nn
3261 \g_@@_renderers_seq
3262 {
3263 \@@_define_renderer:n
3264 { #1 }
3265 }
3266 }
3267 \cs_new:Nn \@@_define_renderer:n
3268 {
3269 \@@_renderer_tl_to_csname:nN
3270 { #1 }
3271 \l_tmpa_tl
3272 \prop_get:NnN
3273 \g_@@_renderer_arities_prop
3274 { #1 }
3275 \l_tmpb_tl
3276 \@@_define_renderer:ncV
3277 { #1 }
3278 { \l_tmpa_tl }
3279 \l_tmpb_tl
3280 }
3281 \cs_new:Nn \@@_renderer_tl_to_csname:nN
3282 {
3283 \tl_set:Nn
3284 \l_tmpa_tl
3285 { \str_uppercase:n { #1 } }
3286 \tl_set:Nx
3287 #2
3288 {
3289 markdownRenderer
3290 \tl_head:f { \l_tmpa_tl }
3291 \tl_tail:n { #1 }
3292 }
3293 }
3294 \tl_new:N
3295 \l_@@_renderer_definition_tl
3296 \bool_new:N
3297 \g_@@_appending_renderer_bool

```

```

3298 \bool_new:N
3299 \g_@@_unprotected_renderer_bool
3300 \cs_new:Nn \@@_define_renderer:nNn
3301 {
3302 \keys_define:nn
3303 { markdown/options/renderers }
3304 {
3305 #1 .code:n = {
3306 \tl_set:Nn
3307 \l_@@_renderer_definition_tl
3308 { ##1 }
3309 \regex_replace_all:nnN
3310 { \cP\#0 }
3311 { #1 }
3312 \l_@@_renderer_definition_tl
3313 \bool_if:NT
3314 \g_@@_appending_renderer_bool
3315 {
3316 \@@_tl_set_from_cs:NNn
3317 \l_tmpa_tl
3318 #2
3319 { #3 }
3320 \tl_put_left:NV
3321 \l_@@_renderer_definition_tl
3322 \l_tmpa_tl
3323 }
3324 \bool_if:NTF
3325 \g_@@_unprotected_renderer_bool
3326 {
3327 \tl_set:Nn
3328 \l_tmpa_tl
3329 { \cs_set:Npn }
3330 }
3331 {
3332 \tl_set:Nn
3333 \l_tmpa_tl
3334 { \cs_set_protected:Npn }
3335 }
3336 \exp_last_unbraced:NNV
3337 \cs_generate_from_arg_count:NNnV
3338 #2
3339 \l_tmpa_tl
3340 { #3 }
3341 \l_@@_renderer_definition_tl
3342 },
3343 }

```

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.

```

3344 \str_if_eq:nnT
3345 { #1 }
3346 { jekyllDataString }
3347 {
3348 \cs_undefine:N
3349 #2
3350 }
3351 }

```

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.

```

3352 \cs_new_protected:Nn
3353 \@@_tl_set_from_cs:NNn
3354 {
3355 \tl_set:Nn
3356 \l_tmpa_tl
3357 { #2 }
3358 \int_step_inline:nn
3359 { #3 }
3360 {
3361 \exp_args:Nnc
3362 \tl_put_right:Nn
3363 \l_tmpa_tl
3364 { @@_tl_set_from_cs_parameter_ ##1 }
3365 }
3366 \exp_args:NNV
3367 \tl_set:No
3368 \l_tmpb_tl
3369 \l_tmpa_tl
3370 \regex_replace_all:nnN
3371 { \cP. }
3372 { \0\0 }
3373 \l_tmpb_tl
3374 \int_step_inline:nn
3375 { #3 }
3376 {
3377 \regex_replace_all:nnN
3378 { \c { @@_tl_set_from_cs_parameter_ ##1 } }
3379 { \cP\# ##1 }
3380 \l_tmpb_tl
3381 }
3382 \tl_set:NV
3383 #1

```

```

3384 \l_tmpb_tl
3385 }
3386 \cs_generate_variant:Nn
3387 \@@_define_renderer:nNn
3388 { ncV }
3389 \cs_generate_variant:Nn
3390 \cs_generate_from_arg_count:NNnn
3391 { NNnV }
3392 \cs_generate_variant:Nn
3393 \tl_put_left:Nn
3394 { Nv }
3395 \keys_define:nn
3396 { markdown/options }
3397 {
3398 renderers .code:n = {
3399 \bool_gset_false:N
3400 \g_@@_unprotected_renderer_bool
3401 \keys_set:nn
3402 { markdown/options/renderers }
3403 { #1 }
3404 },
3405 unprotectedRenderers .code:n = {
3406 \bool_gset_true:N
3407 \g_@@_unprotected_renderer_bool
3408 \keys_set:nn
3409 { markdown/options/renderers }
3410 { #1 }
3411 },
3412 }

```

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.
 }
}

```

```

3413 \tl_new:N
3414 \l_@@_renderer_glob_definition_tl
3415 \seq_new:N
3416 \l_@@_renderer_glob_results_seq
3417 \regex_const:Nn
3418 \c_@@_appending_key_regex
3419 { \s*+$ }

```

```

3420 \keys_define:nn
3421 { markdown/options/renderers }
3422 {
3423 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 += {...},
 },
 }
 },
 },
}

```

```

3424 \regex_match:NVTF
3425 \c_@@_appending_key_regex
3426 \l_keys_key_str
3427 {
3428 \bool_gset_true:N
3429 \g_@@_appending_renderer_bool
3430 \tl_set:NV
3431 \l_tmpa_tl
3432 \l_keys_key_str
3433 \regex_replace_once:NnN

```

```

3434 \c_@@_appending_key_regex
3435 { }
3436 \l_tmpa_tl
3437 \tl_set:Nx
3438 \l_tmpb_tl
3439 { { \l_tmpa_tl } = }
3440 \tl_put_right:Nn
3441 \l_tmpb_tl
3442 { { #1 } }
3443 \keys_set:nV
3444 { markdown/options/renderers }
3445 \l_tmpb_tl
3446 \bool_gset_false:N
3447 \g_@@_appending_renderer_bool
3448 }

```

In addition to exact token renderer names, we also support wildcards (\*) and enumerations (l) 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.
 }
}

```

```

3449 {
3450 \@_glob_seq:VnN
3451 \l_keys_key_str
3452 { g_@_renderers_seq }
3453 \l_@_renderer_glob_results_seq
3454 \seq_if_empty:NTF
3455 \l_@_renderer_glob_results_seq
3456 {
3457 \msg_error:nnV
3458 { markdown }
3459 { undefined-renderer }
3460 \l_keys_key_str
3461 }
3462 {
3463 \tl_set:Nn
3464 \l_@_renderer_glob_definition_tl
3465 { \exp_not:n { #1 } }
3466 \seq_map_inline:Nn
3467 \l_@_renderer_glob_results_seq
3468 {
3469 \tl_set:Nn
3470 \l_tmpa_tl
3471 { { ##1 } = }
3472 \tl_put_right:Nx
3473 \l_tmpa_tl
3474 { { \l_@_renderer_glob_definition_tl } }
3475 \keys_set:nV
3476 { markdown/options/renderers }
3477 \l_tmpa_tl
3478 }
3479 }
3480 }
3481 },
3482 }
3483 \msg_new:nnn
3484 { markdown }
3485 { undefined-renderer }
3486 {
3487 Renderer~#1~is~undefined.
3488 }
3489 \cs_generate_variant:Nn
3490 \@_glob_seq:nnN
3491 { VnN }
3492 \cs_generate_variant:Nn
3493 \cs_generate_from_arg_count:NNnn
3494 { cNVV }
3495 \cs_generate_variant:Nn

```

```

3496 \msg_error:nnn
3497 { nnV }
3498 \prg_generate_conditional_variant:Nnn
3499 \regex_match:Nn
3500 { NV }
3501 { TF }
3502 \prop_new:N
3503 \g_@@_glob_cache_prop
3504 \tl_new:N
3505 \l_@@_current_glob_tl
3506 \cs_new:Nn
3507 \@@_glob_seq:nnN
3508 {
3509 \tl_set:Nn
3510 \l_@@_current_glob_tl
3511 { ^ #1 $ }
3512 \prop_get:NeNTF
3513 \g_@@_glob_cache_prop
3514 { #2 / \l_@@_current_glob_tl }
3515 \l_tmpa_clist
3516 {
3517 \seq_set_from_clist:NN
3518 #3
3519 \l_tmpa_clist
3520 }
3521 {
3522 \seq_clear:N
3523 #3
3524 \regex_replace_all:nnN
3525 { * }
3526 { .* }
3527 \l_@@_current_glob_tl
3528 \regex_set:NV
3529 \l_tmpa_regex
3530 \l_@@_current_glob_tl
3531 \seq_map_inline:cn
3532 { #2 }
3533 {
3534 \regex_match:NnT
3535 \l_tmpa_regex
3536 { ##1 }
3537 {
3538 \seq_put_right:Nn
3539 #3
3540 { ##1 }
3541 }
3542 }

```

```

3543 \clist_set_from_seq:NN
3544 \l_tmpa_clist
3545 #3
3546 \prop_gput:NeV
3547 \g_@@_glob_cache_prop
3548 { #2 / \l_@@_current_glob_tl }
3549 \l_tmpa_clist
3550 }
3551 }
3552 \cs_generate_variant:Nn
3553 \regex_set:Nn
3554 { NV }
3555 \cs_generate_variant:Nn
3556 \prop_gput:Nnn
3557 { NeV }

```

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

```

3558 \str_if_eq:VVT
3559 \c_@@_top_layer_tl
3560 \c_@@_option_layer_plain_tex_tl
3561 {
3562 \@@_define_renderers:
3563 }
3564 \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`.

```

3565 \ExplSyntaxOn
3566 \keys_define:nn
3567 { markdown/jekyllData }
3568 { }
3569 \ExplSyntaxOff

```

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

```

3570 \ExplSyntaxOn
3571 \@@_with_various_cases:nn
3572 { jekyllDataRenderers }
3573 {
3574 \keys_define:nn
3575 { markdown/options }
3576 {

```

```

3577 #1 .code:n = {
3578 \tl_set:Nn
3579 \l_tmpa_tl
3580 { ##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.

```

3581 \tl_replace_all:NnV
3582 \l_tmpa_tl
3583 { / }
3584 \c_backslash_str
3585 \keys_set:nV
3586 { markdown/options/jekyll-data-renderers }
3587 \l_tmpa_tl
3588 },
3589 }
3590 }
3591 \keys_define:nn
3592 { markdown/options/jekyll-data-renderers }
3593 {
3594 unknown .code:n = {
3595 \tl_set_eq:NN
3596 \l_tmpa_tl
3597 \l_keys_key_str
3598 \tl_replace_all:NVn
3599 \l_tmpa_tl
3600 \c_backslash_str
3601 { / }
3602 \tl_put_right:Nn
3603 \l_tmpa_tl
3604 {
3605 .code:n = { #1 }
3606 }
3607 \keys_define:nV
3608 { markdown/jekyllData }
3609 \l_tmpa_tl
3610 }
3611 }
3612 \cs_generate_variant:Nn
3613 \keys_define:nn
3614 { nV }
3615 \ExplSyntaxOff

```

For complex YAML metadata, the option `jekyllDataKeyValue=<module>` [13] can



be used to route the processing of all YAML metadata in the current  $\TeX$  group to the key-values from  $\langle module \rangle$ .

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

We define the command `\@@_define_renderer_prototypes:` that defines plain  $\TeX$  macros for token renderer prototypes. Furthermore, the `\markdownSetup` macro also accepts the `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.

```

3616 \ExplSyntaxOn
3617 \cs_new:Nn \@@_define_renderer_prototypes:
3618 {
3619 \seq_map_inline:Nn
3620 \g_@@_renderers_seq
3621 {
3622 \@@_define_renderer_prototype:n
3623 { ##1 }
3624 }
3625 }
3626 \cs_new:Nn \@@_define_renderer_prototype:n
3627 {
3628 \@@_renderer_prototype_tl_to_csname:nN
3629 { #1 }
3630 \l_tmpa_tl
3631 \prop_get:NnN
3632 \g_@@_renderer_arities_prop
3633 { #1 }
3634 \l_tmpb_tl
3635 \@@_define_renderer_prototype:ncV
3636 { #1 }
3637 { \l_tmpa_tl }
3638 \l_tmpb_tl
3639 }
3640 \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
3641 {
3642 \tl_set:Nn
3643 \l_tmpa_tl
3644 { \str_uppercase:n { #1 } }
3645 \tl_set:Nx

```

```

3646 #2
3647 {
3648 markdownRenderer
3649 \tl_head:f { \l_tmpa_tl }
3650 \tl_tail:n { #1 }
3651 Prototype
3652 }
3653 }
3654 \tl_new:N
3655 \l_@@_renderer_prototype_definition_tl
3656 \bool_new:N
3657 \g_@@_appending_renderer_prototype_bool
3658 \bool_new:N
3659 \g_@@_unprotected_renderer_prototype_bool
3660 \cs_new:Nn \@@_define_renderer_prototype:nNn
3661 {
3662 \keys_define:nn
3663 { markdown/options/renderer-prototypes }
3664 {
3665 #1 .code:n = {
3666 \tl_set:Nn
3667 \l_@@_renderer_prototype_definition_tl
3668 { ##1 }
3669 \regex_replace_all:nnN
3670 { \cP\#0 }
3671 { #1 }
3672 \l_@@_renderer_prototype_definition_tl
3673 \bool_if:NT
3674 \g_@@_appending_renderer_prototype_bool
3675 {
3676 \@@_tl_set_from_cs:NNn
3677 \l_tmpa_tl
3678 #2
3679 { #3 }
3680 \tl_put_left:NV
3681 \l_@@_renderer_prototype_definition_tl
3682 \l_tmpa_tl
3683 }
3684 \bool_if:NTF
3685 \g_@@_unprotected_renderer_prototype_bool
3686 {
3687 \tl_set:Nn
3688 \l_tmpa_tl
3689 { \cs_set:Npn }
3690 }
3691 {
3692 \tl_set:Nn

```

```

3693 \l_tmpa_tl
3694 { \cs_set_protected:Npn }
3695 }
3696 \exp_last_unbraced:NNV
3697 \cs_generate_from_arg_count:NNnV
3698 #2
3699 \l_tmpa_tl
3700 { #3 }
3701 \l_@@_renderer_prototype_definition_tl
3702 },
3703 }

```

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.

```

3704 \str_if_eq:nnF
3705 { #1 }
3706 { jekyllDataString }
3707 {
3708 \cs_if_free:NT
3709 #2
3710 {
3711 \cs_generate_from_arg_count:NNnn
3712 #2
3713 \cs_gset_protected:Npn
3714 { #3 }
3715 { }
3716 }
3717 }
3718 }
3719 \cs_generate_variant:Nn
3720 \@@_define_renderer_prototype:nNn
3721 { ncV }

```

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

```

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

```

```

3722 \keys_define:nn

```

```

3723 { markdown/options/renderer-prototypes }
3724 {
3725 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 += {...},
 },
 }
 },
 },
}

```

```

3726 \regex_match:NVTF
3727 \c_@@_appending_key_regex
3728 \l_keys_key_str
3729 {
3730 \bool_gset_true:N
3731 \g_@@_appending_renderer_prototype_bool
3732 \tl_set:NV
3733 \l_tmpa_tl
3734 \l_keys_key_str
3735 \regex_replace_once:NnN
3736 \c_@@_appending_key_regex

```

```

3737 { }
3738 \l_tmpa_tl
3739 \tl_set:Nx
3740 \l_tmpb_tl
3741 { { \l_tmpa_tl } = }
3742 \tl_put_right:Nn
3743 \l_tmpb_tl
3744 { { #1 } }
3745 \keys_set:nV
3746 { markdown/options/renderer-prototypes }
3747 \l_tmpb_tl
3748 \bool_gset_false:N
3749 \g_@@_appending_renderer_prototype_bool
3750 }

```

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 = {
 *ListItem|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.
}

```

```

3751 {
3752 \@@_glob_seq:VnN
3753 \l_keys_key_str
3754 { g_@@_renderers_seq }
3755 \l_@@_renderer_glob_results_seq
3756 \seq_if_empty:NTF
3757 \l_@@_renderer_glob_results_seq
3758 {
3759 \msg_error:nnV
3760 { markdown }
3761 { undefined-renderer-prototype }
3762 \l_keys_key_str
3763 }
3764 {
3765 \tl_set:Nn
3766 \l_@@_renderer_glob_definition_tl
3767 { \exp_not:n { #1 } }
3768 \seq_map_inline:Nn
3769 \l_@@_renderer_glob_results_seq
3770 {
3771 \tl_set:Nn
3772 \l_tmpa_tl
3773 { { ##1 } = }
3774 \tl_put_right:Nx
3775 \l_tmpa_tl
3776 { { \l_@@_renderer_glob_definition_tl } }
3777 \keys_set:nV
3778 { markdown/options/renderer-prototypes }
3779 \l_tmpa_tl
3780 }
3781 }
3782 }
3783 },
3784 }
3785 \msg_new:nnn
3786 { markdown }
3787 { undefined-renderer-prototype }
3788 {
3789 Renderer~prototype~#1~is~undefined.
3790 }
3791 \@@_with_various_cases:nn
3792 { rendererPrototypes }
3793 {
3794 \keys_define:nn
3795 { markdown/options }
3796 {
3797 #1 .code:n = {

```

```

3798 \bool_gset_false:N
3799 \g_@@_unprotected_renderer_prototype_bool
3800 \keys_set:nn
3801 { markdown/options/renderer-prototypes }
3802 { ##1 }
3803 },
3804 }
3805 }
3806 \@@_with_various_cases:nn
3807 { unprotectedRendererPrototypes }
3808 {
3809 \keys_define:nn
3810 { markdown/options }
3811 {
3812 #1 .code:n = {
3813 \bool_gset_true:N
3814 \g_@@_unprotected_renderer_prototype_bool
3815 \keys_set:nn
3816 { markdown/options/renderer-prototypes }
3817 { ##1 }
3818 },
3819 }
3820 }

```

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.

```

3821 \str_if_eq:VVT
3822 \c_@@_top_layer_tl
3823 \c_@@_option_layer_plain_tex_tl
3824 {
3825 \@@_define_renderer_prototypes:
3826 }
3827 \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*.

```
3828 \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.

```
3829 \let\markdownReadAndConvert\relax
3830 \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*.

```
3831 \catcode`\|=0\catcode`\=12%
3832 |gdef|markdownBegin{%
3833 |markdownReadAndConvert{\markdownEnd}%
3834 {|markdownEnd}}%
3835 |gdef|yamlBegin{%
3836 |begingroup
3837 |yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
3838 |markdownReadAndConvert{\yamlEnd}%
3839 {|yamlEnd}}%
3840 |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.

```
3841 \ExplSyntaxOn
3842 \keys_define:nn
3843 { markdown/options }
3844 {
3845 code .code:n = { #1 },
3846 }
3847 \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.

```
3848 \ExplSyntaxOn
3849 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
3850 \cs_generate_variant:Nn
3851 \tl_const:Nn
3852 { NV }
3853 \tl_if_exist:NF
3854 \c_@@_top_layer_tl
3855 {
3856 \tl_const:NV
3857 \c_@@_top_layer_tl
3858 \c_@@_option_layer_latex_tl
3859 }
3860 \ExplSyntaxOff
3861 \input markdown/markdown
```

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

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

where *<options>* are the L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.3). Note that *<options>* inside the `\usepackage` macro may not set the `markdownRenderers` (see Section 2.2.5.45) and `markdownRendererPrototypes` (see Section 2.2.6.2) keys. Furthermore, although the base variant of the `import` key that loads a single L<sup>A</sup>T<sub>E</sub>X theme (see Section 2.3.4) can be used, the extended variant that can load multiple themes and import snippets from them (see Section 2.2.4) cannot. This limitation is due to the way L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> parses package options.

### 2.3.1 Typesetting Markdown

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

#### 2.3.1.1 Typesetting Markdown and YAML directly

The `markdown` and `markdown*` L<sup>A</sup>T<sub>E</sub>X environments are aliases for the macros `\markdownBegin` and `\markdownEnd` exposed by the plain T<sub>E</sub>X interface.

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

```
3862 \newenvironment{markdown}\relax\relax
3863 \newenvironment{markdown*}[1]\relax\relax
```

Furthermore, both environments accept L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.3) as the only argument. This argument is optional for the `markdown` environment and mandatory for the `markdown*` environment.

The `markdown` and `markdown*` environments are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

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

<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` L<sup>A</sup>T<sub>E</sub>X environment by using it in other environments as follows:

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

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

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

Specifically, the `\markdown` macro must appear at the end of the replacement before-text and must be followed by text that has not yet been ingested by T<sub>E</sub>X's input processor.

Furthermore, using the `\markdownEnd` macro in of after the replacement after-text is optional and only makes a difference if you redefined it to produce special effects before and after the `markdown` L<sup>A</sup>T<sub>E</sub>X environment.

Lastly, you can't nest the other environments. For example, the following definition would be incorrect:

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

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

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

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

```
3864 \newenvironment{yaml}\relax\relax
```

Furthermore, the environment accepts L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.3) as the only optional argument.

The `yaml` environment is subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

The following example L<sup>A</sup>T<sub>E</sub>X code showcases the usage of the `yaml` environment:

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\begin{yaml}[smartEllipses]
title: _Hello_ world ...
author: John Doe
\end{yaml}
\end{document}
```

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

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\begin{markdown}[
 jekyllData,
 expectJekyllData,
 ensureJekyllData,
 smartEllipses,
]
title: _Hello_ world ...
author: John Doe
\end{markdown}
\end{document}
```

You can't directly extend the `yaml` L<sup>A</sup>T<sub>E</sub>X environment by using it in other environments. However, you can work around this limitation by using the `\yaml` and `\yamlEnd` macros directly in the definition, similarly to the `\markdown` and `\markdownEnd` macros described previously. Unlike with the `\markdown` and `\markdownEnd` macros, The `\yamlEnd` macro `__must__` be used in or after the replacement after-text.

The `\markinline` macro accepts a single mandatory parameter containing inline markdown content and expands to the result of the conversion of the input markdown document to plain T<sub>E</sub>X. Unlike the `\markinline` macro provided by the plain T<sub>E</sub>X interface, this macro also accepts L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.3) as its optional argument. These options will only influence this markdown content.

### 2.3.1.2 Typesetting Markdown and YAML from external documents

The `\markdownInput` macro accepts a single mandatory parameter containing the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain T<sub>E</sub>X. Unlike the `\markdownInput` macro provided by the plain T<sub>E</sub>X interface, this macro also accepts L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.3) as its optional argument. These options will only influence this markdown document.

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

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

The `\yamlInput` macro accepts a single mandatory parameter containing the filename of a YAML document and expands to the result of the conversion of the input YAML document to plain T<sub>E</sub>X. Unlike the `\yamlInput` macro provided by the plain T<sub>E</sub>X interface, this macro also accepts L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.3) as its optional argument. These options will only influence this YAML document.

The following example L<sup>A</sup>T<sub>E</sub>X code showcases the usage of the `\yamlInput` macro:

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

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

```

\documentclass{article}
\usepackage{markdown}
\begin{document}
\markdownInput[
 jekyllData,
 expectJekyllData,
 ensureJekyllData,
 smartEllipses,
]{hello.yml}
\end{document}

```

### 2.3.2 Using L<sup>A</sup>T<sub>E</sub>X hooks with the Markdown package

L<sup>A</sup>T<sub>E</sub>X provides an intricate hook management system that allows users to insert extra material before and after certain T<sub>E</sub>X macros and L<sup>A</sup>T<sub>E</sub>X 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 T<sub>E</sub>X commands and before/after L<sup>A</sup>T<sub>E</sub>X 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 “`markdownfoo emphasis: _bar_ baz!<markdown>`”, 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  $\LaTeX$  will produce the text “`markdownfoo emphasis_bar_/emphasis baz!/markdown`”, 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  $\LaTeX$  options are represented by a comma-delimited list of  $\langle key \rangle = \langle value \rangle$  pairs. For boolean options, the  $= \langle value \rangle$  part is optional, and  $\langle key \rangle$  will be interpreted as  $\langle key \rangle = \text{true}$  if the  $= \langle value \rangle$  part has been omitted.

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

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

#### 2.3.3.1 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  $\TeX$  options, the Markdown package also recognizes these as aliases and accepts them as document class options. By passing `finalizcache` and `frozenscache` as document class options, you may conveniently control the behavior of both packages at once:

```

\documentclass[frozenscache]{article}

```

```

\usepackage{markdown,minted}
\begin{document}
\end{document}

```

We hope that other packages will support the `finalizcache` and `frozenscache` package options in the future, so that they can become a standard interface for preparing L<sup>A</sup>T<sub>E</sub>X document sources for distribution.

```

3865 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
3866 \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.

```

3867 \ExplSyntaxOn
3868 \str_if_eq:VVT
3869 \c_@@_top_layer_tl
3870 \c_@@_option_layer_latex_tl
3871 {
3872 \@@_define_option_commands_and_keyvals:
3873 \@@_define_renderers:
3874 \@@_define_renderer_prototypes:
3875 }
3876 \ExplSyntaxOff

```

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

```

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

```

### 2.3.4 Themes

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

package named `markdowntheme` $\langle$ *munged theme name* $\rangle$ .`sty` if it exists and a  $\text{\TeX}$  document named `markdowntheme` $\langle$ *munged theme name* $\rangle$ .`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}
```

```
3877 \newif\ifmarkdownLaTeXLoaded
3878 \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.

```
3879 \ExplSyntaxOn
3880 \prop_new:N
3881 \g_@@_latex_built_in_themes_prop
3882 \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.

```
3883 \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).



```

3884 \ExplSyntaxOn
3885 \str_if_eq:VVT
3886 \c_@@_top_layer_tl
3887 \c_@@_option_layer_latex_tl
3888 {
3889 \use:c
3890 { ExplSyntaxOff }
3891 \AtEndOfPackage
3892 {
3893 \@@_if_option:nF
3894 { noDefaults }
3895 {
3896 \@@_if_option:nTF
3897 { experimental }
3898 {
3899 \@@_setup:n
3900 { theme = witiko/markdown/defaults@experimental }
3901 }
3902 {
3903 \@@_setup:n
3904 { theme = witiko/markdown/defaults }
3905 }
3906 }
3907 }
3908 \use:c
3909 { ExplSyntaxOn }
3910 }
3911 \ExplSyntaxOff

```

Please, see Section 3.3.2 for implementation details of the built-in L<sup>A</sup>T<sub>E</sub>X 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.

```

3912 \ExplSyntaxOn
3913 \tl_const:Nn \c_@@_option_layer_context_tl { context }
3914 \cs_generate_variant:Nn
3915 \tl_const:Nn
3916 { NV }
3917 \tl_if_exist:NF
3918 \c_@@_top_layer_tl
3919 {
3920 \tl_const:NV
3921 \c_@@_top_layer_tl
3922 \c_@@_option_layer_context_tl

```

```

3923 }
3924 \ExplSyntaxOff

```

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

```

3925 \writestatus{loading}{ConTeXt User Module / markdown}%
3926 \startmodule[markdown]
3927 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
3928 \do#\do\^\do_do\%do\~}%
3929 \input markdown/markdown

```

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

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

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

## 2.4.1 Typesetting Markdown and YAML

The interface exposes the `\startmarkdown`, `\stopmarkdown`, `\startyaml`, `\stopyaml`, `\inputmarkdown`, and `\inputyaml` macros.

### 2.4.1.1 Typesetting Markdown and YAML directly

The `\startmarkdown` and `\stopmarkdown` macros are aliases for the macros `\markdownBegin` and `\markdownEnd` exposed by the plain TeX interface.

```

3930 \let\startmarkdown\relax
3931 \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.

```
3932 \let\startyaml\relax
3933 \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 `TeX` interface.

```
3934 \let\inputmarkdown\relax
```

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

The following example `ConTeXt` code showcases the usage of the `\inputmarkdown` macro:

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

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

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

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

```
3935 \let\inputyaml\relax
```

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

The following example `ConTeXt` code showcases the usage of the `\inputyaml` macro:

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

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

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

## 2.4.2 Options

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

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

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

```
3936 \ExplSyntaxOn
3937 \cs_new:Npn
3938 \setupmarkdown
3939 [#1]
3940 {
3941 \@@_setup:n
3942 { #1 }
3943 }
```

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

```
3944 \cs_gset_eq:NN
3945 \setupyaml
3946 \setupmarkdown
```

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

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

```
3947 \cs_new:Nn \@@_caseless:N
3948 {
3949 \regex_replace_all:nnN
3950 { ([a-z])([A-Z]) }
3951 { \1 \c { str_lowercase:n } \cB{\ \2 \cE\} }
3952 #1
3953 \tl_set:Nx
3954 #1
3955 { #1 }
3956 }
3957 \seq_gput_right:Nn \g_@@_cases_seq { @@_caseless:N }
```

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

```
3958 \str_if_eq:VVT
3959 \c_@@_top_layer_tl
3960 \c_@@_option_layer_context_tl
3961 {
3962 \@@_define_option_commands_and_keyvals:
3963 \@@_define_renderers:
```

```

3964 \@@_define_renderer_prototypes:
3965 }

```

### 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.

```

3966 \prop_new:N
3967 \g_@@_context_built_in_themes_prop
3968 \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.

```

3969 \startmodule[markdownthemewitiko_markdown_defaults]
3970 \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  $\text{\TeX}$  *token renderers* is performed by the Lua layer. The plain  $\text{\TeX}$  layer provides default definitions for the token renderers. The  $\text{\LaTeX}$  and  $\text{\ConTeXt}$  layers correct idiosyncrasies of the respective  $\text{\TeX}$  formats, and provide format-specific default definitions for the token renderers.

### 3.1 Lua Implementation

The Lua implementation implements `writer` and `reader` objects, which provide the conversion from markdown to plain  $\text{\TeX}$ , and `extensions` objects, which provide syntax extensions for the `writer` and `reader` objects.

The Lunamark Lua module implements writers for the conversion to various other formats, such as DocBook, Groff, or HTML. These were stripped from the module and the remaining markdown reader and plain  $\text{\TeX}$  writer were hidden behind the converter functions exposed by the Lua interface (see Section 2.1).

```

3971 local upper, format, length =
3972 string.upper, string.format, string.len
3973 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, Cp, any =
3974 lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
3975 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  $\text{\TeX}$  writer and the markdown reader. These functions are encapsulated in the `util` object. The functions were originally located in the `lunamark/util.lua` file in the Lunamark Lua module.

```

3976 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.

```

3977 function util.err(msg, exit_code)
3978 io.stderr:write("markdown.lua: " .. msg .. "\n")
3979 os.exit(exit_code or 1)
3980 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.

```

3981 function util.cache(dir, string, salt, transform, suffix)
3982 local digest = md5.sumhexa(string .. (salt or ""))
3983 local name = util.pathname(dir, digest .. suffix)
3984 local file = io.open(name, "r")

```

```

3985 local result = nil
3986 if file == nil then -- If no cache entry exists, create a new one.
3987 file = assert(io.open(name, "w"),
3988 [[Could not open file]] .. name .. [[for writing]])
3989 result = string
3990 if transform ~= nil then
3991 result = transform(result)
3992 end
3993 assert(file:write(result))
3994 assert(file:close())
3995 end
3996 return name, result
3997 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.

```

3998 function util.cache_verbatim(dir, string)
3999 local name = util.cache(dir, string, nil, nil, ".verbatim")
4000 return name
4001 end

```

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

```

4002 function util.table_copy(t)
4003 local u = { }
4004 for k, v in pairs(t) do u[k] = v end
4005 return setmetatable(u, getmetatable(t))
4006 end

```

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

```

4007 function util.encode_json_string(s)
4008 s = s:gsub([[\\]], [[\\]])
4009 s = s:gsub([[\"]], [[\"]])
4010 return [[\"]] .. s .. [[\"]]
4011 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].

```

4012 function util.expand_tabs_in_line(s, tabstop)
4013 local tab = tabstop or 4
4014 local corr = 0
4015 return (s:gsub(")\t", function(p)
4016 local sp = tab - (p - 1 + corr) % tab
4017 corr = corr - 1 + sp
4018 return string.rep(" ", sp)
4019 end))

```



```
4020 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.

```
4021 function util.walk(t, f)
4022 local typ = type(t)
4023 if typ == "string" then
4024 f(t)
4025 elseif typ == "table" then
4026 local i = 1
4027 local n
4028 n = t[i]
4029 while n do
4030 util.walk(n, f)
4031 i = i + 1
4032 n = t[i]
4033 end
4034 elseif typ == "function" then
4035 local ok, val = pcall(t)
4036 if ok then
4037 util.walk(val, f)
4038 end
4039 else
4040 f(tostring(t))
4041 end
4042 end
```

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

```
4043 function util.flatten(ary)
4044 local new = {}
4045 for _,v in ipairs(ary) do
4046 if type(v) == "table" then
4047 for _,w in ipairs(util.flatten(v)) do
4048 new[#new + 1] = w
4049 end
4050 else
4051 new[#new + 1] = v
4052 end
4053 end
4054 return new
4055 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.

```
4056 function util.rope_to_string(rope)
```

```

4057 local buffer = {}
4058 util.walk(rope, function(x) buffer[#buffer + 1] = x end)
4059 return table.concat(buffer)
4060 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.

```

4061 function util.rope_last(rope)
4062 if #rope == 0 then
4063 return nil
4064 else
4065 local l = rope[#rope]
4066 if type(l) == "table" then
4067 return util.rope_last(l)
4068 else
4069 return l
4070 end
4071 end
4072 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  $1 \leq i \leq \#ary$ .

```

4073 function util.intersperse(ary, x)
4074 local new = {}
4075 local l = #ary
4076 for i,v in ipairs(ary) do
4077 local n = #new
4078 new[n + 1] = v
4079 if i ~= l then
4080 new[n + 2] = x
4081 end
4082 end
4083 return new
4084 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  $1 \leq i \leq \#ary$ .

```

4085 function util.map(ary, f)
4086 local new = {}
4087 for i,v in ipairs(ary) do
4088 new[i] = f(v)
4089 end
4090 return new
4091 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.

```
4092 function util.escaper(char_escapes, string_escapes)
```

Build a string of escapable characters.

```
4093 local char_escapes_list = ""
4094 for i,_ in pairs(char_escapes) do
4095 char_escapes_list = char_escapes_list .. i
4096 end
```

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

```
4097 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.

```
4098 if string_escapes then
4099 for k,v in pairs(string_escapes) do
4100 escapable = P(k) / v + escapable
4101 end
4102 end
```

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

```
4103 local escape_string = Cs((escapable + any)^0)
```

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

```
4104 return function(s)
4105 return lpeg.match(escape_string, s)
4106 end
4107 end
```

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

```
4108 function util.pathname(dir, file)
4109 if #dir == 0 then
4110 return file
4111 else
4112 return dir .. "/" .. file
4113 end
4114 end
```

```
4113 end
4114 end
```

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

```
4115 function util.salt(options)
4116 local opt_string = {}
4117 for k, _ in pairs(defaultOptions) do
4118 local v = options[k]
4119 if type(v) == "table" then
4120 for _, i in ipairs(v) do
4121 opt_string[#opt_string+1] = k .. "=" .. tostring(i)
4122 end
4123 end
4124 end
4125 end
```

The `cacheDir` option is disregarded.

```
4123 elseif k ~= "cacheDir" then
4124 opt_string[#opt_string+1] = k .. "=" .. tostring(v)
4125 end
4126 end
4127 table.sort(opt_string)
4128 local salt = table.concat(opt_string, ",")
4129 .. "," .. metadata.version
4130 return salt
4131 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.

```
4132 function util.warning(s)
4133 io.stderr:write("Warning: " .. s .. "\n")
4134 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.

```
4135 local entities = {}
4136
4137 local character_entities = {
4138 ["Tab"] = 9,
4139 ["NewLine"] = 10,
4140 ["excl"] = 33,
4141 ["QUOT"] = 34,
4142 ["quot"] = 34,
4143 ["num"] = 35,
4144 ["dollar"] = 36,
4145 ["percent"] = 37,
```

4146 ["AMP"] = 38,  
 4147 ["amp"] = 38,  
 4148 ["apos"] = 39,  
 4149 ["lpar"] = 40,  
 4150 ["rpar"] = 41,  
 4151 ["ast"] = 42,  
 4152 ["midast"] = 42,  
 4153 ["plus"] = 43,  
 4154 ["comma"] = 44,  
 4155 ["period"] = 46,  
 4156 ["sol"] = 47,  
 4157 ["colon"] = 58,  
 4158 ["semi"] = 59,  
 4159 ["LT"] = 60,  
 4160 ["lt"] = 60,  
 4161 ["nvlt"] = {60, 8402},  
 4162 ["bne"] = {61, 8421},  
 4163 ["equals"] = 61,  
 4164 ["GT"] = 62,  
 4165 ["gt"] = 62,  
 4166 ["nvgt"] = {62, 8402},  
 4167 ["quest"] = 63,  
 4168 ["commat"] = 64,  
 4169 ["lbrack"] = 91,  
 4170 ["lsqb"] = 91,  
 4171 ["bsol"] = 92,  
 4172 ["rbrack"] = 93,  
 4173 ["rsqb"] = 93,  
 4174 ["Hat"] = 94,  
 4175 ["UnderBar"] = 95,  
 4176 ["lowbar"] = 95,  
 4177 ["DiacriticalGrave"] = 96,  
 4178 ["grave"] = 96,  
 4179 ["fjlig"] = {102, 106},  
 4180 ["lbrace"] = 123,  
 4181 ["lcub"] = 123,  
 4182 ["VerticalLine"] = 124,  
 4183 ["verbar"] = 124,  
 4184 ["vert"] = 124,  
 4185 ["rbrace"] = 125,  
 4186 ["rcub"] = 125,  
 4187 ["NonBreakingSpace"] = 160,  
 4188 ["nbsp"] = 160,  
 4189 ["iexcl"] = 161,  
 4190 ["cent"] = 162,  
 4191 ["pound"] = 163,  
 4192 ["curren"] = 164,

4193 ["yen"] = 165,  
4194 ["brvbar"] = 166,  
4195 ["sect"] = 167,  
4196 ["Dot"] = 168,  
4197 ["DoubleDot"] = 168,  
4198 ["die"] = 168,  
4199 ["uml"] = 168,  
4200 ["COPY"] = 169,  
4201 ["copy"] = 169,  
4202 ["ordf"] = 170,  
4203 ["laquo"] = 171,  
4204 ["not"] = 172,  
4205 ["shy"] = 173,  
4206 ["REG"] = 174,  
4207 ["circledR"] = 174,  
4208 ["reg"] = 174,  
4209 ["macr"] = 175,  
4210 ["strns"] = 175,  
4211 ["deg"] = 176,  
4212 ["PlusMinus"] = 177,  
4213 ["plusmn"] = 177,  
4214 ["pm"] = 177,  
4215 ["sup2"] = 178,  
4216 ["sup3"] = 179,  
4217 ["DiacriticalAcute"] = 180,  
4218 ["acute"] = 180,  
4219 ["micro"] = 181,  
4220 ["para"] = 182,  
4221 ["CenterDot"] = 183,  
4222 ["centerdot"] = 183,  
4223 ["middot"] = 183,  
4224 ["Cedilla"] = 184,  
4225 ["cedil"] = 184,  
4226 ["sup1"] = 185,  
4227 ["ordm"] = 186,  
4228 ["raquo"] = 187,  
4229 ["frac14"] = 188,  
4230 ["frac12"] = 189,  
4231 ["half"] = 189,  
4232 ["frac34"] = 190,  
4233 ["iquest"] = 191,  
4234 ["Agrave"] = 192,  
4235 ["Aacute"] = 193,  
4236 ["Acirc"] = 194,  
4237 ["Atilde"] = 195,  
4238 ["Auml"] = 196,  
4239 ["Aring"] = 197,

4240 ["angst"] = 197,  
4241 ["AElig"] = 198,  
4242 ["Ccedil"] = 199,  
4243 ["Egrave"] = 200,  
4244 ["Eacute"] = 201,  
4245 ["Ecirc"] = 202,  
4246 ["Euml"] = 203,  
4247 ["Igrave"] = 204,  
4248 ["Iacute"] = 205,  
4249 ["Icirc"] = 206,  
4250 ["Iuml"] = 207,  
4251 ["ETH"] = 208,  
4252 ["Ntilde"] = 209,  
4253 ["Ograve"] = 210,  
4254 ["Oacute"] = 211,  
4255 ["Ocirc"] = 212,  
4256 ["Otilde"] = 213,  
4257 ["Ouml"] = 214,  
4258 ["times"] = 215,  
4259 ["Oslash"] = 216,  
4260 ["Ugrave"] = 217,  
4261 ["Uacute"] = 218,  
4262 ["Ucirc"] = 219,  
4263 ["Uuml"] = 220,  
4264 ["Yacute"] = 221,  
4265 ["THORN"] = 222,  
4266 ["szlig"] = 223,  
4267 ["agrave"] = 224,  
4268 ["aacute"] = 225,  
4269 ["acirc"] = 226,  
4270 ["atilde"] = 227,  
4271 ["auml"] = 228,  
4272 ["aring"] = 229,  
4273 ["aelig"] = 230,  
4274 ["ccedil"] = 231,  
4275 ["egrave"] = 232,  
4276 ["eacute"] = 233,  
4277 ["ecirc"] = 234,  
4278 ["euml"] = 235,  
4279 ["igrave"] = 236,  
4280 ["iacute"] = 237,  
4281 ["icirc"] = 238,  
4282 ["iuml"] = 239,  
4283 ["eth"] = 240,  
4284 ["ntilde"] = 241,  
4285 ["ograve"] = 242,  
4286 ["oacute"] = 243,

4287 ["ocirc"] = 244,  
4288 ["otilde"] = 245,  
4289 ["ouml"] = 246,  
4290 ["div"] = 247,  
4291 ["divide"] = 247,  
4292 ["oslash"] = 248,  
4293 ["ugrave"] = 249,  
4294 ["uacute"] = 250,  
4295 ["ucirc"] = 251,  
4296 ["uuml"] = 252,  
4297 ["yacute"] = 253,  
4298 ["thorn"] = 254,  
4299 ["yuml"] = 255,  
4300 ["Amacr"] = 256,  
4301 ["amacr"] = 257,  
4302 ["Abreve"] = 258,  
4303 ["abreve"] = 259,  
4304 ["Aogon"] = 260,  
4305 ["aogon"] = 261,  
4306 ["Cacute"] = 262,  
4307 ["cacute"] = 263,  
4308 ["Ccirc"] = 264,  
4309 ["ccirc"] = 265,  
4310 ["Cdot"] = 266,  
4311 ["cdot"] = 267,  
4312 ["Ccaron"] = 268,  
4313 ["ccaron"] = 269,  
4314 ["Dcaron"] = 270,  
4315 ["dcaron"] = 271,  
4316 ["Dstrok"] = 272,  
4317 ["dstrok"] = 273,  
4318 ["Emacr"] = 274,  
4319 ["emacr"] = 275,  
4320 ["Edot"] = 278,  
4321 ["edot"] = 279,  
4322 ["Eogon"] = 280,  
4323 ["eogon"] = 281,  
4324 ["Ecaron"] = 282,  
4325 ["ecaron"] = 283,  
4326 ["Gcirc"] = 284,  
4327 ["gcirc"] = 285,  
4328 ["Gbreve"] = 286,  
4329 ["gbreve"] = 287,  
4330 ["Gdot"] = 288,  
4331 ["gdot"] = 289,  
4332 ["Gcedil"] = 290,  
4333 ["Hcirc"] = 292,



4334 ["hcirc"] = 293,  
4335 ["Hstrokh"] = 294,  
4336 ["hstrokh"] = 295,  
4337 ["Itilde"] = 296,  
4338 ["itilde"] = 297,  
4339 ["Imacr"] = 298,  
4340 ["imacr"] = 299,  
4341 ["Iogon"] = 302,  
4342 ["iogon"] = 303,  
4343 ["Idot"] = 304,  
4344 ["imath"] = 305,  
4345 ["inodot"] = 305,  
4346 ["IJlig"] = 306,  
4347 ["ijlig"] = 307,  
4348 ["Jcirc"] = 308,  
4349 ["jcirc"] = 309,  
4350 ["Kcedil"] = 310,  
4351 ["kcedil"] = 311,  
4352 ["kgreen"] = 312,  
4353 ["Lacute"] = 313,  
4354 ["lacute"] = 314,  
4355 ["Lcedil"] = 315,  
4356 ["lcedil"] = 316,  
4357 ["Lcaron"] = 317,  
4358 ["lcaron"] = 318,  
4359 ["Lmidot"] = 319,  
4360 ["lmidot"] = 320,  
4361 ["Lstrokh"] = 321,  
4362 ["lstrokh"] = 322,  
4363 ["Nacute"] = 323,  
4364 ["nacute"] = 324,  
4365 ["Ncedil"] = 325,  
4366 ["ncedil"] = 326,  
4367 ["Ncaron"] = 327,  
4368 ["ncaron"] = 328,  
4369 ["napos"] = 329,  
4370 ["ENG"] = 330,  
4371 ["eng"] = 331,  
4372 ["Omacr"] = 332,  
4373 ["omacr"] = 333,  
4374 ["Odblac"] = 336,  
4375 ["odblac"] = 337,  
4376 ["OElig"] = 338,  
4377 ["oelig"] = 339,  
4378 ["Racute"] = 340,  
4379 ["racute"] = 341,  
4380 ["Rcedil"] = 342,

4381 ["rcedil"] = 343,  
4382 ["Rcaron"] = 344,  
4383 ["rcaron"] = 345,  
4384 ["Sacute"] = 346,  
4385 ["sacute"] = 347,  
4386 ["Scirc"] = 348,  
4387 ["scirc"] = 349,  
4388 ["Scedil"] = 350,  
4389 ["scedil"] = 351,  
4390 ["Scaron"] = 352,  
4391 ["scaron"] = 353,  
4392 ["Tcedil"] = 354,  
4393 ["tcedil"] = 355,  
4394 ["Tcaron"] = 356,  
4395 ["tcaron"] = 357,  
4396 ["Tstrok"] = 358,  
4397 ["tstrok"] = 359,  
4398 ["Utilde"] = 360,  
4399 ["utilde"] = 361,  
4400 ["Umacr"] = 362,  
4401 ["umacr"] = 363,  
4402 ["Ubreve"] = 364,  
4403 ["ubreve"] = 365,  
4404 ["Uring"] = 366,  
4405 ["uring"] = 367,  
4406 ["Udblac"] = 368,  
4407 ["udblac"] = 369,  
4408 ["Uogon"] = 370,  
4409 ["uogon"] = 371,  
4410 ["Wcirc"] = 372,  
4411 ["wcirc"] = 373,  
4412 ["Ycirc"] = 374,  
4413 ["ycirc"] = 375,  
4414 ["Yuml"] = 376,  
4415 ["Zacute"] = 377,  
4416 ["zacute"] = 378,  
4417 ["Zdot"] = 379,  
4418 ["zdot"] = 380,  
4419 ["Zcaron"] = 381,  
4420 ["zcaron"] = 382,  
4421 ["fnof"] = 402,  
4422 ["imped"] = 437,  
4423 ["gacute"] = 501,  
4424 ["jmath"] = 567,  
4425 ["circ"] = 710,  
4426 ["Hacek"] = 711,  
4427 ["caron"] = 711,

4428 ["Breve"] = 728,  
4429 ["breve"] = 728,  
4430 ["DiacriticalDot"] = 729,  
4431 ["dot"] = 729,  
4432 ["ring"] = 730,  
4433 ["ogon"] = 731,  
4434 ["DiacriticalTilde"] = 732,  
4435 ["tilde"] = 732,  
4436 ["DiacriticalDoubleAcute"] = 733,  
4437 ["dblac"] = 733,  
4438 ["DownBreve"] = 785,  
4439 ["Alpha"] = 913,  
4440 ["Beta"] = 914,  
4441 ["Gamma"] = 915,  
4442 ["Delta"] = 916,  
4443 ["Epsilon"] = 917,  
4444 ["Zeta"] = 918,  
4445 ["Eta"] = 919,  
4446 ["Theta"] = 920,  
4447 ["Iota"] = 921,  
4448 ["Kappa"] = 922,  
4449 ["Lambda"] = 923,  
4450 ["Mu"] = 924,  
4451 ["Nu"] = 925,  
4452 ["Xi"] = 926,  
4453 ["Omicron"] = 927,  
4454 ["Pi"] = 928,  
4455 ["Rho"] = 929,  
4456 ["Sigma"] = 931,  
4457 ["Tau"] = 932,  
4458 ["Upsilon"] = 933,  
4459 ["Phi"] = 934,  
4460 ["Chi"] = 935,  
4461 ["Psi"] = 936,  
4462 ["Omega"] = 937,  
4463 ["ohm"] = 937,  
4464 ["alpha"] = 945,  
4465 ["beta"] = 946,  
4466 ["gamma"] = 947,  
4467 ["delta"] = 948,  
4468 ["epsi"] = 949,  
4469 ["epsilon"] = 949,  
4470 ["zeta"] = 950,  
4471 ["eta"] = 951,  
4472 ["theta"] = 952,  
4473 ["iota"] = 953,  
4474 ["kappa"] = 954,

4475 ["lambda"] = 955,  
4476 ["mu"] = 956,  
4477 ["nu"] = 957,  
4478 ["xi"] = 958,  
4479 ["omicron"] = 959,  
4480 ["pi"] = 960,  
4481 ["rho"] = 961,  
4482 ["sigmaf"] = 962,  
4483 ["sigmav"] = 962,  
4484 ["varsigma"] = 962,  
4485 ["sigma"] = 963,  
4486 ["tau"] = 964,  
4487 ["upsilon"] = 965,  
4488 ["upsilon"] = 965,  
4489 ["phi"] = 966,  
4490 ["chi"] = 967,  
4491 ["psi"] = 968,  
4492 ["omega"] = 969,  
4493 ["thetasym"] = 977,  
4494 ["thetav"] = 977,  
4495 ["vartheta"] = 977,  
4496 ["Upsilon"] = 978,  
4497 ["upsih"] = 978,  
4498 ["phiv"] = 981,  
4499 ["straightphi"] = 981,  
4500 ["varphi"] = 981,  
4501 ["piv"] = 982,  
4502 ["varpi"] = 982,  
4503 ["Gammad"] = 988,  
4504 ["digamma"] = 989,  
4505 ["gammad"] = 989,  
4506 ["kappav"] = 1008,  
4507 ["varkappa"] = 1008,  
4508 ["rhov"] = 1009,  
4509 ["varrho"] = 1009,  
4510 ["epsiv"] = 1013,  
4511 ["straightepsilon"] = 1013,  
4512 ["varepsilon"] = 1013,  
4513 ["backepsilon"] = 1014,  
4514 ["bepsi"] = 1014,  
4515 ["IOcy"] = 1025,  
4516 ["DJcy"] = 1026,  
4517 ["GJcy"] = 1027,  
4518 ["Jukcy"] = 1028,  
4519 ["DScy"] = 1029,  
4520 ["Iukcy"] = 1030,  
4521 ["YIcy"] = 1031,

4522 ["Jsercy"] = 1032,  
4523 ["LJcy"] = 1033,  
4524 ["NJcy"] = 1034,  
4525 ["TSHcy"] = 1035,  
4526 ["KJcy"] = 1036,  
4527 ["Ubrcy"] = 1038,  
4528 ["DZcy"] = 1039,  
4529 ["Acy"] = 1040,  
4530 ["Bcy"] = 1041,  
4531 ["Vcy"] = 1042,  
4532 ["Gcy"] = 1043,  
4533 ["Dcy"] = 1044,  
4534 ["IEcy"] = 1045,  
4535 ["ZHcy"] = 1046,  
4536 ["Zcy"] = 1047,  
4537 ["Icy"] = 1048,  
4538 ["Jcy"] = 1049,  
4539 ["Kcy"] = 1050,  
4540 ["Lcy"] = 1051,  
4541 ["Mcy"] = 1052,  
4542 ["Ncy"] = 1053,  
4543 ["Ocy"] = 1054,  
4544 ["Pcy"] = 1055,  
4545 ["Rcy"] = 1056,  
4546 ["Scy"] = 1057,  
4547 ["Tcy"] = 1058,  
4548 ["Ucy"] = 1059,  
4549 ["Fcy"] = 1060,  
4550 ["KHcy"] = 1061,  
4551 ["TScy"] = 1062,  
4552 ["CHcy"] = 1063,  
4553 ["SHcy"] = 1064,  
4554 ["SHCHcy"] = 1065,  
4555 ["HARDcy"] = 1066,  
4556 ["Ycy"] = 1067,  
4557 ["SOFTcy"] = 1068,  
4558 ["Ecy"] = 1069,  
4559 ["YUcy"] = 1070,  
4560 ["YAcy"] = 1071,  
4561 ["acy"] = 1072,  
4562 ["bcy"] = 1073,  
4563 ["vcy"] = 1074,  
4564 ["gcy"] = 1075,  
4565 ["dcy"] = 1076,  
4566 ["iecy"] = 1077,  
4567 ["zhcy"] = 1078,  
4568 ["zcy"] = 1079,

4569 ["icy"] = 1080,  
4570 ["jcy"] = 1081,  
4571 ["kcy"] = 1082,  
4572 ["lcy"] = 1083,  
4573 ["mcy"] = 1084,  
4574 ["ncy"] = 1085,  
4575 ["ocy"] = 1086,  
4576 ["pcy"] = 1087,  
4577 ["rcy"] = 1088,  
4578 ["scy"] = 1089,  
4579 ["tcy"] = 1090,  
4580 ["ucy"] = 1091,  
4581 ["fcy"] = 1092,  
4582 ["khcy"] = 1093,  
4583 ["tscy"] = 1094,  
4584 ["chcy"] = 1095,  
4585 ["shcy"] = 1096,  
4586 ["shchcy"] = 1097,  
4587 ["hardcy"] = 1098,  
4588 ["ycy"] = 1099,  
4589 ["softcy"] = 1100,  
4590 ["ecy"] = 1101,  
4591 ["yucy"] = 1102,  
4592 ["yacy"] = 1103,  
4593 ["iocy"] = 1105,  
4594 ["djcy"] = 1106,  
4595 ["gjcy"] = 1107,  
4596 ["jukcy"] = 1108,  
4597 ["dscy"] = 1109,  
4598 ["iukcy"] = 1110,  
4599 ["yicy"] = 1111,  
4600 ["jsercy"] = 1112,  
4601 ["ljcy"] = 1113,  
4602 ["njcy"] = 1114,  
4603 ["tshcy"] = 1115,  
4604 ["kjcy"] = 1116,  
4605 ["ubrcy"] = 1118,  
4606 ["dzcy"] = 1119,  
4607 ["ensp"] = 8194,  
4608 ["emsp"] = 8195,  
4609 ["emsp13"] = 8196,  
4610 ["emsp14"] = 8197,  
4611 ["numsp"] = 8199,  
4612 ["puncsp"] = 8200,  
4613 ["ThinSpace"] = 8201,  
4614 ["thinsp"] = 8201,  
4615 ["VeryThinSpace"] = 8202,

4616 ["hairsp"] = 8202,  
4617 ["NegativeMediumSpace"] = 8203,  
4618 ["NegativeThickSpace"] = 8203,  
4619 ["NegativeThinSpace"] = 8203,  
4620 ["NegativeVeryThinSpace"] = 8203,  
4621 ["ZeroWidthSpace"] = 8203,  
4622 ["zwnj"] = 8204,  
4623 ["zwj"] = 8205,  
4624 ["lrm"] = 8206,  
4625 ["rlm"] = 8207,  
4626 ["dash"] = 8208,  
4627 ["hyphen"] = 8208,  
4628 ["ndash"] = 8211,  
4629 ["mdash"] = 8212,  
4630 ["horbar"] = 8213,  
4631 ["Verbar"] = 8214,  
4632 ["Vert"] = 8214,  
4633 ["OpenCurlyQuote"] = 8216,  
4634 ["lsquo"] = 8216,  
4635 ["CloseCurlyQuote"] = 8217,  
4636 ["rsquo"] = 8217,  
4637 ["rsquor"] = 8217,  
4638 ["lsquor"] = 8218,  
4639 ["sbquo"] = 8218,  
4640 ["OpenCurlyDoubleQuote"] = 8220,  
4641 ["ldquo"] = 8220,  
4642 ["CloseCurlyDoubleQuote"] = 8221,  
4643 ["rdquo"] = 8221,  
4644 ["rdquor"] = 8221,  
4645 ["bdquo"] = 8222,  
4646 ["ldquor"] = 8222,  
4647 ["dagger"] = 8224,  
4648 ["Dagger"] = 8225,  
4649 ["ddagger"] = 8225,  
4650 ["bull"] = 8226,  
4651 ["bullet"] = 8226,  
4652 ["nldr"] = 8229,  
4653 ["hellip"] = 8230,  
4654 ["mldr"] = 8230,  
4655 ["permil"] = 8240,  
4656 ["pertenk"] = 8241,  
4657 ["prime"] = 8242,  
4658 ["Prime"] = 8243,  
4659 ["tprime"] = 8244,  
4660 ["backprime"] = 8245,  
4661 ["bprime"] = 8245,  
4662 ["lsaquo"] = 8249,

4663 ["rsaquo"] = 8250,  
4664 ["OverBar"] = 8254,  
4665 ["oline"] = 8254,  
4666 ["caret"] = 8257,  
4667 ["hybull"] = 8259,  
4668 ["frasl"] = 8260,  
4669 ["bsemi"] = 8271,  
4670 ["qprime"] = 8279,  
4671 ["MediumSpace"] = 8287,  
4672 ["ThickSpace"] = {8287, 8202},  
4673 ["NoBreak"] = 8288,  
4674 ["ApplyFunction"] = 8289,  
4675 ["af"] = 8289,  
4676 ["InvisibleTimes"] = 8290,  
4677 ["it"] = 8290,  
4678 ["InvisibleComma"] = 8291,  
4679 ["ic"] = 8291,  
4680 ["euro"] = 8364,  
4681 ["TripleDot"] = 8411,  
4682 ["tdot"] = 8411,  
4683 ["DotDot"] = 8412,  
4684 ["Copf"] = 8450,  
4685 ["complexes"] = 8450,  
4686 ["incare"] = 8453,  
4687 ["gscr"] = 8458,  
4688 ["HilbertSpace"] = 8459,  
4689 ["Hscr"] = 8459,  
4690 ["hamilt"] = 8459,  
4691 ["Hfr"] = 8460,  
4692 ["Poincareplane"] = 8460,  
4693 ["Hopf"] = 8461,  
4694 ["quaternions"] = 8461,  
4695 ["planckh"] = 8462,  
4696 ["hbar"] = 8463,  
4697 ["hslash"] = 8463,  
4698 ["planck"] = 8463,  
4699 ["plankv"] = 8463,  
4700 ["Iscr"] = 8464,  
4701 ["imagline"] = 8464,  
4702 ["Ifr"] = 8465,  
4703 ["Im"] = 8465,  
4704 ["image"] = 8465,  
4705 ["imagpart"] = 8465,  
4706 ["Laplacetrif"] = 8466,  
4707 ["Lscr"] = 8466,  
4708 ["lagran"] = 8466,  
4709 ["ell"] = 8467,



4710 ["Nopf"] = 8469,  
4711 ["naturals"] = 8469,  
4712 ["numero"] = 8470,  
4713 ["copysr"] = 8471,  
4714 ["weierp"] = 8472,  
4715 ["wp"] = 8472,  
4716 ["Popf"] = 8473,  
4717 ["primes"] = 8473,  
4718 ["Qopf"] = 8474,  
4719 ["rationals"] = 8474,  
4720 ["Rscr"] = 8475,  
4721 ["realine"] = 8475,  
4722 ["Re"] = 8476,  
4723 ["Rfr"] = 8476,  
4724 ["real"] = 8476,  
4725 ["realpart"] = 8476,  
4726 ["Ropf"] = 8477,  
4727 ["reals"] = 8477,  
4728 ["rx"] = 8478,  
4729 ["TRADE"] = 8482,  
4730 ["trade"] = 8482,  
4731 ["Zopf"] = 8484,  
4732 ["integers"] = 8484,  
4733 ["mho"] = 8487,  
4734 ["Zfr"] = 8488,  
4735 ["zeetrf"] = 8488,  
4736 ["iiota"] = 8489,  
4737 ["Bernoullis"] = 8492,  
4738 ["Bscr"] = 8492,  
4739 ["bernou"] = 8492,  
4740 ["Cayleys"] = 8493,  
4741 ["Cfr"] = 8493,  
4742 ["escr"] = 8495,  
4743 ["Escr"] = 8496,  
4744 ["expectation"] = 8496,  
4745 ["Fouriertrf"] = 8497,  
4746 ["Fscr"] = 8497,  
4747 ["Mellintrf"] = 8499,  
4748 ["Mscr"] = 8499,  
4749 ["phmmat"] = 8499,  
4750 ["order"] = 8500,  
4751 ["orderof"] = 8500,  
4752 ["oscr"] = 8500,  
4753 ["alefsym"] = 8501,  
4754 ["aleph"] = 8501,  
4755 ["beth"] = 8502,  
4756 ["gimel"] = 8503,

4757 ["daleth"] = 8504,  
 4758 ["CapitalDifferentialD"] = 8517,  
 4759 ["DD"] = 8517,  
 4760 ["DifferentialD"] = 8518,  
 4761 ["dd"] = 8518,  
 4762 ["ExponentialE"] = 8519,  
 4763 ["ee"] = 8519,  
 4764 ["exponentiale"] = 8519,  
 4765 ["ImaginaryI"] = 8520,  
 4766 ["ii"] = 8520,  
 4767 ["frac13"] = 8531,  
 4768 ["frac23"] = 8532,  
 4769 ["frac15"] = 8533,  
 4770 ["frac25"] = 8534,  
 4771 ["frac35"] = 8535,  
 4772 ["frac45"] = 8536,  
 4773 ["frac16"] = 8537,  
 4774 ["frac56"] = 8538,  
 4775 ["frac18"] = 8539,  
 4776 ["frac38"] = 8540,  
 4777 ["frac58"] = 8541,  
 4778 ["frac78"] = 8542,  
 4779 ["LeftArrow"] = 8592,  
 4780 ["ShortLeftArrow"] = 8592,  
 4781 ["larr"] = 8592,  
 4782 ["leftarrow"] = 8592,  
 4783 ["slarr"] = 8592,  
 4784 ["ShortUpArrow"] = 8593,  
 4785 ["UpArrow"] = 8593,  
 4786 ["uarr"] = 8593,  
 4787 ["uparrow"] = 8593,  
 4788 ["RightArrow"] = 8594,  
 4789 ["ShortRightArrow"] = 8594,  
 4790 ["rarr"] = 8594,  
 4791 ["rightarrow"] = 8594,  
 4792 ["srarr"] = 8594,  
 4793 ["DownArrow"] = 8595,  
 4794 ["ShortDownArrow"] = 8595,  
 4795 ["darr"] = 8595,  
 4796 ["downarrow"] = 8595,  
 4797 ["LeftRightArrow"] = 8596,  
 4798 ["harr"] = 8596,  
 4799 ["leftrightarrow"] = 8596,  
 4800 ["UpDownArrow"] = 8597,  
 4801 ["updownarrow"] = 8597,  
 4802 ["varr"] = 8597,  
 4803 ["UpperLeftArrow"] = 8598,

4804 ["nwarr"] = 8598,  
 4805 ["nwarrow"] = 8598,  
 4806 ["UpperRightArrow"] = 8599,  
 4807 ["nearr"] = 8599,  
 4808 ["nearrow"] = 8599,  
 4809 ["LowerRightArrow"] = 8600,  
 4810 ["searr"] = 8600,  
 4811 ["searrow"] = 8600,  
 4812 ["LowerLeftArrow"] = 8601,  
 4813 ["swarr"] = 8601,  
 4814 ["swarrow"] = 8601,  
 4815 ["nlarr"] = 8602,  
 4816 ["nleftarrow"] = 8602,  
 4817 ["nrarr"] = 8603,  
 4818 ["nrightarrow"] = 8603,  
 4819 ["nrarrw"] = {8605, 824},  
 4820 ["rarrw"] = 8605,  
 4821 ["rightsquigarrow"] = 8605,  
 4822 ["Larr"] = 8606,  
 4823 ["twoheadleftarrow"] = 8606,  
 4824 ["Uarr"] = 8607,  
 4825 ["Rarr"] = 8608,  
 4826 ["twoheadrightarrow"] = 8608,  
 4827 ["Darr"] = 8609,  
 4828 ["larrtl"] = 8610,  
 4829 ["leftarrowtail"] = 8610,  
 4830 ["rarrtl"] = 8611,  
 4831 ["rightarrowtail"] = 8611,  
 4832 ["LeftTeeArrow"] = 8612,  
 4833 ["mapstoleft"] = 8612,  
 4834 ["UpTeeArrow"] = 8613,  
 4835 ["mapstoup"] = 8613,  
 4836 ["RightTeeArrow"] = 8614,  
 4837 ["map"] = 8614,  
 4838 ["mapsto"] = 8614,  
 4839 ["DownTeeArrow"] = 8615,  
 4840 ["mapstodown"] = 8615,  
 4841 ["hookleftarrow"] = 8617,  
 4842 ["larrhk"] = 8617,  
 4843 ["hookrightarrow"] = 8618,  
 4844 ["rarrhk"] = 8618,  
 4845 ["larrlp"] = 8619,  
 4846 ["looparrowleft"] = 8619,  
 4847 ["looparrowright"] = 8620,  
 4848 ["rarrlp"] = 8620,  
 4849 ["harrw"] = 8621,  
 4850 ["leftrightsquigarrow"] = 8621,

4851 ["nharr"] = 8622,  
 4852 ["nleftrightharrow"] = 8622,  
 4853 ["Lsh"] = 8624,  
 4854 ["lsh"] = 8624,  
 4855 ["Rsh"] = 8625,  
 4856 ["rsh"] = 8625,  
 4857 ["ldsh"] = 8626,  
 4858 ["rdsh"] = 8627,  
 4859 ["crarr"] = 8629,  
 4860 ["cularr"] = 8630,  
 4861 ["curvearrowleft"] = 8630,  
 4862 ["curarr"] = 8631,  
 4863 ["curvearrowright"] = 8631,  
 4864 ["circlearrowleft"] = 8634,  
 4865 ["olarr"] = 8634,  
 4866 ["circlearrowright"] = 8635,  
 4867 ["orarr"] = 8635,  
 4868 ["LeftVector"] = 8636,  
 4869 ["leftharpoonup"] = 8636,  
 4870 ["lharu"] = 8636,  
 4871 ["DownLeftVector"] = 8637,  
 4872 ["leftharpoondown"] = 8637,  
 4873 ["lhard"] = 8637,  
 4874 ["RightUpVector"] = 8638,  
 4875 ["uharr"] = 8638,  
 4876 ["upharpoonright"] = 8638,  
 4877 ["LeftUpVector"] = 8639,  
 4878 ["uharl"] = 8639,  
 4879 ["upharpoonleft"] = 8639,  
 4880 ["RightVector"] = 8640,  
 4881 ["rharu"] = 8640,  
 4882 ["rightharpoonup"] = 8640,  
 4883 ["DownRightVector"] = 8641,  
 4884 ["rhard"] = 8641,  
 4885 ["rightharpoondown"] = 8641,  
 4886 ["RightDownVector"] = 8642,  
 4887 ["dharr"] = 8642,  
 4888 ["downharpoonright"] = 8642,  
 4889 ["LeftDownVector"] = 8643,  
 4890 ["dharl"] = 8643,  
 4891 ["downharpoonleft"] = 8643,  
 4892 ["RightArrowLeftArrow"] = 8644,  
 4893 ["rightleftarrows"] = 8644,  
 4894 ["rlarr"] = 8644,  
 4895 ["UpArrowDownArrow"] = 8645,  
 4896 ["udarr"] = 8645,  
 4897 ["LeftArrowRightArrow"] = 8646,

4898 ["leftrightharpoons"] = 8646,  
 4899 ["larr"] = 8646,  
 4900 ["leftleftarrows"] = 8647,  
 4901 ["llarr"] = 8647,  
 4902 ["uparrows"] = 8648,  
 4903 ["uarr"] = 8648,  
 4904 ["rightrightarrows"] = 8649,  
 4905 ["rrarr"] = 8649,  
 4906 ["ddarr"] = 8650,  
 4907 ["downdownarrows"] = 8650,  
 4908 ["ReverseEquilibrium"] = 8651,  
 4909 ["leftrightharpoons"] = 8651,  
 4910 ["lrhar"] = 8651,  
 4911 ["Equilibrium"] = 8652,  
 4912 ["rightleftharpoons"] = 8652,  
 4913 ["rlhar"] = 8652,  
 4914 ["nLeftarrow"] = 8653,  
 4915 ["nlArr"] = 8653,  
 4916 ["nLeftrightarrow"] = 8654,  
 4917 ["nhArr"] = 8654,  
 4918 ["nRightarrow"] = 8655,  
 4919 ["nrArr"] = 8655,  
 4920 ["DoubleLeftArrow"] = 8656,  
 4921 ["Leftarrow"] = 8656,  
 4922 ["lArr"] = 8656,  
 4923 ["DoubleUpArrow"] = 8657,  
 4924 ["Uparrow"] = 8657,  
 4925 ["uArr"] = 8657,  
 4926 ["DoubleRightArrow"] = 8658,  
 4927 ["Implies"] = 8658,  
 4928 ["Rightarrow"] = 8658,  
 4929 ["rArr"] = 8658,  
 4930 ["DoubleDownArrow"] = 8659,  
 4931 ["Downarrow"] = 8659,  
 4932 ["dArr"] = 8659,  
 4933 ["DoubleLeftRightArrow"] = 8660,  
 4934 ["Leftrightarrow"] = 8660,  
 4935 ["hArr"] = 8660,  
 4936 ["iff"] = 8660,  
 4937 ["DoubleUpDownArrow"] = 8661,  
 4938 ["Updownarrow"] = 8661,  
 4939 ["vArr"] = 8661,  
 4940 ["nwArr"] = 8662,  
 4941 ["neArr"] = 8663,  
 4942 ["seArr"] = 8664,  
 4943 ["swArr"] = 8665,  
 4944 ["Lleftarrow"] = 8666,

4945 ["lAarr"] = 8666,  
 4946 ["Rightarrow"] = 8667,  
 4947 ["rAarr"] = 8667,  
 4948 ["zigrarr"] = 8669,  
 4949 ["LeftArrowBar"] = 8676,  
 4950 ["larrb"] = 8676,  
 4951 ["RightArrowBar"] = 8677,  
 4952 ["rarrb"] = 8677,  
 4953 ["DownArrowUpArrow"] = 8693,  
 4954 ["duarr"] = 8693,  
 4955 ["loarr"] = 8701,  
 4956 ["roarr"] = 8702,  
 4957 ["hoarr"] = 8703,  
 4958 ["ForAll"] = 8704,  
 4959 ["forall"] = 8704,  
 4960 ["comp"] = 8705,  
 4961 ["complement"] = 8705,  
 4962 ["PartialD"] = 8706,  
 4963 ["npart"] = {8706, 824},  
 4964 ["part"] = 8706,  
 4965 ["Exists"] = 8707,  
 4966 ["exist"] = 8707,  
 4967 ["NotExists"] = 8708,  
 4968 ["nexist"] = 8708,  
 4969 ["nexists"] = 8708,  
 4970 ["empty"] = 8709,  
 4971 ["emptyset"] = 8709,  
 4972 ["emptyv"] = 8709,  
 4973 ["varnothing"] = 8709,  
 4974 ["Del"] = 8711,  
 4975 ["nabla"] = 8711,  
 4976 ["Element"] = 8712,  
 4977 ["in"] = 8712,  
 4978 ["isin"] = 8712,  
 4979 ["isinv"] = 8712,  
 4980 ["NotElement"] = 8713,  
 4981 ["notin"] = 8713,  
 4982 ["notinva"] = 8713,  
 4983 ["ReverseElement"] = 8715,  
 4984 ["SuchThat"] = 8715,  
 4985 ["ni"] = 8715,  
 4986 ["niv"] = 8715,  
 4987 ["NotReverseElement"] = 8716,  
 4988 ["notni"] = 8716,  
 4989 ["notniva"] = 8716,  
 4990 ["Product"] = 8719,  
 4991 ["prod"] = 8719,

4992 ["Coproduct"] = 8720,  
 4993 ["coprod"] = 8720,  
 4994 ["Sum"] = 8721,  
 4995 ["sum"] = 8721,  
 4996 ["minus"] = 8722,  
 4997 ["MinusPlus"] = 8723,  
 4998 ["mnplus"] = 8723,  
 4999 ["mp"] = 8723,  
 5000 ["dotplus"] = 8724,  
 5001 ["plusdo"] = 8724,  
 5002 ["Backslash"] = 8726,  
 5003 ["setminus"] = 8726,  
 5004 ["setmn"] = 8726,  
 5005 ["smallsetminus"] = 8726,  
 5006 ["ssetmn"] = 8726,  
 5007 ["lowast"] = 8727,  
 5008 ["SmallCircle"] = 8728,  
 5009 ["compfn"] = 8728,  
 5010 ["Sqrt"] = 8730,  
 5011 ["radic"] = 8730,  
 5012 ["Proportional"] = 8733,  
 5013 ["prop"] = 8733,  
 5014 ["propto"] = 8733,  
 5015 ["varpropto"] = 8733,  
 5016 ["vprop"] = 8733,  
 5017 ["infin"] = 8734,  
 5018 ["angrt"] = 8735,  
 5019 ["ang"] = 8736,  
 5020 ["angle"] = 8736,  
 5021 ["nang"] = {8736, 8402},  
 5022 ["angmsd"] = 8737,  
 5023 ["measuredangle"] = 8737,  
 5024 ["angsph"] = 8738,  
 5025 ["VerticalBar"] = 8739,  
 5026 ["mid"] = 8739,  
 5027 ["shortmid"] = 8739,  
 5028 ["smid"] = 8739,  
 5029 ["NotVerticalBar"] = 8740,  
 5030 ["nmid"] = 8740,  
 5031 ["nshortmid"] = 8740,  
 5032 ["nsmid"] = 8740,  
 5033 ["DoubleVerticalBar"] = 8741,  
 5034 ["par"] = 8741,  
 5035 ["parallel"] = 8741,  
 5036 ["shortparallel"] = 8741,  
 5037 ["spar"] = 8741,  
 5038 ["NotDoubleVerticalBar"] = 8742,

5039 ["npar"] = 8742,  
5040 ["nparallel"] = 8742,  
5041 ["nshortparallel"] = 8742,  
5042 ["nspar"] = 8742,  
5043 ["and"] = 8743,  
5044 ["wedge"] = 8743,  
5045 ["or"] = 8744,  
5046 ["vee"] = 8744,  
5047 ["cap"] = 8745,  
5048 ["caps"] = {8745, 65024},  
5049 ["cup"] = 8746,  
5050 ["cups"] = {8746, 65024},  
5051 ["Integral"] = 8747,  
5052 ["int"] = 8747,  
5053 ["Int"] = 8748,  
5054 ["iiint"] = 8749,  
5055 ["tint"] = 8749,  
5056 ["ContourIntegral"] = 8750,  
5057 ["conint"] = 8750,  
5058 ["oint"] = 8750,  
5059 ["Conint"] = 8751,  
5060 ["DoubleContourIntegral"] = 8751,  
5061 ["Cconint"] = 8752,  
5062 ["cwint"] = 8753,  
5063 ["ClockwiseContourIntegral"] = 8754,  
5064 ["cwconint"] = 8754,  
5065 ["CounterClockwiseContourIntegral"] = 8755,  
5066 ["awconint"] = 8755,  
5067 ["Therefore"] = 8756,  
5068 ["there4"] = 8756,  
5069 ["therefore"] = 8756,  
5070 ["Because"] = 8757,  
5071 ["because"] = 8757,  
5072 ["because"] = 8757,  
5073 ["ratio"] = 8758,  
5074 ["Colon"] = 8759,  
5075 ["Proportion"] = 8759,  
5076 ["dotminus"] = 8760,  
5077 ["minusd"] = 8760,  
5078 ["mDDot"] = 8762,  
5079 ["homtht"] = 8763,  
5080 ["Tilde"] = 8764,  
5081 ["nvsim"] = {8764, 8402},  
5082 ["sim"] = 8764,  
5083 ["thicksim"] = 8764,  
5084 ["thksim"] = 8764,  
5085 ["backsim"] = 8765,



5086 ["bsim"] = 8765,  
5087 ["race"] = {8765, 817},  
5088 ["ac"] = 8766,  
5089 ["acE"] = {8766, 819},  
5090 ["mstpos"] = 8766,  
5091 ["acd"] = 8767,  
5092 ["VerticalTilde"] = 8768,  
5093 ["wr"] = 8768,  
5094 ["wreath"] = 8768,  
5095 ["NotTilde"] = 8769,  
5096 ["nsim"] = 8769,  
5097 ["EqualTilde"] = 8770,  
5098 ["NotEqualTilde"] = {8770, 824},  
5099 ["eqsim"] = 8770,  
5100 ["esim"] = 8770,  
5101 ["nesim"] = {8770, 824},  
5102 ["TildeEqual"] = 8771,  
5103 ["sime"] = 8771,  
5104 ["simeq"] = 8771,  
5105 ["NotTildeEqual"] = 8772,  
5106 ["nsime"] = 8772,  
5107 ["nsimeq"] = 8772,  
5108 ["TildeFullEqual"] = 8773,  
5109 ["cong"] = 8773,  
5110 ["simne"] = 8774,  
5111 ["NotTildeFullEqual"] = 8775,  
5112 ["ncong"] = 8775,  
5113 ["TildeTilde"] = 8776,  
5114 ["ap"] = 8776,  
5115 ["approx"] = 8776,  
5116 ["asymp"] = 8776,  
5117 ["thickapprox"] = 8776,  
5118 ["thkap"] = 8776,  
5119 ["NotTildeTilde"] = 8777,  
5120 ["nap"] = 8777,  
5121 ["napprox"] = 8777,  
5122 ["ape"] = 8778,  
5123 ["approxpeq"] = 8778,  
5124 ["apid"] = 8779,  
5125 ["napid"] = {8779, 824},  
5126 ["backcong"] = 8780,  
5127 ["bcong"] = 8780,  
5128 ["CupCap"] = 8781,  
5129 ["asympeq"] = 8781,  
5130 ["nvap"] = {8781, 8402},  
5131 ["Bumpeq"] = 8782,  
5132 ["HumpDownHump"] = 8782,

5133 ["NotHumpDownHump"] = {8782, 824},  
5134 ["bump"] = 8782,  
5135 ["nbump"] = {8782, 824},  
5136 ["HumpEqual"] = 8783,  
5137 ["NotHumpEqual"] = {8783, 824},  
5138 ["bumpe"] = 8783,  
5139 ["bumpeq"] = 8783,  
5140 ["nbumpe"] = {8783, 824},  
5141 ["DotEqual"] = 8784,  
5142 ["doteq"] = 8784,  
5143 ["esdot"] = 8784,  
5144 ["nedot"] = {8784, 824},  
5145 ["doteqdot"] = 8785,  
5146 ["eDot"] = 8785,  
5147 ["efDot"] = 8786,  
5148 ["fallingdotseq"] = 8786,  
5149 ["erDot"] = 8787,  
5150 ["risingdotseq"] = 8787,  
5151 ["Assign"] = 8788,  
5152 ["colone"] = 8788,  
5153 ["coloneq"] = 8788,  
5154 ["ecolon"] = 8789,  
5155 ["eqcolon"] = 8789,  
5156 ["ecir"] = 8790,  
5157 ["eqcirc"] = 8790,  
5158 ["circeq"] = 8791,  
5159 ["cire"] = 8791,  
5160 ["wedgeq"] = 8793,  
5161 ["veeeq"] = 8794,  
5162 ["triangleq"] = 8796,  
5163 ["trie"] = 8796,  
5164 ["equest"] = 8799,  
5165 ["questeq"] = 8799,  
5166 ["NotEqual"] = 8800,  
5167 ["ne"] = 8800,  
5168 ["Congruent"] = 8801,  
5169 ["bnequiv"] = {8801, 8421},  
5170 ["equiv"] = 8801,  
5171 ["NotCongruent"] = 8802,  
5172 ["nequiv"] = 8802,  
5173 ["le"] = 8804,  
5174 ["leq"] = 8804,  
5175 ["nvle"] = {8804, 8402},  
5176 ["GreaterEqual"] = 8805,  
5177 ["ge"] = 8805,  
5178 ["geq"] = 8805,  
5179 ["nvge"] = {8805, 8402},

5180 ["LessFullEqual"] = 8806,  
5181 ["lE"] = 8806,  
5182 ["leqq"] = 8806,  
5183 ["nlE"] = {8806, 824},  
5184 ["nleqq"] = {8806, 824},  
5185 ["GreaterFullEqual"] = 8807,  
5186 ["NotGreaterFullEqual"] = {8807, 824},  
5187 ["gE"] = 8807,  
5188 ["geqq"] = 8807,  
5189 ["ngE"] = {8807, 824},  
5190 ["ngeqq"] = {8807, 824},  
5191 ["lnE"] = 8808,  
5192 ["lneqq"] = 8808,  
5193 ["lvertneqq"] = {8808, 65024},  
5194 ["lvnE"] = {8808, 65024},  
5195 ["gnE"] = 8809,  
5196 ["gneqq"] = 8809,  
5197 ["gvertneqq"] = {8809, 65024},  
5198 ["gvnE"] = {8809, 65024},  
5199 ["Lt"] = 8810,  
5200 ["NestedLessLess"] = 8810,  
5201 ["NotLessLess"] = {8810, 824},  
5202 ["ll"] = 8810,  
5203 ["nLt"] = {8810, 8402},  
5204 ["nLtv"] = {8810, 824},  
5205 ["Gt"] = 8811,  
5206 ["NestedGreaterGreater"] = 8811,  
5207 ["NotGreaterGreater"] = {8811, 824},  
5208 ["gg"] = 8811,  
5209 ["nGt"] = {8811, 8402},  
5210 ["nGtv"] = {8811, 824},  
5211 ["between"] = 8812,  
5212 ["twixt"] = 8812,  
5213 ["NotCupCap"] = 8813,  
5214 ["NotLess"] = 8814,  
5215 ["nless"] = 8814,  
5216 ["nlt"] = 8814,  
5217 ["NotGreater"] = 8815,  
5218 ["ngt"] = 8815,  
5219 ["ngtr"] = 8815,  
5220 ["NotLessEqual"] = 8816,  
5221 ["nle"] = 8816,  
5222 ["nleq"] = 8816,  
5223 ["NotGreaterEqual"] = 8817,  
5224 ["nge"] = 8817,  
5225 ["ngeq"] = 8817,  
5226 ["LessTilde"] = 8818,

5227 ["lesssim"] = 8818,  
5228 ["lsim"] = 8818,  
5229 ["GreaterTilde"] = 8819,  
5230 ["gsim"] = 8819,  
5231 ["gtrsim"] = 8819,  
5232 ["NotLessTilde"] = 8820,  
5233 ["nlsim"] = 8820,  
5234 ["NotGreaterTilde"] = 8821,  
5235 ["ngsim"] = 8821,  
5236 ["LessGreater"] = 8822,  
5237 ["lessgtr"] = 8822,  
5238 ["lg"] = 8822,  
5239 ["GreaterLess"] = 8823,  
5240 ["gl"] = 8823,  
5241 ["gtrless"] = 8823,  
5242 ["NotLessGreater"] = 8824,  
5243 ["ntlg"] = 8824,  
5244 ["NotGreaterLess"] = 8825,  
5245 ["ntgl"] = 8825,  
5246 ["Precedes"] = 8826,  
5247 ["pr"] = 8826,  
5248 ["prec"] = 8826,  
5249 ["Succeeds"] = 8827,  
5250 ["sc"] = 8827,  
5251 ["succ"] = 8827,  
5252 ["PrecedesSlantEqual"] = 8828,  
5253 ["prcue"] = 8828,  
5254 ["preccurlyeq"] = 8828,  
5255 ["SucceedsSlantEqual"] = 8829,  
5256 ["sccue"] = 8829,  
5257 ["succcurlyeq"] = 8829,  
5258 ["PrecedesTilde"] = 8830,  
5259 ["precsim"] = 8830,  
5260 ["prsim"] = 8830,  
5261 ["NotSucceedsTilde"] = {8831, 824},  
5262 ["SucceedsTilde"] = 8831,  
5263 ["scsim"] = 8831,  
5264 ["succsim"] = 8831,  
5265 ["NotPrecedes"] = 8832,  
5266 ["npr"] = 8832,  
5267 ["nprec"] = 8832,  
5268 ["NotSucceeds"] = 8833,  
5269 ["nsc"] = 8833,  
5270 ["nsucc"] = 8833,  
5271 ["NotSubset"] = {8834, 8402},  
5272 ["nsubset"] = {8834, 8402},  
5273 ["sub"] = 8834,

5274 ["subset"] = 8834,  
5275 ["vnsup"] = {8834, 8402},  
5276 ["NotSuperset"] = {8835, 8402},  
5277 ["Superset"] = 8835,  
5278 ["nsupset"] = {8835, 8402},  
5279 ["sup"] = 8835,  
5280 ["supset"] = 8835,  
5281 ["vnsup"] = {8835, 8402},  
5282 ["nsub"] = 8836,  
5283 ["nsup"] = 8837,  
5284 ["SubsetEqual"] = 8838,  
5285 ["sube"] = 8838,  
5286 ["subseteq"] = 8838,  
5287 ["SupersetEqual"] = 8839,  
5288 ["supe"] = 8839,  
5289 ["supseteq"] = 8839,  
5290 ["NotSubsetEqual"] = 8840,  
5291 ["nsube"] = 8840,  
5292 ["nsubseteq"] = 8840,  
5293 ["NotSupersetEqual"] = 8841,  
5294 ["nsupe"] = 8841,  
5295 ["nsupseteq"] = 8841,  
5296 ["subne"] = 8842,  
5297 ["subsetneq"] = 8842,  
5298 ["varsubsetneq"] = {8842, 65024},  
5299 ["vsubne"] = {8842, 65024},  
5300 ["supne"] = 8843,  
5301 ["supsetneq"] = 8843,  
5302 ["varsupsetneq"] = {8843, 65024},  
5303 ["vsupne"] = {8843, 65024},  
5304 ["cupdot"] = 8845,  
5305 ["UnionPlus"] = 8846,  
5306 ["uplus"] = 8846,  
5307 ["NotSquareSubset"] = {8847, 824},  
5308 ["SquareSubset"] = 8847,  
5309 ["sqsub"] = 8847,  
5310 ["sqsubset"] = 8847,  
5311 ["NotSquareSuperset"] = {8848, 824},  
5312 ["SquareSuperset"] = 8848,  
5313 ["sqsup"] = 8848,  
5314 ["sqsupset"] = 8848,  
5315 ["SquareSubsetEqual"] = 8849,  
5316 ["sqsube"] = 8849,  
5317 ["sqsubseteq"] = 8849,  
5318 ["SquareSupersetEqual"] = 8850,  
5319 ["sqsupe"] = 8850,  
5320 ["sqsupseteq"] = 8850,

5321 ["SquareIntersection"] = 8851,  
5322 ["sqcap"] = 8851,  
5323 ["sqcaps"] = {8851, 65024},  
5324 ["SquareUnion"] = 8852,  
5325 ["sqcup"] = 8852,  
5326 ["sqcups"] = {8852, 65024},  
5327 ["CirclePlus"] = 8853,  
5328 ["oplus"] = 8853,  
5329 ["CircleMinus"] = 8854,  
5330 ["ominus"] = 8854,  
5331 ["CircleTimes"] = 8855,  
5332 ["otimes"] = 8855,  
5333 ["osol"] = 8856,  
5334 ["CircleDot"] = 8857,  
5335 ["odot"] = 8857,  
5336 ["circledcirc"] = 8858,  
5337 ["ocir"] = 8858,  
5338 ["circledast"] = 8859,  
5339 ["oast"] = 8859,  
5340 ["circleddash"] = 8861,  
5341 ["odash"] = 8861,  
5342 ["boxplus"] = 8862,  
5343 ["plusb"] = 8862,  
5344 ["boxminus"] = 8863,  
5345 ["minusb"] = 8863,  
5346 ["boxtimes"] = 8864,  
5347 ["timesb"] = 8864,  
5348 ["dotsquare"] = 8865,  
5349 ["sdotb"] = 8865,  
5350 ["RightTee"] = 8866,  
5351 ["vdash"] = 8866,  
5352 ["LeftTee"] = 8867,  
5353 ["dashv"] = 8867,  
5354 ["DownTee"] = 8868,  
5355 ["top"] = 8868,  
5356 ["UpTee"] = 8869,  
5357 ["bot"] = 8869,  
5358 ["bottom"] = 8869,  
5359 ["perp"] = 8869,  
5360 ["models"] = 8871,  
5361 ["DoubleRightTee"] = 8872,  
5362 ["vDash"] = 8872,  
5363 ["Vdash"] = 8873,  
5364 ["Vvdash"] = 8874,  
5365 ["VDash"] = 8875,  
5366 ["nvdash"] = 8876,  
5367 ["nvDash"] = 8877,

5368 ["nVdash"] = 8878,  
5369 ["nVDash"] = 8879,  
5370 ["prurel"] = 8880,  
5371 ["LeftTriangle"] = 8882,  
5372 ["vartriangleleft"] = 8882,  
5373 ["vltri"] = 8882,  
5374 ["RightTriangle"] = 8883,  
5375 ["vartriangleright"] = 8883,  
5376 ["vrtri"] = 8883,  
5377 ["LeftTriangleEqual"] = 8884,  
5378 ["ltrier"] = 8884,  
5379 ["nvltrier"] = {8884, 8402},  
5380 ["trianglelefteq"] = 8884,  
5381 ["RightTriangleEqual"] = 8885,  
5382 ["nvrtrier"] = {8885, 8402},  
5383 ["rtrier"] = 8885,  
5384 ["trianglerighteq"] = 8885,  
5385 ["origof"] = 8886,  
5386 ["imof"] = 8887,  
5387 ["multimap"] = 8888,  
5388 ["mumap"] = 8888,  
5389 ["hercon"] = 8889,  
5390 ["intcal"] = 8890,  
5391 ["intercal"] = 8890,  
5392 ["veebar"] = 8891,  
5393 ["barvee"] = 8893,  
5394 ["angrtvb"] = 8894,  
5395 ["ltri"] = 8895,  
5396 ["Wedge"] = 8896,  
5397 ["bigwedge"] = 8896,  
5398 ["xwedge"] = 8896,  
5399 ["Vee"] = 8897,  
5400 ["bigvee"] = 8897,  
5401 ["xvee"] = 8897,  
5402 ["Intersection"] = 8898,  
5403 ["bigcap"] = 8898,  
5404 ["xcap"] = 8898,  
5405 ["Union"] = 8899,  
5406 ["bigcup"] = 8899,  
5407 ["xcup"] = 8899,  
5408 ["Diamond"] = 8900,  
5409 ["diam"] = 8900,  
5410 ["diamond"] = 8900,  
5411 ["sdot"] = 8901,  
5412 ["Star"] = 8902,  
5413 ["sstarf"] = 8902,  
5414 ["divideontimes"] = 8903,

5415 ["divonx"] = 8903,  
5416 ["bowtie"] = 8904,  
5417 ["ltimes"] = 8905,  
5418 ["rtimes"] = 8906,  
5419 ["leftthreetimes"] = 8907,  
5420 ["lthree"] = 8907,  
5421 ["rightthreetimes"] = 8908,  
5422 ["rthree"] = 8908,  
5423 ["backsimeq"] = 8909,  
5424 ["bsime"] = 8909,  
5425 ["curlyvee"] = 8910,  
5426 ["cuvee"] = 8910,  
5427 ["curlywedge"] = 8911,  
5428 ["cuwed"] = 8911,  
5429 ["Sub"] = 8912,  
5430 ["Subset"] = 8912,  
5431 ["Sup"] = 8913,  
5432 ["Supset"] = 8913,  
5433 ["Cap"] = 8914,  
5434 ["Cup"] = 8915,  
5435 ["fork"] = 8916,  
5436 ["pitchfork"] = 8916,  
5437 ["epar"] = 8917,  
5438 ["lessdot"] = 8918,  
5439 ["ltdot"] = 8918,  
5440 ["gtdot"] = 8919,  
5441 ["gtrdot"] = 8919,  
5442 ["L1"] = 8920,  
5443 ["nL1"] = {8920, 824},  
5444 ["Gg"] = 8921,  
5445 ["ggg"] = 8921,  
5446 ["nGg"] = {8921, 824},  
5447 ["LessEqualGreater"] = 8922,  
5448 ["leg"] = 8922,  
5449 ["lesg"] = {8922, 65024},  
5450 ["lesseqgtr"] = 8922,  
5451 ["GreaterEqualLess"] = 8923,  
5452 ["gel"] = 8923,  
5453 ["gesl"] = {8923, 65024},  
5454 ["gtreqless"] = 8923,  
5455 ["cuepr"] = 8926,  
5456 ["curlyeqprec"] = 8926,  
5457 ["cuesc"] = 8927,  
5458 ["curlyeqsucc"] = 8927,  
5459 ["NotPrecedesSlantEqual"] = 8928,  
5460 ["nprcue"] = 8928,  
5461 ["NotSucceedsSlantEqual"] = 8929,



5462 ["nsccue"] = 8929,  
5463 ["NotSquareSubsetEqual"] = 8930,  
5464 ["nsqsube"] = 8930,  
5465 ["NotSquareSupersetEqual"] = 8931,  
5466 ["nsqsupe"] = 8931,  
5467 ["lnsim"] = 8934,  
5468 ["gnsim"] = 8935,  
5469 ["precnsim"] = 8936,  
5470 ["prnsim"] = 8936,  
5471 ["scnsim"] = 8937,  
5472 ["succnsim"] = 8937,  
5473 ["NotLeftTriangle"] = 8938,  
5474 ["nltri"] = 8938,  
5475 ["ntriangleleft"] = 8938,  
5476 ["NotRightTriangle"] = 8939,  
5477 ["nrtri"] = 8939,  
5478 ["ntriangleright"] = 8939,  
5479 ["NotLeftTriangleEqual"] = 8940,  
5480 ["nltrie"] = 8940,  
5481 ["ntrianglelefteq"] = 8940,  
5482 ["NotRightTriangleEqual"] = 8941,  
5483 ["nrtrie"] = 8941,  
5484 ["ntrianglerighteq"] = 8941,  
5485 ["vellip"] = 8942,  
5486 ["ctdot"] = 8943,  
5487 ["utdot"] = 8944,  
5488 ["dtdot"] = 8945,  
5489 ["disin"] = 8946,  
5490 ["isinsv"] = 8947,  
5491 ["isins"] = 8948,  
5492 ["isindot"] = 8949,  
5493 ["notinvc"] = {8949, 824},  
5494 ["notinvc"] = 8950,  
5495 ["notinvb"] = 8951,  
5496 ["isinE"] = 8953,  
5497 ["notinE"] = {8953, 824},  
5498 ["nisd"] = 8954,  
5499 ["xnis"] = 8955,  
5500 ["nis"] = 8956,  
5501 ["notnivc"] = 8957,  
5502 ["notnivb"] = 8958,  
5503 ["barwed"] = 8965,  
5504 ["barwedge"] = 8965,  
5505 ["Barwed"] = 8966,  
5506 ["doublebarwedge"] = 8966,  
5507 ["LeftCeiling"] = 8968,  
5508 ["lceil"] = 8968,

5509 ["RightCeiling"] = 8969,  
5510 ["rceil"] = 8969,  
5511 ["LeftFloor"] = 8970,  
5512 ["lfloor"] = 8970,  
5513 ["RightFloor"] = 8971,  
5514 ["rfloor"] = 8971,  
5515 ["drcrop"] = 8972,  
5516 ["dlcrop"] = 8973,  
5517 ["urcrop"] = 8974,  
5518 ["ulcrop"] = 8975,  
5519 ["bnot"] = 8976,  
5520 ["proflin"] = 8978,  
5521 ["profsurf"] = 8979,  
5522 ["telrec"] = 8981,  
5523 ["target"] = 8982,  
5524 ["ulcorn"] = 8988,  
5525 ["ulcorner"] = 8988,  
5526 ["urcorn"] = 8989,  
5527 ["urcorner"] = 8989,  
5528 ["dlcorn"] = 8990,  
5529 ["llcorner"] = 8990,  
5530 ["drcorn"] = 8991,  
5531 ["lrcorn"] = 8991,  
5532 ["frown"] = 8994,  
5533 ["sfrown"] = 8994,  
5534 ["smile"] = 8995,  
5535 ["ssmile"] = 8995,  
5536 ["cylcty"] = 9005,  
5537 ["profalar"] = 9006,  
5538 ["topbot"] = 9014,  
5539 ["ovbar"] = 9021,  
5540 ["solbar"] = 9023,  
5541 ["angzarr"] = 9084,  
5542 ["lmoust"] = 9136,  
5543 ["lmoustache"] = 9136,  
5544 ["rmoust"] = 9137,  
5545 ["rmoustache"] = 9137,  
5546 ["OverBracket"] = 9140,  
5547 ["tbrk"] = 9140,  
5548 ["UnderBracket"] = 9141,  
5549 ["bbrk"] = 9141,  
5550 ["bbrktbrk"] = 9142,  
5551 ["OverParenthesis"] = 9180,  
5552 ["UnderParenthesis"] = 9181,  
5553 ["OverBrace"] = 9182,  
5554 ["UnderBrace"] = 9183,  
5555 ["trpezium"] = 9186,

5556 ["elinters"] = 9191,  
5557 ["blank"] = 9251,  
5558 ["circledS"] = 9416,  
5559 ["oS"] = 9416,  
5560 ["HorizontalLine"] = 9472,  
5561 ["boxh"] = 9472,  
5562 ["boxv"] = 9474,  
5563 ["boxdr"] = 9484,  
5564 ["boxdl"] = 9488,  
5565 ["boxur"] = 9492,  
5566 ["boxul"] = 9496,  
5567 ["boxvr"] = 9500,  
5568 ["boxvl"] = 9508,  
5569 ["boxhd"] = 9516,  
5570 ["boxhu"] = 9524,  
5571 ["boxvh"] = 9532,  
5572 ["boxH"] = 9552,  
5573 ["boxV"] = 9553,  
5574 ["boxdR"] = 9554,  
5575 ["boxDr"] = 9555,  
5576 ["boxDR"] = 9556,  
5577 ["boxdL"] = 9557,  
5578 ["boxDL"] = 9558,  
5579 ["boxDL"] = 9559,  
5580 ["boxuR"] = 9560,  
5581 ["boxUr"] = 9561,  
5582 ["boxUR"] = 9562,  
5583 ["boxuL"] = 9563,  
5584 ["boxUL"] = 9564,  
5585 ["boxUL"] = 9565,  
5586 ["boxvR"] = 9566,  
5587 ["boxVr"] = 9567,  
5588 ["boxVR"] = 9568,  
5589 ["boxvL"] = 9569,  
5590 ["boxVL"] = 9570,  
5591 ["boxVL"] = 9571,  
5592 ["boxHd"] = 9572,  
5593 ["boxhD"] = 9573,  
5594 ["boxHD"] = 9574,  
5595 ["boxHu"] = 9575,  
5596 ["boxhU"] = 9576,  
5597 ["boxHU"] = 9577,  
5598 ["boxvH"] = 9578,  
5599 ["boxVh"] = 9579,  
5600 ["boxVH"] = 9580,  
5601 ["uhblk"] = 9600,  
5602 ["lhblk"] = 9604,

5603 ["block"] = 9608,  
5604 ["blk14"] = 9617,  
5605 ["blk12"] = 9618,  
5606 ["blk34"] = 9619,  
5607 ["Square"] = 9633,  
5608 ["squ"] = 9633,  
5609 ["square"] = 9633,  
5610 ["FilledVerySmallSquare"] = 9642,  
5611 ["blacksquare"] = 9642,  
5612 ["squarf"] = 9642,  
5613 ["squf"] = 9642,  
5614 ["EmptyVerySmallSquare"] = 9643,  
5615 ["rect"] = 9645,  
5616 ["marker"] = 9646,  
5617 ["fltns"] = 9649,  
5618 ["bigtriangleup"] = 9651,  
5619 ["xutri"] = 9651,  
5620 ["blacktriangle"] = 9652,  
5621 ["utrif"] = 9652,  
5622 ["triangle"] = 9653,  
5623 ["utri"] = 9653,  
5624 ["blacktriangleright"] = 9656,  
5625 ["rtrif"] = 9656,  
5626 ["rtri"] = 9657,  
5627 ["triangleright"] = 9657,  
5628 ["bigtriangledown"] = 9661,  
5629 ["xdtri"] = 9661,  
5630 ["blacktriangledown"] = 9662,  
5631 ["dtrif"] = 9662,  
5632 ["dtri"] = 9663,  
5633 ["triangledown"] = 9663,  
5634 ["blacktriangleleft"] = 9666,  
5635 ["ltrif"] = 9666,  
5636 ["ltri"] = 9667,  
5637 ["triangleleft"] = 9667,  
5638 ["loz"] = 9674,  
5639 ["lozenge"] = 9674,  
5640 ["cir"] = 9675,  
5641 ["tridot"] = 9708,  
5642 ["bigcirc"] = 9711,  
5643 ["xcirc"] = 9711,  
5644 ["ultri"] = 9720,  
5645 ["urtri"] = 9721,  
5646 ["lltri"] = 9722,  
5647 ["EmptySmallSquare"] = 9723,  
5648 ["FilledSmallSquare"] = 9724,  
5649 ["bigstar"] = 9733,

5650 ["starf"] = 9733,  
5651 ["star"] = 9734,  
5652 ["phone"] = 9742,  
5653 ["female"] = 9792,  
5654 ["male"] = 9794,  
5655 ["spades"] = 9824,  
5656 ["spadesuit"] = 9824,  
5657 ["clubs"] = 9827,  
5658 ["clubsuit"] = 9827,  
5659 ["hearts"] = 9829,  
5660 ["heartsuit"] = 9829,  
5661 ["diamondsuit"] = 9830,  
5662 ["diams"] = 9830,  
5663 ["sung"] = 9834,  
5664 ["flat"] = 9837,  
5665 ["natur"] = 9838,  
5666 ["natural"] = 9838,  
5667 ["sharp"] = 9839,  
5668 ["check"] = 10003,  
5669 ["checkmark"] = 10003,  
5670 ["cross"] = 10007,  
5671 ["malt"] = 10016,  
5672 ["maltese"] = 10016,  
5673 ["sext"] = 10038,  
5674 ["VerticalSeparator"] = 10072,  
5675 ["lbrk"] = 10098,  
5676 ["rbrk"] = 10099,  
5677 ["bsolhsub"] = 10184,  
5678 ["suphsol"] = 10185,  
5679 ["LeftDoubleBracket"] = 10214,  
5680 ["lobrk"] = 10214,  
5681 ["RightDoubleBracket"] = 10215,  
5682 ["robrk"] = 10215,  
5683 ["LeftAngleBracket"] = 10216,  
5684 ["lang"] = 10216,  
5685 ["langle"] = 10216,  
5686 ["RightAngleBracket"] = 10217,  
5687 ["rang"] = 10217,  
5688 ["rangle"] = 10217,  
5689 ["Lang"] = 10218,  
5690 ["Rang"] = 10219,  
5691 ["loang"] = 10220,  
5692 ["roang"] = 10221,  
5693 ["LongLeftArrow"] = 10229,  
5694 ["longleftarrow"] = 10229,  
5695 ["xlarr"] = 10229,  
5696 ["LongRightArrow"] = 10230,

5697 ["longrightarrow"] = 10230,  
5698 ["xrarr"] = 10230,  
5699 ["LongLeftRightArrow"] = 10231,  
5700 ["longleftrightarrow"] = 10231,  
5701 ["xharr"] = 10231,  
5702 ["DoubleLongLeftArrow"] = 10232,  
5703 ["Longleftarrow"] = 10232,  
5704 ["xlArr"] = 10232,  
5705 ["DoubleLongRightArrow"] = 10233,  
5706 ["Longrightarrow"] = 10233,  
5707 ["xrArr"] = 10233,  
5708 ["DoubleLongLeftRightArrow"] = 10234,  
5709 ["Longlefttrightharrow"] = 10234,  
5710 ["xhArr"] = 10234,  
5711 ["longmapsto"] = 10236,  
5712 ["xmap"] = 10236,  
5713 ["dzigrarr"] = 10239,  
5714 ["nvlArr"] = 10498,  
5715 ["nvrArr"] = 10499,  
5716 ["nvHarr"] = 10500,  
5717 ["Map"] = 10501,  
5718 ["lbarr"] = 10508,  
5719 ["bkarow"] = 10509,  
5720 ["rbarr"] = 10509,  
5721 ["lBarr"] = 10510,  
5722 ["dbkarow"] = 10511,  
5723 ["rBarr"] = 10511,  
5724 ["RBarr"] = 10512,  
5725 ["drbkarow"] = 10512,  
5726 ["DDottrahd"] = 10513,  
5727 ["UpArrowBar"] = 10514,  
5728 ["DownArrowBar"] = 10515,  
5729 ["Rarrtl"] = 10518,  
5730 ["latail"] = 10521,  
5731 ["ratail"] = 10522,  
5732 ["lAtail"] = 10523,  
5733 ["rAtail"] = 10524,  
5734 ["larrfs"] = 10525,  
5735 ["rarrfs"] = 10526,  
5736 ["larrbfs"] = 10527,  
5737 ["rarrbfs"] = 10528,  
5738 ["nwarhk"] = 10531,  
5739 ["nearhk"] = 10532,  
5740 ["hksearrow"] = 10533,  
5741 ["searhk"] = 10533,  
5742 ["hkswarrow"] = 10534,  
5743 ["swarhk"] = 10534,

5744 ["nwnear"] = 10535,  
5745 ["nesear"] = 10536,  
5746 ["toea"] = 10536,  
5747 ["seswar"] = 10537,  
5748 ["tosa"] = 10537,  
5749 ["swnwar"] = 10538,  
5750 ["nrarrc"] = {10547, 824},  
5751 ["rarrc"] = 10547,  
5752 ["cudarr"] = 10549,  
5753 ["ldca"] = 10550,  
5754 ["rdca"] = 10551,  
5755 ["cudarrl"] = 10552,  
5756 ["larrpl"] = 10553,  
5757 ["curarrm"] = 10556,  
5758 ["cularrp"] = 10557,  
5759 ["rarrpl"] = 10565,  
5760 ["harrcir"] = 10568,  
5761 ["Uarrocir"] = 10569,  
5762 ["lurdshar"] = 10570,  
5763 ["ldrushar"] = 10571,  
5764 ["LeftRightVector"] = 10574,  
5765 ["RightUpDownVector"] = 10575,  
5766 ["DownLeftRightVector"] = 10576,  
5767 ["LeftUpDownVector"] = 10577,  
5768 ["LeftVectorBar"] = 10578,  
5769 ["RightVectorBar"] = 10579,  
5770 ["RightUpVectorBar"] = 10580,  
5771 ["RightDownVectorBar"] = 10581,  
5772 ["DownLeftVectorBar"] = 10582,  
5773 ["DownRightVectorBar"] = 10583,  
5774 ["LeftUpVectorBar"] = 10584,  
5775 ["LeftDownVectorBar"] = 10585,  
5776 ["LeftTeeVector"] = 10586,  
5777 ["RightTeeVector"] = 10587,  
5778 ["RightUpTeeVector"] = 10588,  
5779 ["RightDownTeeVector"] = 10589,  
5780 ["DownLeftTeeVector"] = 10590,  
5781 ["DownRightTeeVector"] = 10591,  
5782 ["LeftUpTeeVector"] = 10592,  
5783 ["LeftDownTeeVector"] = 10593,  
5784 ["lHar"] = 10594,  
5785 ["uHar"] = 10595,  
5786 ["rHar"] = 10596,  
5787 ["dHar"] = 10597,  
5788 ["luruhar"] = 10598,  
5789 ["ldrdhar"] = 10599,  
5790 ["ruluhar"] = 10600,

5791 ["rdldhar"] = 10601,  
5792 ["lharul"] = 10602,  
5793 ["llhard"] = 10603,  
5794 ["rharul"] = 10604,  
5795 ["lrhard"] = 10605,  
5796 ["UpEquilibrium"] = 10606,  
5797 ["udhar"] = 10606,  
5798 ["ReverseUpEquilibrium"] = 10607,  
5799 ["duhar"] = 10607,  
5800 ["RoundImplies"] = 10608,  
5801 ["erarr"] = 10609,  
5802 ["simrarr"] = 10610,  
5803 ["larrsim"] = 10611,  
5804 ["rarrsim"] = 10612,  
5805 ["rarrap"] = 10613,  
5806 ["ltlarr"] = 10614,  
5807 ["gtrarr"] = 10616,  
5808 ["subrarr"] = 10617,  
5809 ["suplarr"] = 10619,  
5810 ["lfisht"] = 10620,  
5811 ["rfisht"] = 10621,  
5812 ["ufisht"] = 10622,  
5813 ["dfisht"] = 10623,  
5814 ["lopar"] = 10629,  
5815 ["ropar"] = 10630,  
5816 ["lbrke"] = 10635,  
5817 ["rbrke"] = 10636,  
5818 ["lbrkslu"] = 10637,  
5819 ["rbrksld"] = 10638,  
5820 ["lbrksld"] = 10639,  
5821 ["rbrkslu"] = 10640,  
5822 ["langd"] = 10641,  
5823 ["rangd"] = 10642,  
5824 ["lparlt"] = 10643,  
5825 ["rpargt"] = 10644,  
5826 ["gtlPar"] = 10645,  
5827 ["ltrPar"] = 10646,  
5828 ["vzigzag"] = 10650,  
5829 ["vangrt"] = 10652,  
5830 ["angrtvbd"] = 10653,  
5831 ["ange"] = 10660,  
5832 ["range"] = 10661,  
5833 ["dwangle"] = 10662,  
5834 ["uwangle"] = 10663,  
5835 ["angmsdaa"] = 10664,  
5836 ["angmsdab"] = 10665,  
5837 ["angmsdac"] = 10666,



5838 ["angmsdad"] = 10667,  
5839 ["angmsdae"] = 10668,  
5840 ["angmsdaf"] = 10669,  
5841 ["angmsdag"] = 10670,  
5842 ["angmsdah"] = 10671,  
5843 ["bemptyv"] = 10672,  
5844 ["demptyv"] = 10673,  
5845 ["cemptyv"] = 10674,  
5846 ["raemtyv"] = 10675,  
5847 ["laemtyv"] = 10676,  
5848 ["ohbar"] = 10677,  
5849 ["omid"] = 10678,  
5850 ["opar"] = 10679,  
5851 ["operp"] = 10681,  
5852 ["olcross"] = 10683,  
5853 ["odsold"] = 10684,  
5854 ["olcir"] = 10686,  
5855 ["ofcir"] = 10687,  
5856 ["olt"] = 10688,  
5857 ["ogt"] = 10689,  
5858 ["cirscir"] = 10690,  
5859 ["cirE"] = 10691,  
5860 ["solb"] = 10692,  
5861 ["bsolb"] = 10693,  
5862 ["boxbox"] = 10697,  
5863 ["trisb"] = 10701,  
5864 ["rtriltri"] = 10702,  
5865 ["LeftTriangleBar"] = 10703,  
5866 ["NotLeftTriangleBar"] = {10703, 824},  
5867 ["NotRightTriangleBar"] = {10704, 824},  
5868 ["RightTriangleBar"] = 10704,  
5869 ["iinfin"] = 10716,  
5870 ["infintie"] = 10717,  
5871 ["nvinfin"] = 10718,  
5872 ["eparsl"] = 10723,  
5873 ["smeparsl"] = 10724,  
5874 ["eqvparsl"] = 10725,  
5875 ["blacklozenge"] = 10731,  
5876 ["lozf"] = 10731,  
5877 ["RuleDelayed"] = 10740,  
5878 ["dsol"] = 10742,  
5879 ["bigodot"] = 10752,  
5880 ["xodot"] = 10752,  
5881 ["bigoplus"] = 10753,  
5882 ["xoplus"] = 10753,  
5883 ["bigotimes"] = 10754,  
5884 ["xotime"] = 10754,

5885 ["biguplus"] = 10756,  
5886 ["xuplus"] = 10756,  
5887 ["bigsqcup"] = 10758,  
5888 ["xscup"] = 10758,  
5889 ["iiiint"] = 10764,  
5890 ["qint"] = 10764,  
5891 ["fpartint"] = 10765,  
5892 ["cirfnint"] = 10768,  
5893 ["awint"] = 10769,  
5894 ["rppolint"] = 10770,  
5895 ["scpolint"] = 10771,  
5896 ["npolint"] = 10772,  
5897 ["pointint"] = 10773,  
5898 ["quatint"] = 10774,  
5899 ["intlarhk"] = 10775,  
5900 ["pluscir"] = 10786,  
5901 ["plusacir"] = 10787,  
5902 ["simplus"] = 10788,  
5903 ["plusdu"] = 10789,  
5904 ["plussim"] = 10790,  
5905 ["plustwo"] = 10791,  
5906 ["mcomma"] = 10793,  
5907 ["minusdu"] = 10794,  
5908 ["loplus"] = 10797,  
5909 ["roplus"] = 10798,  
5910 ["Cross"] = 10799,  
5911 ["timesd"] = 10800,  
5912 ["timesbar"] = 10801,  
5913 ["smashp"] = 10803,  
5914 ["lotimes"] = 10804,  
5915 ["rotimes"] = 10805,  
5916 ["otimesas"] = 10806,  
5917 ["Otimes"] = 10807,  
5918 ["odiv"] = 10808,  
5919 ["triplus"] = 10809,  
5920 ["triminus"] = 10810,  
5921 ["tritime"] = 10811,  
5922 ["intprod"] = 10812,  
5923 ["iproduct"] = 10812,  
5924 ["amalg"] = 10815,  
5925 ["capdot"] = 10816,  
5926 ["ncup"] = 10818,  
5927 ["ncap"] = 10819,  
5928 ["capand"] = 10820,  
5929 ["cupor"] = 10821,  
5930 ["cupcap"] = 10822,  
5931 ["capcup"] = 10823,

5932 ["cupbrcap"] = 10824,  
5933 ["capbrcup"] = 10825,  
5934 ["cupcup"] = 10826,  
5935 ["capcap"] = 10827,  
5936 ["ccups"] = 10828,  
5937 ["ccaps"] = 10829,  
5938 ["ccupssm"] = 10832,  
5939 ["And"] = 10835,  
5940 ["Or"] = 10836,  
5941 ["andand"] = 10837,  
5942 ["oror"] = 10838,  
5943 ["orslope"] = 10839,  
5944 ["andslope"] = 10840,  
5945 ["andv"] = 10842,  
5946 ["orv"] = 10843,  
5947 ["andd"] = 10844,  
5948 ["ord"] = 10845,  
5949 ["wedbar"] = 10847,  
5950 ["sdote"] = 10854,  
5951 ["simdot"] = 10858,  
5952 ["congdote"] = 10861,  
5953 ["ncongdote"] = {10861, 824},  
5954 ["easter"] = 10862,  
5955 ["apacir"] = 10863,  
5956 ["apE"] = 10864,  
5957 ["napE"] = {10864, 824},  
5958 ["eplus"] = 10865,  
5959 ["pluse"] = 10866,  
5960 ["Esim"] = 10867,  
5961 ["Colone"] = 10868,  
5962 ["Equal"] = 10869,  
5963 ["ddotseq"] = 10871,  
5964 ["eDDot"] = 10871,  
5965 ["equivDD"] = 10872,  
5966 ["ltcir"] = 10873,  
5967 ["gtcir"] = 10874,  
5968 ["ltquest"] = 10875,  
5969 ["gtquest"] = 10876,  
5970 ["LessSlantEqual"] = 10877,  
5971 ["NotLessSlantEqual"] = {10877, 824},  
5972 ["leqslant"] = 10877,  
5973 ["les"] = 10877,  
5974 ["nleqslant"] = {10877, 824},  
5975 ["nles"] = {10877, 824},  
5976 ["GreaterSlantEqual"] = 10878,  
5977 ["NotGreaterSlantEqual"] = {10878, 824},  
5978 ["geqslant"] = 10878,

5979 ["ges"] = 10878,  
5980 ["ngeqslant"] = {10878, 824},  
5981 ["nges"] = {10878, 824},  
5982 ["lesdot"] = 10879,  
5983 ["gesdot"] = 10880,  
5984 ["lesdoto"] = 10881,  
5985 ["gesdoto"] = 10882,  
5986 ["lesdotor"] = 10883,  
5987 ["gesdoto1"] = 10884,  
5988 ["lap"] = 10885,  
5989 ["lessapprox"] = 10885,  
5990 ["gap"] = 10886,  
5991 ["gtrapprox"] = 10886,  
5992 ["lne"] = 10887,  
5993 ["lneq"] = 10887,  
5994 ["gne"] = 10888,  
5995 ["gneq"] = 10888,  
5996 ["lnap"] = 10889,  
5997 ["lnapprox"] = 10889,  
5998 ["gnap"] = 10890,  
5999 ["gnapprox"] = 10890,  
6000 ["lEg"] = 10891,  
6001 ["lesseqqgtr"] = 10891,  
6002 ["gEl"] = 10892,  
6003 ["gtreqqless"] = 10892,  
6004 ["lsime"] = 10893,  
6005 ["gsime"] = 10894,  
6006 ["lsimg"] = 10895,  
6007 ["gsiml"] = 10896,  
6008 ["lgE"] = 10897,  
6009 ["glE"] = 10898,  
6010 ["lesges"] = 10899,  
6011 ["gesles"] = 10900,  
6012 ["els"] = 10901,  
6013 ["eqslantless"] = 10901,  
6014 ["egs"] = 10902,  
6015 ["eqslantgtr"] = 10902,  
6016 ["elsdot"] = 10903,  
6017 ["egsdot"] = 10904,  
6018 ["el"] = 10905,  
6019 ["eg"] = 10906,  
6020 ["siml"] = 10909,  
6021 ["sing"] = 10910,  
6022 ["simlE"] = 10911,  
6023 ["singE"] = 10912,  
6024 ["LessLess"] = 10913,  
6025 ["NotNestedLessLess"] = {10913, 824},

```

6026 ["GreaterGreater"] = 10914,
6027 ["NotNestedGreaterGreater"] = {10914, 824},
6028 ["glj"] = 10916,
6029 ["gla"] = 10917,
6030 ["ltcc"] = 10918,
6031 ["gtcc"] = 10919,
6032 ["lescc"] = 10920,
6033 ["gescc"] = 10921,
6034 ["smt"] = 10922,
6035 ["lat"] = 10923,
6036 ["smte"] = 10924,
6037 ["smtes"] = {10924, 65024},
6038 ["late"] = 10925,
6039 ["lates"] = {10925, 65024},
6040 ["bumpE"] = 10926,
6041 ["NotPrecedesEqual"] = {10927, 824},
6042 ["PrecedesEqual"] = 10927,
6043 ["npre"] = {10927, 824},
6044 ["npreceq"] = {10927, 824},
6045 ["pre"] = 10927,
6046 ["preceq"] = 10927,
6047 ["NotSucceedsEqual"] = {10928, 824},
6048 ["SucceedsEqual"] = 10928,
6049 ["nsce"] = {10928, 824},
6050 ["nsucceq"] = {10928, 824},
6051 ["sce"] = 10928,
6052 ["succeq"] = 10928,
6053 ["prE"] = 10931,
6054 ["scE"] = 10932,
6055 ["precneqq"] = 10933,
6056 ["prnE"] = 10933,
6057 ["scnE"] = 10934,
6058 ["succneqq"] = 10934,
6059 ["prap"] = 10935,
6060 ["precapprox"] = 10935,
6061 ["scap"] = 10936,
6062 ["succapprox"] = 10936,
6063 ["precnapprox"] = 10937,
6064 ["prnap"] = 10937,
6065 ["scnap"] = 10938,
6066 ["succnapprox"] = 10938,
6067 ["Pr"] = 10939,
6068 ["Sc"] = 10940,
6069 ["subdot"] = 10941,
6070 ["supdot"] = 10942,
6071 ["subplus"] = 10943,
6072 ["supplus"] = 10944,

```

6073 ["submult"] = 10945,  
6074 ["supmult"] = 10946,  
6075 ["subedot"] = 10947,  
6076 ["supedot"] = 10948,  
6077 ["nsubE"] = {10949, 824},  
6078 ["nsubseteqq"] = {10949, 824},  
6079 ["subE"] = 10949,  
6080 ["subseteqq"] = 10949,  
6081 ["nsupE"] = {10950, 824},  
6082 ["nsupseteqq"] = {10950, 824},  
6083 ["supE"] = 10950,  
6084 ["supseteqq"] = 10950,  
6085 ["subsim"] = 10951,  
6086 ["supsim"] = 10952,  
6087 ["subnE"] = 10955,  
6088 ["subsetneqq"] = 10955,  
6089 ["varsubsetneqq"] = {10955, 65024},  
6090 ["vsubnE"] = {10955, 65024},  
6091 ["supnE"] = 10956,  
6092 ["supsetneqq"] = 10956,  
6093 ["varsupsetneqq"] = {10956, 65024},  
6094 ["vsupnE"] = {10956, 65024},  
6095 ["csub"] = 10959,  
6096 ["csup"] = 10960,  
6097 ["csube"] = 10961,  
6098 ["csupe"] = 10962,  
6099 ["subsup"] = 10963,  
6100 ["supsub"] = 10964,  
6101 ["subsub"] = 10965,  
6102 ["supsup"] = 10966,  
6103 ["suphsub"] = 10967,  
6104 ["supdsub"] = 10968,  
6105 ["forkv"] = 10969,  
6106 ["topfork"] = 10970,  
6107 ["mlcp"] = 10971,  
6108 ["Dashv"] = 10980,  
6109 ["DoubleLeftTee"] = 10980,  
6110 ["Vdashl"] = 10982,  
6111 ["Barv"] = 10983,  
6112 ["vBar"] = 10984,  
6113 ["vBarv"] = 10985,  
6114 ["Vbar"] = 10987,  
6115 ["Not"] = 10988,  
6116 ["bNot"] = 10989,  
6117 ["rnmid"] = 10990,  
6118 ["cirmid"] = 10991,  
6119 ["midcir"] = 10992,

6120 ["topcir"] = 10993,  
6121 ["nhpar"] = 10994,  
6122 ["parsim"] = 10995,  
6123 ["nparsl"] = {11005, 8421},  
6124 ["parsl"] = 11005,  
6125 ["fflig"] = 64256,  
6126 ["filig"] = 64257,  
6127 ["fllig"] = 64258,  
6128 ["ffilig"] = 64259,  
6129 ["ffllig"] = 64260,  
6130 ["Ascr"] = 119964,  
6131 ["Cscr"] = 119966,  
6132 ["Dscr"] = 119967,  
6133 ["Gscr"] = 119970,  
6134 ["Jscr"] = 119973,  
6135 ["Kscr"] = 119974,  
6136 ["Nscr"] = 119977,  
6137 ["Oscr"] = 119978,  
6138 ["Pscr"] = 119979,  
6139 ["Qscr"] = 119980,  
6140 ["Sscr"] = 119982,  
6141 ["Tscr"] = 119983,  
6142 ["Uscr"] = 119984,  
6143 ["Vscr"] = 119985,  
6144 ["Wscr"] = 119986,  
6145 ["Xscr"] = 119987,  
6146 ["Yscr"] = 119988,  
6147 ["Zscr"] = 119989,  
6148 ["ascr"] = 119990,  
6149 ["bscr"] = 119991,  
6150 ["cscr"] = 119992,  
6151 ["dscr"] = 119993,  
6152 ["fscr"] = 119995,  
6153 ["hscr"] = 119997,  
6154 ["iscr"] = 119998,  
6155 ["jscr"] = 119999,  
6156 ["kscr"] = 120000,  
6157 ["lscr"] = 120001,  
6158 ["mscr"] = 120002,  
6159 ["nscr"] = 120003,  
6160 ["pscr"] = 120005,  
6161 ["qscr"] = 120006,  
6162 ["rscr"] = 120007,  
6163 ["sscr"] = 120008,  
6164 ["tscr"] = 120009,  
6165 ["uscr"] = 120010,  
6166 ["vscr"] = 120011,

6167 ["wscr"] = 120012,  
6168 ["xscr"] = 120013,  
6169 ["yscr"] = 120014,  
6170 ["zscr"] = 120015,  
6171 ["Afr"] = 120068,  
6172 ["Bfr"] = 120069,  
6173 ["Dfr"] = 120071,  
6174 ["Efr"] = 120072,  
6175 ["Ffr"] = 120073,  
6176 ["Gfr"] = 120074,  
6177 ["Jfr"] = 120077,  
6178 ["Kfr"] = 120078,  
6179 ["Lfr"] = 120079,  
6180 ["Mfr"] = 120080,  
6181 ["Nfr"] = 120081,  
6182 ["Ofr"] = 120082,  
6183 ["Pfr"] = 120083,  
6184 ["Qfr"] = 120084,  
6185 ["Sfr"] = 120086,  
6186 ["Tfr"] = 120087,  
6187 ["Ufr"] = 120088,  
6188 ["Vfr"] = 120089,  
6189 ["Wfr"] = 120090,  
6190 ["Xfr"] = 120091,  
6191 ["Yfr"] = 120092,  
6192 ["afr"] = 120094,  
6193 ["bfr"] = 120095,  
6194 ["cfr"] = 120096,  
6195 ["dfr"] = 120097,  
6196 ["efr"] = 120098,  
6197 ["ffr"] = 120099,  
6198 ["gfr"] = 120100,  
6199 ["hfr"] = 120101,  
6200 ["ifr"] = 120102,  
6201 ["jfr"] = 120103,  
6202 ["kfr"] = 120104,  
6203 ["lfr"] = 120105,  
6204 ["mfr"] = 120106,  
6205 ["nfr"] = 120107,  
6206 ["ofr"] = 120108,  
6207 ["pfr"] = 120109,  
6208 ["qfr"] = 120110,  
6209 ["rfr"] = 120111,  
6210 ["sfr"] = 120112,  
6211 ["tfr"] = 120113,  
6212 ["ufr"] = 120114,  
6213 ["vfr"] = 120115,



6214 ["wfr"] = 120116,  
6215 ["xfr"] = 120117,  
6216 ["yfr"] = 120118,  
6217 ["zfr"] = 120119,  
6218 ["Aopf"] = 120120,  
6219 ["Bopf"] = 120121,  
6220 ["Dopf"] = 120123,  
6221 ["Eopf"] = 120124,  
6222 ["Fopf"] = 120125,  
6223 ["Gopf"] = 120126,  
6224 ["Iopf"] = 120128,  
6225 ["Jopf"] = 120129,  
6226 ["Kopf"] = 120130,  
6227 ["Lopf"] = 120131,  
6228 ["Mopf"] = 120132,  
6229 ["Oopf"] = 120134,  
6230 ["Sopf"] = 120138,  
6231 ["Topf"] = 120139,  
6232 ["Uopf"] = 120140,  
6233 ["Vopf"] = 120141,  
6234 ["Wopf"] = 120142,  
6235 ["Xopf"] = 120143,  
6236 ["Yopf"] = 120144,  
6237 ["aopf"] = 120146,  
6238 ["bopf"] = 120147,  
6239 ["copf"] = 120148,  
6240 ["dopf"] = 120149,  
6241 ["eopf"] = 120150,  
6242 ["fopf"] = 120151,  
6243 ["gopf"] = 120152,  
6244 ["hopf"] = 120153,  
6245 ["iopf"] = 120154,  
6246 ["jopf"] = 120155,  
6247 ["kopf"] = 120156,  
6248 ["lopf"] = 120157,  
6249 ["mopf"] = 120158,  
6250 ["nopf"] = 120159,  
6251 ["oopf"] = 120160,  
6252 ["popf"] = 120161,  
6253 ["qopf"] = 120162,  
6254 ["ropf"] = 120163,  
6255 ["sopf"] = 120164,  
6256 ["topf"] = 120165,  
6257 ["uopf"] = 120166,  
6258 ["vopf"] = 120167,  
6259 ["wopf"] = 120168,  
6260 ["xopf"] = 120169,

```

6261 ["yopf"] = 120170,
6262 ["zopf"] = 120171,
6263 }

```

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

```

6264 function entities.dec_entity(s)
6265 local n = tonumber(s)
6266 if n == nil then
6267 return "&#" .. s .. ";" -- fallback for unknown entities
6268 end
6269 return utf8.char(n)
6270 end

```

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

```

6271 function entities.hex_entity(s)
6272 local n = tonumber("0x"..s)
6273 if n == nil then
6274 return "&#x" .. s .. ";" -- fallback for unknown entities
6275 end
6276 return utf8.char(n)
6277 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.

```

6278 function entities.hex_entity_with_x_char(x, s)
6279 local n = tonumber("0x"..s)
6280 if n == nil then
6281 return "&#" .. x .. s .. ";" -- fallback for unknown entities
6282 end
6283 return utf8.char(n)
6284 end

```

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

```

6285 function entities.char_entity(s)
6286 local code_points = character_entities[s]
6287 if code_points == nil then
6288 return "&" .. s .. ";"
6289 end
6290 if type(code_points) ~= 'table' then
6291 code_points = {code_points}
6292 end
6293 local char_table = {}
6294 for _, code_point in ipairs(code_points) do
6295 table.insert(char_table, utf8.char(code_point))

```

```

6296 end
6297 return table.concat(char_table)
6298 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.

```

6299 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.

```

6300 local parsers
6301 function M.writer.new(options)
6302 local self = {}

```

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

```

6303 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.

```

6304 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.

```

6305 local slice_specifiers = {}
6306 for specifier in options.slice:gmatch("[~%s]+") do
6307 table.insert(slice_specifiers, specifier)
6308 end
6309
6310 if #slice_specifiers == 2 then
6311 self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
6312 local slice_begin_type = self.slice_begin:sub(1, 1)

```

```

6313 if slice_begin_type =~ "^" and slice_begin_type =~ "$" then
6314 self.slice_begin = "^" .. self.slice_begin
6315 end
6316 local slice_end_type = self.slice_end:sub(1, 1)
6317 if slice_end_type =~ "^" and slice_end_type =~ "$" then
6318 self.slice_end = "$" .. self.slice_end
6319 end
6320 elsif #slice_specifiers == 1 then
6321 self.slice_begin = "^" .. slice_specifiers[1]
6322 self.slice_end = "$" .. slice_specifiers[1]
6323 end
6324
6325 self.slice_begin_type = self.slice_begin:sub(1, 1)
6326 self.slice_begin_identifier = self.slice_begin:sub(2) or ""
6327 self.slice_end_type = self.slice_end:sub(1, 1)
6328 self.slice_end_identifier = self.slice_end:sub(2) or ""
6329
6330 if self.slice_begin == "^" and self.slice_end =~ "^" then
6331 self.is_writing = true
6332 else
6333 self.is_writing = false
6334 end

```

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

```

6335 self.space = " "

```

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

```

6336 self.nbsp = "\\markdownRendererNbsp{}"

```

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

```

6337 function self.plain(s)
6338 return s
6339 end

```

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

```

6340 function self.paragraph(s)
6341 if not self.is_writing then return "" end
6342 return s
6343 end

```

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

```

6344 self.interblocksep_text = "\\markdownRendererInterblockSeparator\n{}"
6345 function self.interblocksep()
6346 if not self.is_writing then return "" end
6347 return self.interblocksep_text
6348 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.

```
6349 self.paragraphsep_text = "\\markdownRendererParagraphSeparator\n{"
6350 function self.paragraphsep()
6351 if not self.is_writing then return "" end
6352 return self.paragraphsep_text
6353 end
```

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

```
6354 self.undosep_text = "\\markdownRendererUndoSeparator\n{"
6355 function self.undosep()
6356 if not self.is_writing then return "" end
6357 return self.undosep_text
6358 end
```

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

```
6359 self.soft_line_break = function()
6360 if self.flatten_inlines then return "\n" end
6361 return "\\markdownRendererSoftLineBreak\n{"
6362 end
```

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

```
6363 self.hard_line_break = function()
6364 if self.flatten_inlines then return "\n" end
6365 return "\\markdownRendererHardLineBreak\n{"
6366 end
```

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

```
6367 self.ellipsis = "\\markdownRendererEllipsis{"
```

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

```
6368 function self.thematic_break()
6369 if not self.is_writing then return "" end
6370 return "\\markdownRendererThematicBreak{"
6371 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.

```
6372 self.escaped_uri_chars = {
6373 [{""] = "\\markdownRendererLeftBrace{" ,
6374 ["}"] = "\\markdownRendererRightBrace{" ,
6375 [{"\\"}] = "\\markdownRendererBackslash{" ,
6376 [{"r"}] = " ",
6377 [{"n"}] = " ",
```

```

6378 }
6379 self.escaped_minimal_strings = {
6380 ["^"] = "\\markdownRendererCircumflex",
6381 .. "\\markdownRendererCircumflex ",
6382 ["☒"] = "\\markdownRendererTickedBox{}",
6383 ["☐"] = "\\markdownRendererHalfTickedBox{}",
6384 ["□"] = "\\markdownRendererUntickedBox{}",
6385 [entities.hex_entity('FFFD')]
6386 = "\\markdownRendererReplacementCharacter{}",
6387 }

```

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

```

6388 self.escaped_strings = util.table_copy(self.escaped_minimal_strings)
6389 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.

```

6390 self.escaped_chars = {
6391 [{""] = "\\markdownRendererLeftBrace{}",
6392 ["}"] = "\\markdownRendererRightBrace{}",
6393 [%"] = "\\markdownRendererPercentSign{}",
6394 [\\"] = "\\markdownRendererBackslash{}",
6395 [#"] = "\\markdownRendererHash{}",
6396 [$"] = "\\markdownRendererDollarSign{}",
6397 [&"] = "\\markdownRendererAmpersand{}",
6398 [_"] = "\\markdownRendererUnderscore{}",
6399 [^"] = "\\markdownRendererCircumflex{}",
6400 [~"] = "\\markdownRendererTilde{}",
6401 [|"] = "\\markdownRendererPipe{}",
6402 [entities.hex_entity('0000')]
6403 = "\\markdownRendererReplacementCharacter{}",
6404 }

```

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

```

6405 local function create_escaper(char_escapes, string_escapes)
6406 local escape = util.escaper(char_escapes, string_escapes)
6407 return function(s)
6408 if self.flatten_inlines then return s end
6409 return escape(s)
6410 end
6411 end
6412 local escape_typographic_text = create_escaper(
6413 self.escaped_chars, self.escaped_strings)
6414 local escape_programmatic_text = create_escaper(

```

```

6415 self.escaped_uri_chars, self.escaped_minimal_strings)
6416 local escape_minimal = create_escaper(
6417 {}, 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.

```

6418 self.escape = escape_typographic_text
6419 self.math = escape_minimal
6420 if options.hybrid then
6421 self.identifier = escape_minimal
6422 self.string = escape_minimal
6423 self.uri = escape_minimal
6424 self.infostring = escape_minimal
6425 else
6426 self.identifier = escape_programmatic_text
6427 self.string = escape_typographic_text
6428 self.uri = escape_programmatic_text
6429 self.infostring = escape_programmatic_text
6430 end

```

Define `writer->warning` as a function that will transform an input warning `t` with optional more warning text `m` to the output format.

```

6431 function self.warning(t, m)
6432 return {"\markdownRendererWarning{" , self.escape(t), "}{" ,
6433 escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
6434 escape_minimal(m or ""), "}"}
6435 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.

```

6436 function self.error(t, m)
6437 return {"\markdownRendererError{" , self.escape(t), "}{" ,
6438 escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
6439 escape_minimal(m or ""), "}"}
6440 end

```

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

```

6441 function self.code(s, attributes)
6442 if self.flatten_inlines then return s end
6443 local buf = {}
6444 if attributes ~= nil then
6445 table.insert(buf,
6446 "\\markdownRendererCodeSpanAttributeContextBegin\n")
6447 table.insert(buf, self.attributes(attributes))
6448 end
6449 table.insert(buf,
6450 {"\\markdownRendererCodeSpan{" , self.escape(s), "}"})
6451 if attributes ~= nil then
6452 table.insert(buf,
6453 "\\markdownRendererCodeSpanAttributeContextEnd{")
6454 end
6455 return buf
6456 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.

```

6457 function self.link(lab, src, tit, attributes)
6458 if self.flatten_inlines then return lab end
6459 local buf = {}
6460 if attributes ~= nil then
6461 table.insert(buf,
6462 "\\markdownRendererLinkAttributeContextBegin\n")
6463 table.insert(buf, self.attributes(attributes))
6464 end
6465 table.insert(buf, {"\\markdownRendererLink{" ,lab,"} ,
6466 {"",self.escape(src),"} ,
6467 {"",self.uri(src),"} ,
6468 {"",self.string(tit or ""),"}"}))
6469 if attributes ~= nil then
6470 table.insert(buf,
6471 "\\markdownRendererLinkAttributeContextEnd{")
6472 end
6473 return buf
6474 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.

```

6475 function self.image(lab, src, tit, attributes)
6476 if self.flatten_inlines then return lab end
6477 local buf = {}
6478 if attributes ~= nil then
6479 table.insert(buf,

```



```

6480 "\\markdownRendererImageAttributeContextBegin\n")
6481 table.insert(buf, self.attributes(attributes))
6482 end
6483 table.insert(buf, {"\\markdownRendererImage{" ,lab,"} ",
6484 {" ,self.string(src),"} ",
6485 {" ,self.uri(src),"} ",
6486 {" ,self.string(tit or ""),"} })
6487 if attributes ~= nil then
6488 table.insert(buf,
6489 "\\markdownRendererImageAttributeContextEnd{ }")
6490 end
6491 return buf
6492 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.

```

6493 function self.bulletlist(items,tight)
6494 if not self.is_writing then return "" end
6495 local buffer = {}
6496 for _,item in ipairs(items) do
6497 if item ~= "" then
6498 buffer[#buffer + 1] = self.bulletitem(item)
6499 end
6500 end
6501 local contents = util.intersperse(buffer,"\n")
6502 if tight and options.tightLists then
6503 return {"\\markdownRendererUlBeginTight\n",contents,
6504 "\n\\markdownRendererUlEndTight "}
6505 else
6506 return {"\\markdownRendererUlBegin\n",contents,
6507 "\n\\markdownRendererUlEnd "}
6508 end
6509 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.

```

6510 function self.bulletitem(s)
6511 return {"\\markdownRendererUlItem ",s,
6512 "\\markdownRendererUlItemEnd "}
6513 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.

```

6514 function self.orderedlist(items,tight,startnum)
6515 if not self.is_writing then return "" end

```

```

6516 local buffer = {}
6517 local num = startnum
6518 for _,item in ipairs(items) do
6519 if item ~= "" then
6520 buffer[#buffer + 1] = self.ordereditem(item,num)
6521 end
6522 if num ~= nil and item ~= "" then
6523 num = num + 1
6524 end
6525 end
6526 local contents = util.intersperse(buffer,"\n")
6527 if tight and options.tightLists then
6528 return {"\markdownRenderer01BeginTight\n",contents,
6529 "\n\markdownRenderer01EndTight "}
6530 else
6531 return {"\markdownRenderer01Begin\n",contents,
6532 "\n\markdownRenderer01End "}
6533 end
6534 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.

```

6535 function self.ordereditem(s,num)
6536 if num ~= nil then
6537 return {"\markdownRenderer01ItemWithNumber{" ,num,"} ",s,
6538 "\markdownRenderer01ItemEnd "}
6539 else
6540 return {"\markdownRenderer01Item ",s,
6541 "\markdownRenderer01ItemEnd "}
6542 end
6543 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.

```

6544 function self.inline_html_comment(contents)
6545 if self.flatten_inlines then return contents end
6546 return {"\markdownRendererInlineHtmlComment{" ,contents,"}"}
6547 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.

```

6548 function self.inline_html_tag(contents)
6549 if self.flatten_inlines then return contents end
6550 return {"\markdownRendererInlineHtmlTag{" ,
6551 self.string(contents),"}"}

```

```
6552 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.

```
6553 function self.block_html_element(s)
6554 if not self.is_writing then return "" end
6555 local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
6556 return {"\\markdownRendererInputBlockHtmlElement{" ,name,"}"}
6557 end
```

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

```
6558 function self.emphasis(s)
6559 if self.flatten_inlines then return s end
6560 return {"\\markdownRendererEmphasis{" ,s,"}"}
6561 end
```

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

```
6562 function self.checkbox(f)
6563 if f == 1.0 then
6564 return "☒ "
6565 elseif f == 0.0 then
6566 return "☐ "
6567 else
6568 return "◻ "
6569 end
6570 end
```

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

```
6571 function self.strong(s)
6572 if self.flatten_inlines then return s end
6573 return {"\\markdownRendererStrongEmphasis{" ,s,"}"}
6574 end
```

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

```
6575 function self.blockquote(s)
6576 if not self.is_writing then return "" end
6577 return {"\\markdownRendererBlockQuoteBegin\n",s,
6578 "\\markdownRendererBlockQuoteEnd "}
6579 end
```

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

```
6580 function self.verbatim(s)
6581 if not self.is_writing then return "" end
```

```

6582 s = s:gsub("\n$", "")
6583 local name = util.cache_verbatim(options.cacheDir, s)
6584 return {"\\markdownRendererInputVerbatim{" ,name,"}"}
6585 end

```

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

```

6586 function self.document(d)
6587 local buf = {"\\markdownRendererDocumentBegin\n"}
6588
6589 -- warn against the `hybrid` option
6590 if options.hybrid then
6591 local text = "The `hybrid` option has been soft-deprecated."
6592 local more = "Consider using one of the following better options "
6593 .. "for mixing TeX and markdown: `contentBlocks`, "
6594 .. "`rawAttribute`, `texComments`, `texMathDollars`, "
6595 .. "`texMathSingleBackslash`, and "
6596 .. "`texMathDoubleBackslash`."
6597 .. "For more information, see the user manual at "
6598 .. "<https://witiko.github.io/markdown/>."
6599 table.insert(buf, self.warning(text, more))
6600 end
6601
6602 -- insert the text of the document
6603 table.insert(buf, d)
6604
6605 -- pop all attributes
6606 table.insert(buf, self.pop_attributes())
6607
6608 table.insert(buf, "\\markdownRendererDocumentEnd")
6609
6610 return buf
6611 end

```

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

```

6612 local seen_identifiers = {}
6613 local key_value_regex = "([^\s=]+)%s*=%s*(.*)"
6614 local function normalize_attributes(attributes, auto_identifiers)
6615 -- normalize attributes
6616 local normalized_attributes = {}
6617 local has_explicit_identifiers = false
6618 local key, value
6619 for _, attribute in ipairs(attributes or {}) do
6620 if attribute:sub(1, 1) == "#" then
6621 table.insert(normalized_attributes, attribute)
6622 has_explicit_identifiers = true
6623 seen_identifiers[attribute:sub(2)] = true

```

```

6624 elseif attribute:sub(1, 1) == "." then
6625 table.insert(normalized_attributes, attribute)
6626 else
6627 key, value = attribute:match(key_value_regex)
6628 if key:lower() == "id" then
6629 table.insert(normalized_attributes, "#" .. value)
6630 elseif key:lower() == "class" then
6631 local classes = {}
6632 for class in value:gmatch("%S+") do
6633 table.insert(classes, class)
6634 end
6635 table.sort(classes)
6636 for _, class in ipairs(classes) do
6637 table.insert(normalized_attributes, "." .. class)
6638 end
6639 else
6640 table.insert(normalized_attributes, attribute)
6641 end
6642 end
6643 end
6644
6645 -- if no explicit identifiers exist, add auto identifiers
6646 if not has_explicit_identifiers and auto_identifiers ~= nil then
6647 local seen_auto_identifiers = {}
6648 for _, auto_identifier in ipairs(auto_identifiers) do
6649 if seen_auto_identifiers[auto_identifier] == nil then
6650 seen_auto_identifiers[auto_identifier] = true
6651 if seen_identifiers[auto_identifier] == nil then
6652 seen_identifiers[auto_identifier] = true
6653 table.insert(normalized_attributes,
6654 "#" .. auto_identifier)
6655 else
6656 local auto_identifier_number = 1
6657 while true do
6658 local numbered_auto_identifier = auto_identifier .. "-"
6659 .. auto_identifier_number
6660 if seen_identifiers[numbered_auto_identifier] == nil then
6661 seen_identifiers[numbered_auto_identifier] = true
6662 table.insert(normalized_attributes,
6663 "#" .. numbered_auto_identifier)
6664 break
6665 end
6666 auto_identifier_number = auto_identifier_number + 1
6667 end
6668 end
6669 end
6670 end

```

```

6671 end
6672
6673 -- sort and deduplicate normalized attributes
6674 table.sort(normalized_attributes)
6675 local seen_normalized_attributes = {}
6676 local deduplicated_normalized_attributes = {}
6677 for _, attribute in ipairs(normalized_attributes) do
6678 if seen_normalized_attributes[attribute] == nil then
6679 seen_normalized_attributes[attribute] = true
6680 table.insert(deduplicated_normalized_attributes, attribute)
6681 end
6682 end
6683
6684 return deduplicated_normalized_attributes
6685 end
6686
6687 function self.attributes(attributes, should_normalize_attributes)
6688 local normalized_attributes
6689 if should_normalize_attributes == false then
6690 normalized_attributes = attributes
6691 else
6692 normalized_attributes = normalize_attributes(attributes)
6693 end
6694
6695 local buf = {}
6696 local key, value
6697 for _, attribute in ipairs(normalized_attributes) do
6698 if attribute:sub(1, 1) == "#" then
6699 table.insert(buf, {"\\markdownRendererAttributeIdentifier{" ,
6700 attribute:sub(2), "}"})
6701 elseif attribute:sub(1, 1) == "." then
6702 table.insert(buf, {"\\markdownRendererAttributeName{" ,
6703 attribute:sub(2), "}"})
6704 else
6705 key, value = attribute:match(key_value_regex)
6706 table.insert(buf, {"\\markdownRendererAttributeKeyValue{" ,
6707 key, "}{" , value, "}"})
6708 end
6709 end
6710
6711 return buf
6712 end

```

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

```

6713 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`.

```
6714 self.attribute_type_levels = {}
6715 setmetatable(self.attribute_type_levels,
6716 { __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`.

```
6717 local function apply_attributes()
6718 local buf = {}
6719 for i = 1, #self.active_attributes do
6720 local start_output = self.active_attributes[i][3]
6721 if start_output ~= nil then
6722 table.insert(buf, start_output)
6723 end
6724 end
6725 return buf
6726 end
6727
6728 local function tear_down_attributes()
6729 local buf = {}
6730 for i = #self.active_attributes, 1, -1 do
6731 local end_output = self.active_attributes[i][4]
6732 if end_output ~= nil then
6733 table.insert(buf, end_output)
6734 end
6735 end
6736 return buf
6737 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.

```
6738 function self.push_attributes(attribute_type, attributes,
6739 start_output, end_output)
6740 local attribute_type_level
6741 = self.attribute_type_levels[attribute_type]
6742 self.attribute_type_levels[attribute_type]
6743 = attribute_type_level + 1
6744
6745 -- index attributes in a hash table for easy lookup
6746 attributes = attributes or {}
6747 for i = 1, #attributes do
6748 attributes[attributes[i]] = true
```

```

6749 end
6750
6751 local buf = {}
6752 -- handle slicing
6753 if attributes["#" .. self.slice_end_identifrier] ~= nil and
6754 self.slice_end_type == "^" then
6755 if self.is_writing then
6756 table.insert(buf, self.undosep())
6757 table.insert(buf, tear_down_attributes())
6758 end
6759 self.is_writing = false
6760 end
6761 if attributes["#" .. self.slice_begin_identifrier] ~= nil and
6762 self.slice_begin_type == "^" then
6763 table.insert(buf, apply_attributes())
6764 self.is_writing = true
6765 end
6766 if self.is_writing and start_output ~= nil then
6767 table.insert(buf, start_output)
6768 end
6769 table.insert(self.active_attributes,
6770 {attribute_type, attributes,
6771 start_output, end_output})
6772 return buf
6773 end
6774

```

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`).

```

6775 function self.pop_attributes(attribute_type)
6776 local buf = {}
6777 -- pop attributes until we find attributes of correct type
6778 -- or until no attributes remain
6779 local current_attribute_type = false
6780 while current_attribute_type ~= attribute_type and
6781 #self.active_attributes > 0 do
6782 local attributes, _, end_output
6783 current_attribute_type, attributes, _, end_output = table.unpack(
6784 self.active_attributes[#self.active_attributes])
6785 local attribute_type_level
6786 = self.attribute_type_levels[current_attribute_type]
6787 self.attribute_type_levels[current_attribute_type]
6788 = attribute_type_level - 1

```



```

6789 if self.is_writing and end_output ~= nil then
6790 table.insert(buf, end_output)
6791 end
6792 table.remove(self.active_attributes, #self.active_attributes)
6793 -- handle slicing
6794 if attributes["#" .. self.slice_end_identifer] ~= nil
6795 and self.slice_end_type == "$" then
6796 if self.is_writing then
6797 table.insert(buf, self.undosep())
6798 table.insert(buf, tear_down_attributes())
6799 end
6800 self.is_writing = false
6801 end
6802 if attributes["#" .. self.slice_begin_identifer] ~= nil and
6803 self.slice_begin_type == "$" then
6804 self.is_writing = true
6805 table.insert(buf, apply_attributes())
6806 end
6807 end
6808 return buf
6809 end

```

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

```

6810 local function create_auto_identifer(s)
6811 local buffer = {}
6812 local prev_space = false
6813 local letter_found = false
6814 local normalized_s = s
6815 if not options.unicodeNormalization
6816 or options.unicodeNormalizationForm ~= "nfc" then
6817 normalized_s = uni_algos.normalize.NFC(normalized_s)
6818 end
6819
6820 for _, code in utf8.codes(normalized_s) do
6821 local char = utf8.char(code)
6822
6823 -- Remove everything up to the first letter.
6824 if not letter_found then
6825 local is_letter = lpeg.match(
6826 parsers.unicode.following_alpha,
6827 char
6828)
6829 if is_letter then
6830 letter_found = true
6831 else
6832 goto continue
6833 end
6834 end

```

```

6835
6836 -- Remove all non-alphanumeric characters, except underscores,
6837 -- hyphens, and periods.
6838 if not lpeg.match(
6839 (parsers.underscore
6840 + parsers.dash
6841 + parsers.period
6842 + parsers.unicode.following_word
6843 + parsers.unicode.following_whitespace),
6844 char
6845) then
6846 goto continue
6847 end
6848
6849 -- Replace all spaces and newlines with hyphens.
6850 if lpeg.match(
6851 (parsers.newline
6852 + parsers.unicode.following_whitespace),
6853 char
6854) then
6855 char = "-"
6856 if prev_space then
6857 goto continue
6858 else
6859 prev_space = true
6860 end
6861 else
6862 -- Case-fold all alphabetic characters.
6863 char = uni_algos.case.casefold(char)
6864 prev_space = false
6865 end
6866
6867 table.insert(buffer, char)
6868
6869 ::continue::
6870 end
6871
6872 if prev_space then
6873 table.remove(buffer)
6874 end
6875
6876 local identifier = #buffer == 0 and "section"
6877 or table.concat(buffer, "")
6878 return identifier
6879 end

```

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

```
6880 local function create_gfm_auto_identifier(s)
6881 local buffer = {}
6882 local prev_space = false
6883 local letter_found = false
6884 local normalized_s = s
6885 if not options.unicodeNormalization
6886 or options.unicodeNormalizationForm ~= "nfc" then
6887 normalized_s = uni_algos.normalize.NFC(normalized_s)
6888 end
6889
6890 for _, code in utf8.codes(normalized_s) do
6891 local char = utf8.char(code)
6892
6893 -- Remove everything up to the first non-space.
6894 if not letter_found then
6895 local is_letter = not lpeg.match(
6896 parsers.unicode.following_whitespace,
6897 char
6898)
6899 if is_letter then
6900 letter_found = true
6901 else
6902 goto continue
6903 end
6904 end
6905
6906 -- Remove all non-alphanumeric characters, except underscores
6907 -- and hyphens.
6908 if not lpeg.match(
6909 (parsers.underscore
6910 + parsers.dash
6911 + parsers.unicode.following_word
6912 + parsers.unicode.following_whitespace),
6913 char
6914) then
6915 prev_space = false
6916 goto continue
6917 end
6918
6919 -- Replace all spaces and newlines with hyphens.
6920 if lpeg.match(
6921 (parsers.newline
6922 + parsers.unicode.following_whitespace),
6923 char
6924) then
```

```

6925 char = "-"
6926 if prev_space then
6927 goto continue
6928 else
6929 prev_space = true
6930 end
6931 else
6932 -- Case-fold all alphabetic characters.
6933 char = uni_algos.case.casefold(char)
6934 prev_space = false
6935 end
6936
6937 table.insert(buffer, char)
6938
6939 ::continue::
6940 end
6941
6942 if prev_space then
6943 table.remove(buffer)
6944 end
6945
6946 local identifier = #buffer == 0 and "section"
6947 or table.concat(buffer, "")
6948 return identifier
6949 end

```

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

```

6950 self.secbegin_text = "\\markdownRendererSectionBegin\n"
6951 self.secend_text = "\n\\markdownRendererSectionEnd "
6952 function self.heading(s, level, attributes)
6953 local buf = {}
6954 local flat_text, inlines = table.unpack(s)
6955
6956 -- push empty attributes for implied sections
6957 while self.attribute_type_levels["heading"] < level - 1 do
6958 table.insert(buf,
6959 self.push_attributes("heading",
6960 nil,
6961 self.secbegin_text,
6962 self.secend_text))
6963 end
6964
6965 -- pop attributes for sections that have ended
6966 while self.attribute_type_levels["heading"] >= level do
6967 table.insert(buf, self.pop_attributes("heading"))
6968 end

```

```

6969
6970 -- construct attributes for the new section
6971 local auto_identifiers = {}
6972 if self.options.autoIdentifiers then
6973 table.insert(auto_identifiers, create_auto_identifier(flat_text))
6974 end
6975 if self.options.gfmAutoIdentifiers then
6976 table.insert(auto_identifiers,
6977 create_gfm_auto_identifier(flat_text))
6978 end
6979 local normalized_attributes = normalize_attributes(attributes,
6980 auto_identifiers)
6981
6982 -- push attributes for the new section
6983 local start_output = {}
6984 local end_output = {}
6985 table.insert(start_output, self.secbegin_text)
6986 table.insert(end_output, self.secend_text)
6987
6988 table.insert(buf, self.push_attributes("heading",
6989 normalized_attributes,
6990 start_output,
6991 end_output))
6992 assert(self.attribute_type_levels["heading"] == level)
6993
6994 -- render the heading and its attributes
6995 if self.is_writing and #normalized_attributes > 0 then
6996 table.insert(buf,
6997 "\\markdownRendererHeaderAttributeContextBegin\n")
6998 table.insert(buf, self.attributes(normalized_attributes, false))
6999 end
7000
7001 local cmd
7002 level = level + options.shiftHeadings
7003 if level <= 1 then
7004 cmd = "\\markdownRendererHeadingOne"
7005 elseif level == 2 then
7006 cmd = "\\markdownRendererHeadingTwo"
7007 elseif level == 3 then
7008 cmd = "\\markdownRendererHeadingThree"
7009 elseif level == 4 then
7010 cmd = "\\markdownRendererHeadingFour"
7011 elseif level == 5 then
7012 cmd = "\\markdownRendererHeadingFive"
7013 elseif level >= 6 then
7014 cmd = "\\markdownRendererHeadingSix"
7015 else

```

```

7016 cmd = ""
7017 end
7018 if self.is_writing then
7019 table.insert(buf, {cmd, "{", inlines, "}"})
7020 end
7021
7022 if self.is_writing and #normalized_attributes > 0 then
7023 table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd{")
7024 end
7025
7026 return buf
7027 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.

```

7028 function self.get_state()
7029 return {
7030 is_writing=self.is_writing,
7031 flatten_inlines=self.flatten_inlines,
7032 active_attributes={table.unpack(self.active_attributes)},
7033 }
7034 end

```

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

```

7035 function self.set_state(s)
7036 local previous_state = self.get_state()
7037 for key, value in pairs(s) do
7038 self[key] = value
7039 end
7040 return previous_state
7041 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`.

```

7042 function self.defer_call(f)
7043 local previous_state = self.get_state()
7044 return function(...)
7045 local state = self.set_state(previous_state)
7046 local return_value = f(...)
7047 self.set_state(state)
7048 return return_value
7049 end
7050 end
7051
7052 return self
7053 end

```

### 3.1.4 Parsers

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

```
7054 parsers = {}
```

#### 3.1.4.1 Basic Parsers

```
7055 parsers.percent = P("%")
7056 parsers.at = P("@")
7057 parsers.comma = P(",")
7058 parsers.asterisk = P("*")
7059 parsers.dash = P("-")
7060 parsers.plus = P("+")
7061 parsers.underscore = P("_")
7062 parsers.period = P(".")
7063 parsers.hash = P("#")
7064 parsers.dollar = P("$")
7065 parsers.ampersand = P("&")
7066 parsers.backtick = P("`")
7067 parsers.less = P("<")
7068 parsers.more = P(">")
7069 parsers.space = P(" ")
7070 parsers.squote = P("'")
7071 parsers.dquote = P('"')
7072 parsers.lparent = P("(")
7073 parsers.rparent = P(")")
7074 parsers.lbracket = P("[")
7075 parsers.rbracket = P("]")
7076 parsers.lbrace = P("{")
7077 parsers.rbrace = P("}")
7078 parsers.circumflex = P("^")
7079 parsers.slash = P("/")
7080 parsers.equal = P("=")
7081 parsers.colon = P(":")
7082 parsers.semicolon = P(";")
7083 parsers.exclamation = P("!")
7084 parsers.pipe = P("|")
7085 parsers.tilde = P("~")
7086 parsers.backslash = P("\\")
7087 parsers.tab = P("\t")
7088 parsers.newline = P("\n")
7089
7090 parsers.digit = R("09")
7091 parsers.hexdigit = R("09", "af", "AF")
7092 parsers.letter = R("AZ", "az")
7093 parsers.alphanumeric = R("AZ", "az", "09")
```

```

7094 parsers.keyword = parsers.letter
7095 * (parsers.alphanumeric + parsers.dash)^0
7096
7097 parsers.doubleasterisks = P("**")
7098 parsers.doubleunderscores = P("__")
7099 parsers.doubletildes = P("~~")
7100 parsers.fourspaces = P(" ")
7101
7102 parsers.any = P(1)
7103 parsers.succeed = P(true)
7104 parsers.fail = P(false)
7105
7106 parsers.internal_punctuation = S(":,;.?.")
7107 parsers.ascii_punctuation = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")
7108

```

### 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.

```

7109 ;(function()
7110 local pathname = assert(kpse.find_file("UnicodeData.txt"),
7111 [[Could not locate file "UnicodeData.txt"]])
7112 local file = assert(io.open(pathname, "r"),
7113 [[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.

```

7114 local categories = {"L", "N", "P", "Pc", "S", "Z"}
7115 local prefix_trees = {}
7116 for _, category in ipairs(categories) do
7117 prefix_trees[category] = {}
7118 for char_length = 1, 4 do
7119 prefix_trees[category][char_length] = {}
7120 end
7121 end
7122 for line in file:lines() do
7123 local codepoint, full_category = line:match("^(%x+);[^;]*;(%)a*")
7124 assert(#full_category >= 1)
7125 local major_category = full_category:sub(1, 1)

```



```

7126 for _, category in ipairs({full_category, major_category}) do
7127 if prefix_trees[category] == nil then
7128 goto continue
7129 end
7130 local code = utf8.char(tonumber(codepoint, 16))
7131 local node = prefix_trees[category][#code]
7132 for i = 1, #code do
7133 local byte = code:sub(i, i)
7134 if i < #code then
7135 if node[byte] == nil then
7136 node[byte] = {}
7137 end
7138 node = node[byte]
7139 else
7140 table.insert(node, byte)
7141 end
7142 end
7143 ::continue::
7144 end
7145 end
7146 assert(file:close())
7147

```

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

```

7148 local function depth_first_search(node, path, visit, leave)
7149 visit(node, path)
7150 for label, child in pairs(node) do
7151 if type(child) == "table" then
7152 depth_first_search(child, path .. label, visit, leave)
7153 else
7154 visit(child, path)
7155 end
7156 end
7157 leave(node, path)
7158 end
7159
7160 print("M.categories = {}")
7161 print("local P = lpeg.P")
7162 print("local fail = P(false)")
7163 print("-- luacheck: push no max line length")
7164 for _, category in ipairs(categories) do
7165 print("M.categories." .. category .. " = {}")
7166 for length, prefix_tree in pairs(prefix_trees[category]) do
7167 local subparsers = {}
7168 depth_first_search(prefix_tree, "", function(node, path)
7169 if type(node) == "string" then
7170 local suffix
7171 if node == "]" then

```

```

7172 suffix = "P('" .. node .. "')"
7173 else
7174 suffix = "P([[" .. node .. "]])"
7175 end
7176 if subparsers[path] ~= nil then
7177 subparsers[path] = subparsers[path] .. " + " .. suffix
7178 else
7179 subparsers[path] = suffix
7180 end
7181 end
7182 end, function(_, path)
7183 if #path > 0 then
7184 local byte = path:sub(#path, #path)
7185 local parent_path = path:sub(1, #path-1)
7186 local prefix
7187 if byte == "]" then
7188 prefix = "P('" .. byte .. "')"
7189 else
7190 prefix = "P([[" .. byte .. "]])"
7191 end
7192 local suffix
7193 if subparsers[path]:find("%+ ") then
7194 suffix = prefix .. " * (" .. subparsers[path] .. ")"
7195 else
7196 suffix = prefix .. " * " .. subparsers[path]
7197 end
7198 if subparsers[parent_path] ~= nil then
7199 subparsers[parent_path] = subparsers[parent_path]
7200 .. " + " .. suffix
7201 else
7202 subparsers[parent_path] = suffix
7203 end
7204 else
7205 print(
7206 "M.categories." .. category .. "[" .. length .. "] = "
7207 .. (subparsers[path] or "fail")
7208)
7209 end
7210 end)
7211 end
7212 end
7213 print("-- luacheck: pop")
7214 end)()
7215 print("return M")

```

Back in the Markdown package, we will load the precompiled parsers of Unicode categories.

```

7216 local unicode_data = require("markdown-unicode-data")
7217 if metadata.version ~= unicode_data.metadata.version then
7218 util.warning(
7219 "markdown.lua " .. metadata.version .. " used with " ..
7220 "markdown-unicode-data.lua " .. unicode_data.metadata.version .. ".")
7221)
7222 end

```

Finally, we define high-level parsers for specific types of characters that are interesting for us.

```

7223 parsers.unicode = {}
7224 parsers.unicode.preceding_punctuation = parsers.fail
7225 parsers.unicode.following_punctuation = parsers.fail
7226 parsers.unicode.following_alpha = parsers.fail
7227 parsers.unicode.following_word = parsers.fail
7228 parsers.unicode.preceding_whitespace = parsers.fail
7229 parsers.unicode.following_whitespace = parsers.fail
7230 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))

```

7231 local punctuation_of_length_n
7232 = unicode_data.categories.P[n]
7233 + unicode_data.categories.S[n]
7234 parsers.unicode.preceding_punctuation
7235 = parsers.unicode.preceding_punctuation
7236 + B(punctuation_of_length_n)
7237 parsers.unicode.following_punctuation
7238 = parsers.unicode.following_punctuation
7239 + #punctuation_of_length_n

```

For alphabetical characters, accept any characters from Unicode category L (letter), similar to the character class ‘Unicode.

```

7240 local alpha_of_length_n = unicode_data.categories.L[n]
7241 parsers.unicode.following_alpha
7242 = parsers.unicode.following_alpha
7243 + 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 ‘

```

7244 local word_of_length_n
7245 = unicode_data.categories.L[n]
7246 + unicode_data.categories.N[n]

```

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

```

7247 + unicode_data.categories.Pc[n]
7248 parsers.unicode.following_word
7249 = parsers.unicode.following_word
7250 + 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.

```

7251 local whitespace_of_length_n = unicode_data.categories.Z[n]
7252 if n == 1 then
7253 whitespace_of_length_n
7254 = whitespace_of_length_n
7255 + R("\t\r")
7256 end
7257 parsers.unicode.preceding_whitespace
7258 = parsers.unicode.preceding_whitespace
7259 + B(whitespace_of_length_n)
7260 parsers.unicode.following_whitespace
7261 = parsers.unicode.following_whitespace
7262 + #whitespace_of_length_n
7263 end
7264
7265 parsers.escapable = parsers.ascii_punctuation
7266 parsers.anyescaped = parsers.backslash / ""
7267 * parsers.escapable
7268 + parsers.any
7269
7270 parsers.spacechar = S("\t ")
7271 parsers.spacing = S(" \n\r\t")
7272 parsers.nonspacechar = parsers.any - parsers.spacing
7273 parsers.optionalspace = parsers.spacechar^0
7274
7275 parsers.normalchar = parsers.any - (V("SpecialChar")
7276 + parsers.spacing)
7277 parsers.eof = -parsers.any
7278 parsers.nonindentSPACE = parsers.space^-3 * - parsers.spacechar
7279 parsers.indent = parsers.space^-3 * parsers.tab
7280 + parsers.fourspaces / ""
7281 parsers.linechar = P(1 - parsers.newline)
7282
7283 parsers.blankline = parsers.optionalspace
7284 * parsers.newline / "\n"
7285 parsers.blanklines = parsers.blankline^0
7286 parsers.skipblanklines = (parsers.optionalspace
7287 * parsers.newline)^0
7288 parsers.indentedline = parsers.indent / ""
7289 * C(parsers.linechar^1

```

```

7290 * parsers.newline^-1)
7291 parsers.optionallyindentedline = parsers.indent^-1 /""
7292 * C(parsers.linechar^1
7293 * parsers.newline^-1)
7294 parsers.sp = parsers.spacing^0
7295 parsers.spnl = parsers.optionalspace
7296 * (parsers.newline
7297 * parsers.optionalspace)^-1
7298 parsers.line = parsers.linechar^0 * parsers.newline
7299 parsers.nonemptyline = parsers.line - parsers.blankline

```

### 3.1.5.1 Parsers Used for Indentation

```

7300
7301 parsers.leader = parsers.space^-3
7302

```

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

```

7303 local function has_trail(indent_table)
7304 return indent_table ~= nil and
7305 indent_table.trail ~= nil and
7306 next(indent_table.trail) ~= nil
7307 end
7308

```

Check if indent table `indent_table` has any indents.

```

7309 local function has_indents(indent_table)
7310 return indent_table ~= nil and
7311 indent_table.indents ~= nil and
7312 next(indent_table.indents) ~= nil
7313 end
7314

```

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

```

7315 local function add_trail(indent_table, trail_info)
7316 indent_table.trail = trail_info
7317 return indent_table
7318 end
7319

```

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

```

7320 local function remove_trail(indent_table)
7321 indent_table.trail = nil
7322 return indent_table
7323 end
7324

```

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

```

7325 local function update_indent_table(indent_table, new_indent, add)

```

```

7326 indent_table = remove_trail(indent_table)
7327
7328 if not has_indents(indent_table) then
7329 indent_table.indents = {}
7330 end
7331
7332
7333 if add then
7334 indent_table.indents[#indent_table.indents + 1] = new_indent
7335 else
7336 if indent_table.indents[#indent_table.indents].name
7337 == new_indent.name then
7338 indent_table.indents[#indent_table.indents] = nil
7339 end
7340 end
7341
7342 return indent_table
7343 end
7344

```

Remove an indent by its name `name`.

```

7345 local function remove_indent(name)
7346 local remove_indent_level =
7347 function(s, i, indent_table) -- luacheck: ignore s i
7348 indent_table = update_indent_table(indent_table, {name=name},
7349 false)
7350 return true, indent_table
7351 end
7352
7353 return Cg(Cmt(Cb("indent_info"), remove_indent_level), "indent_info")
7354 end
7355

```

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`.

```

7356 local function process_starter_spacing(indent, spacing,
7357 minimum, left_strip_length)
7358 left_strip_length = left_strip_length or 0
7359
7360 local count = 0
7361 local tab_value = 4 - (indent) % 4
7362
7363 local code_started, minimum_found = false, false
7364 local code_start, minimum_remainder = "", ""

```

```

7365
7366 local left_total_stripped = 0
7367 local full_remainder = ""
7368
7369 if spacing ~= nil then
7370 for i = 1, #spacing do
7371 local character = spacing:sub(i, i)
7372
7373 if character == "\t" then
7374 count = count + tab_value
7375 tab_value = 4
7376 elseif character == " " then
7377 count = count + 1
7378 tab_value = 4 - (1 - tab_value) % 4
7379 end
7380
7381 if (left_strip_length ~= 0) then
7382 local possible_to_strip = math.min(count, left_strip_length)
7383 count = count - possible_to_strip
7384 left_strip_length = left_strip_length - possible_to_strip
7385 left_total_stripped = left_total_stripped + possible_to_strip
7386 else
7387 full_remainder = full_remainder .. character
7388 end
7389
7390 if (minimum_found) then
7391 minimum_remainder = minimum_remainder .. character
7392 elseif (count >= minimum) then
7393 minimum_found = true
7394 minimum_remainder = minimum_remainder
7395 .. string.rep(" ", count - minimum)
7396 end
7397
7398 if (code_started) then
7399 code_start = code_start .. character
7400 elseif (count >= minimum + 4) then
7401 code_started = true
7402 code_start = code_start
7403 .. string.rep(" ", count - (minimum + 4))
7404 end
7405 end
7406 end
7407
7408 local remainder
7409 if (code_started) then
7410 remainder = code_start
7411 else

```

```

7412 remainder = string.rep(" ", count - minimum)
7413 end
7414
7415 local is_minimum = count >= minimum
7416 return {
7417 is_code = code_started,
7418 remainder = remainder,
7419 left_total_stripped = left_total_stripped,
7420 is_minimum = is_minimum,
7421 minimum_remainder = minimum_remainder,
7422 total_length = count,
7423 full_remainder = full_remainder
7424 }
7425 end
7426

```

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

```

7427 local function count_indent_tab_level(indent_table)
7428 local count = 0
7429 if not has_indents(indent_table) then
7430 return count
7431 end
7432
7433 for i=1, #indent_table.indents do
7434 count = count + indent_table.indents[i].length
7435 end
7436 return count
7437 end
7438

```

Count the total width of a delimiter `delimiter`.

```

7439 local function total_delimiter_length(delimiter)
7440 local count = 0
7441 if type(delimiter) == "string" then return #delimiter end
7442 for _, value in pairs(delimiter) do
7443 count = count + total_delimiter_length(value)
7444 end
7445 return count
7446 end
7447

```

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`.

```

7448 local function process_starter_indent(_, _, indent_table, starter,
7449 is_blank, indent_type, breakable)
7450 local last_trail = starter[1]
7451 local delimiter = starter[2]
7452 local raw_new_trail = starter[3]

```



```

7453
7454 if indent_type == "bq" and not breakable then
7455 indent_table.ignore_blockquote_blank = true
7456 end
7457
7458 if has_trail(indent_table) then
7459 local trail = indent_table.trail
7460 if trail.is_code then
7461 return false
7462 end
7463 last_trail = trail.remainder
7464 else
7465 local sp = process_starter_spacing(0, last_trail, 0, 0)
7466
7467 if sp.is_code then
7468 return false
7469 end
7470 last_trail = sp.remainder
7471 end
7472
7473 local preceding_indentation = count_indent_tab_level(indent_table) % 4
7474 local last_trail_length = #last_trail
7475 local delimiter_length = total_delimiter_length(delimiter)
7476
7477 local total_indent_level = preceding_indentation + last_trail_length
7478 + delimiter_length
7479
7480 local sp = {}
7481 if not is_blank then
7482 sp = process_starter_spacing(total_indent_level, raw_new_trail,
7483 0, 1)
7484 end
7485
7486 local del_trail_length = sp.left_total_stripped
7487 if is_blank then
7488 del_trail_length = 1
7489 elseif not sp.is_code then
7490 del_trail_length = del_trail_length + #sp.remainder
7491 end
7492
7493 local indent_length = last_trail_length + delimiter_length
7494 + del_trail_length
7495 local new_indent_info = {name=indent_type, length=indent_length}
7496
7497 indent_table = update_indent_table(indent_table, new_indent_info,
7498 true)
7499 indent_table = add_trail(indent_table,

```

```

7500 {is_code=sp.is_code,
7501 remainder=sp.remainder,
7502 total_length=sp.total_length,
7503 full_remainder=sp.full_remainder})
7504
7505 return true, indent_table
7506 end
7507

```

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

```

7508 local function decode_pattern(name)
7509 local delimiter = parsers.succeed
7510 if name == "bq" then
7511 delimiter = parsers.more
7512 end
7513
7514 return C(parsers.optionalspace) * C(delimiter)
7515 * C(parsers.optionalspace) * Cp()
7516 end
7517

```

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

```

7518 local function left_blank_starter(indent_table)
7519 local blank_starter_index
7520
7521 if not has_indents(indent_table) then
7522 return
7523 end
7524
7525 for i = #indent_table.indents,1,-1 do
7526 local value = indent_table.indents[i]
7527 if value.name == "li" then
7528 blank_starter_index = i
7529 else
7530 break
7531 end
7532 end
7533
7534 return blank_starter_index
7535 end
7536

```

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.

```

7537 local function traverse_indent(s, i, indent_table, is_optional,
7538 is_blank, current_line_indents)
7539 local new_index = i
7540
7541 local preceding_indentation = 0
7542 local current_trail = {}
7543
7544 local blank_starter = left_blank_starter(indent_table)
7545
7546 if current_line_indents == nil then
7547 current_line_indents = {}
7548 end
7549
7550 for index = 1,#indent_table.indents do
7551 local value = indent_table.indents[index]
7552 local pattern = decode_pattern(value.name)
7553
7554 -- match decoded pattern
7555 local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
7556 if new_indent_info == nil then
7557 local blankline_end = lpeg.match(
7558 Ct(parsers.blankline * Cg(Cp(), "pos")), s, new_index)
7559 if is_optional or not indent_table.ignore_blockquote_blank
7560 or not blankline_end then
7561 return is_optional, new_index, current_trail,
7562 current_line_indents
7563 end
7564
7565 return traverse_indent(s, tonumber(blankline_end.pos),
7566 indent_table, is_optional, is_blank,
7567 current_line_indents)
7568 end
7569
7570 local raw_last_trail = new_indent_info[1]
7571 local delimiter = new_indent_info[2]
7572 local raw_new_trail = new_indent_info[3]
7573 local next_index = new_indent_info[4]
7574
7575 local space_only = delimiter == ""
7576
7577 -- check previous trail
7578 if not space_only and next(current_trail) == nil then
7579 local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
7580 current_trail = {is_code=sp.is_code, remainder=sp.remainder,
7581 total_length=sp.total_length,
7582 full_remainder=sp.full_remainder}
7583 end

```

```

7584
7585 if next(current_trail) ~= nil then
7586 if not space_only and current_trail.is_code then
7587 return is_optional, new_index, current_trail,
7588 current_line_indents
7589 end
7590 if current_trail.internal_remainder ~= nil then
7591 raw_last_trail = current_trail.internal_remainder
7592 end
7593 end
7594
7595 local raw_last_trail_length = 0
7596 local delimiter_length = 0
7597
7598 if not space_only then
7599 delimiter_length = #delimiter
7600 raw_last_trail_length = #raw_last_trail
7601 end
7602
7603 local total_indent_level = preceding_indentation
7604 + raw_last_trail_length + delimiter_length
7605
7606 local spacing_to_process
7607 local minimum = 0
7608 local left_strip_length = 0
7609
7610 if not space_only then
7611 spacing_to_process = raw_new_trail
7612 left_strip_length = 1
7613 else
7614 spacing_to_process = raw_last_trail
7615 minimum = value.length
7616 end
7617
7618 local sp = process_starter_spacing(total_indent_level,
7619 spacing_to_process, minimum,
7620 left_strip_length)
7621
7622 if space_only and not sp.is_minimum then
7623 return is_optional or (is_blank and blank_starter <= index),
7624 new_index, current_trail, current_line_indents
7625 end
7626
7627 local indent_length = raw_last_trail_length + delimiter_length
7628 + sp.left_total_stripped
7629
7630 -- update info for the next pattern

```

```

7631 if not space_only then
7632 preceding_indentation = preceding_indentation + indent_length
7633 else
7634 preceding_indentation = preceding_indentation + value.length
7635 end
7636
7637 current_trail = {is_code=sp.is_code, remainder=sp.remainder,
7638 internal_remainder=sp.minimum_remainder,
7639 total_length=sp.total_length,
7640 full_remainder=sp.full_remainder}
7641
7642 current_line_indents[#current_line_indents + 1] = new_indent_info
7643 new_index = next_index
7644 end
7645
7646 return true, new_index, current_trail, current_line_indents
7647 end
7648

```

Check if a code trail is expected.

```

7649 local function check_trail(expect_code, is_code)
7650 return (expect_code and is_code) or (not expect_code and not is_code)
7651 end
7652

```

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`.

```

7653 local check_trail_joined =
7654 function(s, i, indent_table, -- luacheck: ignore s i
7655 spacing, expect_code, omit_remainder)
7656 local is_code
7657 local remainder
7658
7659 if has_trail(indent_table) then
7660 local trail = indent_table.trail
7661 is_code = trail.is_code
7662 if is_code then
7663 remainder = trail.remainder
7664 else
7665 remainder = trail.full_remainder
7666 end
7667 else
7668 local sp = process_starter_spacing(0, spacing, 0, 0)
7669 is_code = sp.is_code
7670 if is_code then
7671 remainder = sp.remainder
7672 else

```

```

7673 remainder = sp.full_remainder
7674 end
7675 end
7676
7677 local result = check_trail(expect_code, is_code)
7678 if omit_remainder then
7679 return result
7680 end
7681 return result, remainder
7682 end
7683

```

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

```

7684 local check_trail_length =
7685 function(s, i, indent_table, -- luacheck: ignore s i
7686 spacing, min, max)
7687 local trail
7688
7689 if has_trail(indent_table) then
7690 trail = indent_table.trail
7691 else
7692 trail = process_starter_spacing(0, spacing, 0, 0)
7693 end
7694
7695 local total_length = trail.total_length
7696 if total_length == nil then
7697 return false
7698 end
7699
7700 return min <= total_length and total_length <= max
7701 end
7702

```

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

```

7703 local function check_continuation_indentation(s, i, indent_table,
7704 is_optional, is_blank)
7705 if not has_indents(indent_table) then
7706 return true
7707 end
7708
7709 local passes, new_index, current_trail, current_line_indents =
7710 traverse_indent(s, i, indent_table, is_optional, is_blank)
7711
7712 if passes then
7713 indent_table.current_line_indents = current_line_indents
7714 indent_table = add_trail(indent_table, current_trail)
7715 end
7716 return new_index, indent_table

```

```

7716 end
7717 return false
7718 end
7719

```

Get name of the last indent from the `indent_table`.

```

7720 local function get_last_indent_name(indent_table)
7721 if has_indents(indent_table) then
7722 return indent_table.indents[#indent_table.indents].name
7723 end
7724 end
7725

```

Remove the remainder altogether if the last indent from the `indent_table` is blank-only.

```

7726 local function remove_remainder_if_blank(indent_table, remainder)
7727 if get_last_indent_name(indent_table) == "li" then
7728 return ""
7729 end
7730 return remainder
7731 end
7732

```

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.

```

7733 local check_trail_type =
7734 function(s, i, -- luacheck: ignore s i
7735 trail, spacing, trail_type)
7736 if trail == nil then
7737 trail = process_starter_spacing(0, spacing, 0, 0)
7738 end
7739
7740 if trail_type == "non-code" then
7741 return check_trail(false, trail.is_code)
7742 end
7743 if trail_type == "code" then
7744 return check_trail(true, trail.is_code)
7745 end
7746 if trail_type == "full-code" then
7747 if (trail.is_code) then
7748 return i, trail.remainder
7749 end
7750 return i, ""
7751 end
7752 if trail_type == "full-any" then
7753 return i, trail.internal_remainder
7754 end
7755 end

```

7756

Stores or restores an `is_freezing` trail from indent table `indent_table`.

```
7757 local trail_freezing =
7758 function(s, i, -- luacheck: ignore s i
7759 indent_table, is_freezing)
7760 if is_freezing then
7761 if indent_table.is_trail_frozen then
7762 indent_table.trail = indent_table.frozen_trail
7763 else
7764 indent_table.frozen_trail = indent_table.trail
7765 indent_table.is_trail_frozen = true
7766 end
7767 else
7768 indent_table.frozen_trail = nil
7769 indent_table.is_trail_frozen = false
7770 end
7771 return true, indent_table
7772 end
7773
```

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`.

```
7774 local check_continuation_indentation_and_trail =
7775 function (s, i, indent_table, is_optional, is_blank, trail_type,
7776 reset_rem, omit_remainder)
7777 if not has_indents(indent_table) then
7778 local spacing, new_index = lpeg.match(C(parsers.spacechar^0)
7779 * Cp(), s, i)
7780 local result, remainder = check_trail_type(s, i,
7781 indent_table.trail, spacing, trail_type)
7782 if remainder == nil then
7783 if result then
7784 return new_index
7785 end
7786 return false
7787 end
7788 if result then
7789 return new_index, remainder
7790 end
7791 return false
7792 end
7793
7794 local passes, new_index, current_trail = traverse_indent(s, i,
7795 indent_table, is_optional, is_blank)
7796
7797 if passes then
```



```

7798 local spacing
7799 if current_trail == nil then
7800 local newer_spacing, newer_index = lpeg.match(
7801 C(parsers.spacechar^0) * Cp(), s, i)
7802 current_trail = process_starter_spacing(0, newer_spacing, 0, 0)
7803 new_index = newer_index
7804 spacing = newer_spacing
7805 else
7806 spacing = current_trail.remainder
7807 end
7808 local result, remainder = check_trail_type(s, new_index,
7809 current_trail, spacing, trail_type)
7810 if remainder == nil or omit_remainder then
7811 if result then
7812 return new_index
7813 end
7814 return false
7815 end
7816
7817 if is_blank and reset_rem then
7818 remainder = remove_remainder_if_blank(indent_table, remainder)
7819 end
7820 if result then
7821 return new_index, remainder
7822 end
7823 return false
7824 end
7825 return false
7826 end
7827

```

The following patterns check whitespace indentation at the start of a block.

```

7828 parsers.check_trail = Cmt(Cb("indent_info") * C(parsers.spacechar^0)
7829 * Cc(false), check_trail_joined)
7830
7831 parsers.check_trail_no_rem = Cmt(Cb("indent_info")
7832 * C(parsers.spacechar^0) * Cc(false)
7833 * Cc(true), check_trail_joined)
7834
7835 parsers.check_code_trail = Cmt(Cb("indent_info")
7836 * C(parsers.spacechar^0)
7837 * Cc(true), check_trail_joined)
7838
7839 parsers.check_trail_length_range = function(min, max)
7840 return Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(min)
7841 * Cc(max), check_trail_length)
7842 end
7843

```

```

7844 parsers.check_trail_length = function(n)
7845 return parsers.check_trail_length_range(n, n)
7846 end
7847

```

The following patterns handle trail backup, to prevent a failing pattern to modify it before passing it to the next.

```

7848 parsers.freeze_trail = Cg(Cmt(Cb("indent_info")
7849 * Cc(true), trail_freezing), "indent_info")
7850
7851 parsers.unfreeze_trail = Cg(Cmt(Cb("indent_info") * Cc(false),
7852 trail_freezing), "indent_info")
7853

```

The following patterns check indentation in continuation lines as defined by the container start.

```

7854 parsers.check_minimal_indent = Cmt(Cb("indent_info") * Cc(false),
7855 check_continuation_indentation)
7856
7857 parsers.check_optional_indent = Cmt(Cb("indent_info") * Cc(true),
7858 check_continuation_indentation)
7859
7860 parsers.check_minimal_blank_indent
7861 = Cmt(Cb("indent_info") * Cc(false)
7862 * Cc(true)
7863 , check_continuation_indentation)
7864

```

The following patterns check indentation in continuation lines as defined by the container start. Additionally the subsequent trail is also directly checked.

```

7865
7866 parsers.check_minimal_indent_and_trail =
7867 Cmt(Cb("indent_info")
7868 * Cc(false) * Cc(false) * Cc("non-code") * Cc(true)
7869 , check_continuation_indentation_and_trail)
7870
7871 parsers.check_minimal_indent_and_code_trail =
7872 Cmt(Cb("indent_info")
7873 * Cc(false) * Cc(false) * Cc("code") * Cc(false)
7874 , check_continuation_indentation_and_trail)
7875
7876 parsers.check_minimal_blank_indent_and_full_code_trail =
7877 Cmt(Cb("indent_info")
7878 * Cc(false) * Cc(true) * Cc("full-code") * Cc(true)
7879 , check_continuation_indentation_and_trail)
7880
7881 parsers.check_minimal_indent_and_any_trail =
7882 Cmt(Cb("indent_info")

```

```

7883 * Cc(false) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
7884 , check_continuation_indentation_and_trail)
7885
7886 parsers.check_minimal_blank_indent_and_any_trail =
7887 Cmt(Cb("indent_info")
7888 * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
7889 , check_continuation_indentation_and_trail)
7890
7891 parsers.check_minimal_blank_indent_and_any_trail_no_rem =
7892 Cmt(Cb("indent_info")
7893 * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(true)
7894 , check_continuation_indentation_and_trail)
7895
7896 parsers.check_optional_indent_and_any_trail =
7897 Cmt(Cb("indent_info")
7898 * Cc(true) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
7899 , check_continuation_indentation_and_trail)
7900
7901 parsers.check_optional_blank_indent_and_any_trail =
7902 Cmt(Cb("indent_info")
7903 * Cc(true) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
7904 , check_continuation_indentation_and_trail)
7905

```

The following patterns specify behaviour around newlines.

```

7906
7907 parsers.spnlc_noexc = parsers.optionalspace
7908 * (parsers.newline
7909 * parsers.check_minimal_indent_and_any_trail)^-1
7910
7911 parsers.spnlc = parsers.optionalspace
7912 * (V("EndlineNoSub"))^-1
7913
7914 parsers.spnlc_sep = parsers.optionalspace * V("EndlineNoSub")
7915 + parsers.spacechar^1
7916
7917 parsers.only_blank = parsers.spacechar^0
7918 * (parsers.newline + parsers.eof)
7919

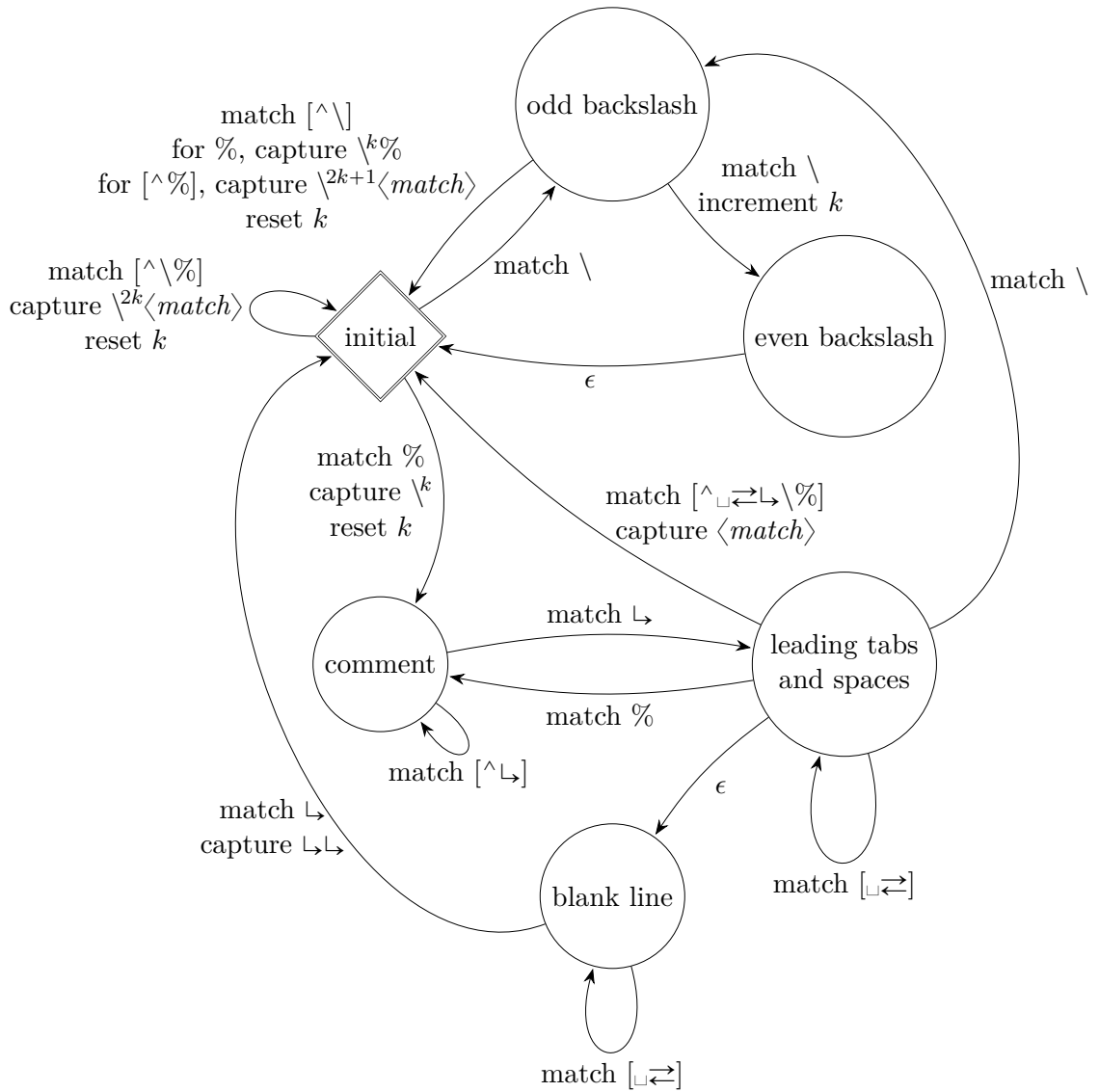
```

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.

```

7920 parsers.commented_line_letter = parsers.linechar
7921 + parsers.newline
7922 - parsers.backslash
7923 - parsers.percent
7924 parsers.commented_line = Cg(Cc(""), "backslashes")
7925 * ((#(parsers.commented_line_letter

```



**Figure 8: A pushdown automaton that recognizes TeX comments**

```

7926 - parsers.newline)
7927 * Cb("backslashes")
7928 * Cs(parsers.commented_line_letter
7929 - parsers.newline)^1 -- initial
7930 * Cg(Cc(""), "backslashes"))
7931 + #(parsers.backslash
7932 * (parsers.backslash + parsers.newline))
7933 * Cg((parsers.backslash -- even backslash
7934 * (parsers.backslash
7935 + #parsers.newline))^1, "backslashes")
7936 + (parsers.backslash
7937 * (#parsers.percent
7938 * Cb("backslashes")
7939 / function(backslashes)
7940 return string.rep("\\", #backslashes / 2)
7941 end
7942 * C(parsers.percent)
7943 + #parsers.commented_line_letter
7944 * Cb("backslashes")
7945 * Cc("\\")
7946 * C(parsers.commented_line_letter))
7947 * Cg(Cc(""), "backslashes"))^0
7948 * (#parsers.percent
7949 * Cb("backslashes")
7950 / function(backslashes)
7951 return string.rep("\\", #backslashes / 2)
7952 end
7953 * ((parsers.percent -- comment
7954 * parsers.line
7955 * #parsers.blankline) -- blank line
7956 / "\n"
7957 + parsers.percent -- comment
7958 * parsers.line
7959 * parsers.optionalspace) -- leading spaces
7960 + #(parsers.newline)
7961 * Cb("backslashes")
7962 * C(parsers.newline))
7963
7964 parsers.chunk = parsers.line * (parsers.optionallyindentedline
7965 - parsers.blankline)^0
7966
7967 parsers.attribute_key_char = parsers.alphanumeric + S("-_:.")
7968 parsers.attribute_raw_char = parsers.alphanumeric + S("-_")
7969 parsers.attribute_key = (parsers.attribute_key_char
7970 - parsers.dash - parsers.digit)
7971 * parsers.attribute_key_char^0
7972 parsers.attribute_value = (parsers.dquote / ""))

```

```

7973 * (parsers.anyescaped - parsers.dquote)^0
7974 * (parsers.dquote / ""))
7975 + ((parsers.squote / ""))
7976 * (parsers.anyescaped - parsers.squote)^0
7977 * (parsers.squote / ""))
7978 + (parsers.anyescaped
7979 - parsers.dquote
7980 - parsers.rbrace
7981 - parsers.space)^0
7982 parsers.attribute_identifier = parsers.attribute_key_char^1
7983 parsers.attribute_classname = parsers.letter
7984 * parsers.attribute_key_char^0
7985 parsers.attribute_raw = parsers.attribute_raw_char^1
7986
7987 parsers.attribute = (parsers.dash * Cc(".unnumbered"))
7988 + C(parsers.hash
7989 * parsers.attribute_identifier)
7990 + C(parsers.period
7991 * parsers.attribute_classname)
7992 + Cs(parsers.attribute_key
7993 * parsers.optionalspace
7994 * parsers.equal
7995 * parsers.optionalspace
7996 * parsers.attribute_value)
7997 parsers.attributes = parsers.lbrace
7998 * parsers.optionalspace
7999 * parsers.attribute
8000 * (parsers.spacechar^1
8001 * parsers.attribute)^0
8002 * parsers.optionalspace
8003 * parsers.rbrace
8004
8005 parsers.raw_attribute = parsers.lbrace
8006 * parsers.optionalspace
8007 * parsers.equal
8008 * C(parsers.attribute_raw)
8009 * parsers.optionalspace
8010 * parsers.rbrace
8011
8012 -- block followed by 0 or more optionally
8013 -- indented blocks with first line indented.
8014 parsers.indented_blocks = function(bl)
8015 return Cs(bl
8016 * (parsers.blankline^1
8017 * parsers.indent
8018 * -parsers.blankline
8019 * bl)^0

```

```

8020 * (parsers.blankline^1 + parsers.eof))
8021 end

```

### 3.1.5.2 Parsers Used for HTML Entities

```

8022 local function repeat_between(pattern, min, max)
8023 return -pattern^(max + 1) * pattern^min
8024 end
8025
8026 parsers.hexentity = parsers.ampersand * parsers.hash * C(S("Xx"))
8027 * C(repeat_between(parsers.hexdigit, 1, 6))
8028 * parsers.semicolon
8029 parsers.decentity = parsers.ampersand * parsers.hash
8030 * C(repeat_between(parsers.digit, 1, 7))
8031 * parsers.semicolon
8032 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
8033 * parsers.semicolon
8034
8035 parsers.html_entities
8036 = parsers.hexentity / entities.hex_entity_with_x_char
8037 + parsers.decentity / entities.dec_entity
8038 + parsers.tagentity / entities.char_entity

```

### 3.1.5.3 Parsers Used for Markdown Lists

```

8039 parsers.bullet = function(bullet_char, interrupting)
8040 local allowed_end
8041 if interrupting then
8042 allowed_end = C(parsers.spacechar^1) * #parsers.linechar
8043 else
8044 allowed_end = C(parsers.spacechar^1)
8045 + #(parsers.newline + parsers.eof)
8046 end
8047 return parsers.check_trail
8048 * Ct(C(bullet_char) * Cc(""))
8049 * allowed_end
8050 end
8051
8052 local function tickbox(interior)
8053 return parsers.optionalspace * parsers.lbracket
8054 * interior * parsers.rbracket * parsers.spacechar^1
8055 end
8056
8057 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
8058 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
8059 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
8060

```

### 3.1.5.4 Parsers Used for Markdown Code Spans

```
8061 parsers.openticks = Cg(parsers.backtick^1, "ticks")
8062
8063 local function captures_equal_length(_,i,a,b)
8064 return #a == #b and i
8065 end
8066
8067 parsers.closeticks = Cmt(C(parsers.backtick^1)
8068 * Cb("ticks"), captures_equal_length)
8069
8070 parsers.intickschar = (parsers.any - S("\n\r`"))
8071 + V("NoSoftLineBreakEndline")
8072 + (parsers.backtick^1 - parsers.closeticks)
8073
8074 local function process_inticks(s)
8075 s = s:gsub("\n", " ")
8076 s = s:gsub("^ (.*) $", "%1")
8077 return s
8078 end
8079
8080 parsers.inticks = parsers.openticks
8081 * C(parsers.space^0)
8082 * parsers.closeticks
8083 + parsers.openticks
8084 * Cs(Cs(parsers.intickschar^0) / process_inticks)
8085 * parsers.closeticks
8086
```

### 3.1.5.5 Parsers Used for HTML

```
8087 -- case-insensitive match (we assume s is lowercase)
8088 -- must be single byte encoding
8089 parsers.keyword_exact = function(s)
8090 local parser = P(0)
8091 for i=1,#s do
8092 local c = s:sub(i,i)
8093 local m = c .. upper(c)
8094 parser = parser * S(m)
8095 end
8096 return parser
8097 end
8098
8099 parsers.special_block_keyword =
8100 parsers.keyword_exact("pre") +
8101 parsers.keyword_exact("script") +
8102 parsers.keyword_exact("style") +
8103 parsers.keyword_exact("textarea")
```



```
8104
8105 parsers.block_keyword =
8106 parsers.keyword_exact("address") +
8107 parsers.keyword_exact("article") +
8108 parsers.keyword_exact("aside") +
8109 parsers.keyword_exact("base") +
8110 parsers.keyword_exact("basefont") +
8111 parsers.keyword_exact("blockquote") +
8112 parsers.keyword_exact("body") +
8113 parsers.keyword_exact("caption") +
8114 parsers.keyword_exact("center") +
8115 parsers.keyword_exact("col") +
8116 parsers.keyword_exact("colgroup") +
8117 parsers.keyword_exact("dd") +
8118 parsers.keyword_exact("details") +
8119 parsers.keyword_exact("dialog") +
8120 parsers.keyword_exact("dir") +
8121 parsers.keyword_exact("div") +
8122 parsers.keyword_exact("dl") +
8123 parsers.keyword_exact("dt") +
8124 parsers.keyword_exact("fieldset") +
8125 parsers.keyword_exact("figcaption") +
8126 parsers.keyword_exact("figure") +
8127 parsers.keyword_exact("footer") +
8128 parsers.keyword_exact("form") +
8129 parsers.keyword_exact("frame") +
8130 parsers.keyword_exact("frameset") +
8131 parsers.keyword_exact("h1") +
8132 parsers.keyword_exact("h2") +
8133 parsers.keyword_exact("h3") +
8134 parsers.keyword_exact("h4") +
8135 parsers.keyword_exact("h5") +
8136 parsers.keyword_exact("h6") +
8137 parsers.keyword_exact("head") +
8138 parsers.keyword_exact("header") +
8139 parsers.keyword_exact("hr") +
8140 parsers.keyword_exact("html") +
8141 parsers.keyword_exact("iframe") +
8142 parsers.keyword_exact("legend") +
8143 parsers.keyword_exact("li") +
8144 parsers.keyword_exact("link") +
8145 parsers.keyword_exact("main") +
8146 parsers.keyword_exact("menu") +
8147 parsers.keyword_exact("menuitem") +
8148 parsers.keyword_exact("nav") +
8149 parsers.keyword_exact("noframes") +
8150 parsers.keyword_exact("ol") +
```

```

8151 parsers.keyword_exact("optgroup") +
8152 parsers.keyword_exact("option") +
8153 parsers.keyword_exact("p") +
8154 parsers.keyword_exact("param") +
8155 parsers.keyword_exact("section") +
8156 parsers.keyword_exact("source") +
8157 parsers.keyword_exact("summary") +
8158 parsers.keyword_exact("table") +
8159 parsers.keyword_exact("tbody") +
8160 parsers.keyword_exact("td") +
8161 parsers.keyword_exact("tfoot") +
8162 parsers.keyword_exact("th") +
8163 parsers.keyword_exact("thead") +
8164 parsers.keyword_exact("title") +
8165 parsers.keyword_exact("tr") +
8166 parsers.keyword_exact("track") +
8167 parsers.keyword_exact("ul")
8168
8169 -- end conditions
8170 parsers.html_blankline_end_condition
8171 = parsers.linechar^0
8172 * (parsers.newline
8173 * (parsers.check_minimal_blank_indent_and_any_trail
8174 * #parsers.blankline
8175 + parsers.check_minimal_indent_and_any_trail)
8176 * parsers.linechar^1)^0
8177 * (parsers.newline^-1 / "")
8178
8179 local function remove_trailing_blank_lines(s)
8180 return s:gsub("[\n\r]+%s*$", "")
8181 end
8182
8183 parsers.html_until_end = function(end_marker)
8184 return Cs(Cs((parsers.newline
8185 * (parsers.check_minimal_blank_indent_and_any_trail
8186 * #parsers.blankline
8187 + parsers.check_minimal_indent_and_any_trail)
8188 + parsers.linechar - end_marker)^0
8189 * parsers.linechar^0 * parsers.newline^-1)
8190 / remove_trailing_blank_lines)
8191 end
8192
8193 -- attributes
8194 parsers.html_attribute_spacing = parsers.optionalspace
8195 * V("NoSoftLineBreakEndline")
8196 * parsers.optionalspace
8197 + parsers.spacechar^1

```

```

8198
8199 parsers.html_attribute_name = (parsers.letter
8200 + parsers.colon
8201 + parsers.underscore)
8202 * (parsers.alphanumeric
8203 + parsers.colon
8204 + parsers.underscore
8205 + parsers.period
8206 + parsers.dash)^0
8207
8208 parsers.html_attribute_value = parsers.squote
8209 * (parsers.linechar - parsers.squote)^0
8210 * parsers.squote
8211 + parsers.dquote
8212 * (parsers.linechar - parsers.dquote)^0
8213 * parsers.dquote
8214 + (parsers.any
8215 - parsers.spacechar
8216 - parsers.newline
8217 - parsers.dquote
8218 - parsers.squote
8219 - parsers.backtick
8220 - parsers.equal
8221 - parsers.less
8222 - parsers.more)^1
8223
8224 parsers.html_inline_attribute_value = parsers.squote
8225 * (V("NoSoftLineBreakEndline")
8226 + parsers.any
8227 - parsers.blankline^2
8228 - parsers.squote)^0
8229 * parsers.squote
8230 + parsers.dquote
8231 * (V("NoSoftLineBreakEndline")
8232 + parsers.any
8233 - parsers.blankline^2
8234 - parsers.dquote)^0
8235 * parsers.dquote
8236 + (parsers.any
8237 - parsers.spacechar
8238 - parsers.newline
8239 - parsers.dquote
8240 - parsers.squote
8241 - parsers.backtick
8242 - parsers.equal
8243 - parsers.less
8244 - parsers.more)^1

```

```

8245
8246 parsers.html_attribute_value_specification
8247 = parsers.optionalspace
8248 * parsers.equal
8249 * parsers.optionalspace
8250 * parsers.html_attribute_value
8251
8252 parsers.html_spn1 = parsers.optionalspace
8253 * (V("NoSoftLineBreakEndline")
8254 * parsers.optionalspace)^-1
8255
8256 parsers.html_inline_attribute_value_specification
8257 = parsers.html_spn1
8258 * parsers.equal
8259 * parsers.html_spn1
8260 * parsers.html_inline_attribute_value
8261
8262 parsers.html_attribute
8263 = parsers.html_attribute_spacing
8264 * parsers.html_attribute_name
8265 * parsers.html_inline_attribute_value_specification^-1
8266
8267 parsers.html_non_newline_attribute
8268 = parsers.spacechar^1
8269 * parsers.html_attribute_name
8270 * parsers.html_attribute_value_specification^-1
8271
8272 parsers.nested_breaking_blank = parsers.newline
8273 * parsers.check_minimal_blank_indent
8274 * parsers.blankline
8275
8276 parsers.html_comment_start = P("<!--")
8277
8278 parsers.html_comment_end = P("-->")
8279
8280 parsers.html_comment
8281 = Cs(parsers.html_comment_start
8282 * parsers.html_until_end(parsers.html_comment_end))
8283
8284 parsers.html_inline_comment = (parsers.html_comment_start / "")
8285 * -P(">") * -P("-->")
8286 * Cs((V("NoSoftLineBreakEndline")
8287 + parsers.any
8288 - parsers.nested_breaking_blank
8289 - parsers.html_comment_end)^0)
8290 * (parsers.html_comment_end / "")
8291

```

```

8292 parsers.html_cdatasection_start = P("<![CDATA[(")
8293
8294 parsers.html_cdatasection_end = P("]]>")
8295
8296 parsers.html_cdatasection
8297 = Cs(parsers.html_cdatasection_start
8298 * parsers.html_until_end(parsers.html_cdatasection_end))
8299
8300 parsers.html_inline_cdatasection
8301 = parsers.html_cdatasection_start
8302 * Cs(V("NoSoftLineBreakEndline") + parsers.any
8303 - parsers.nested_breaking_blank - parsers.html_cdatasection_end)^0
8304 * parsers.html_cdatasection_end
8305
8306 parsers.html_declaration_start = P("<!") * parsers.letter
8307
8308 parsers.html_declaration_end = P(">")
8309
8310 parsers.html_declaration
8311 = Cs(parsers.html_declaration_start
8312 * parsers.html_until_end(parsers.html_declaration_end))
8313
8314 parsers.html_inline_declaration
8315 = parsers.html_declaration_start
8316 * Cs(V("NoSoftLineBreakEndline") + parsers.any
8317 - parsers.nested_breaking_blank - parsers.html_declaration_end)^0
8318 * parsers.html_declaration_end
8319
8320 parsers.html_instruction_start = P("<?")
8321
8322 parsers.html_instruction_end = P("?>")
8323
8324 parsers.html_instruction
8325 = Cs(parsers.html_instruction_start
8326 * parsers.html_until_end(parsers.html_instruction_end))
8327
8328 parsers.html_inline_instruction = parsers.html_instruction_start
8329 * Cs(V("NoSoftLineBreakEndline")
8330 + parsers.any
8331 - parsers.nested_breaking_blank
8332 - parsers.html_instruction_end)^0
8333 * parsers.html_instruction_end
8334
8335 parsers.html_blankline = parsers.newline
8336 * parsers.optionalspace
8337 * parsers.newline
8338

```

```

8339 parsers.html_tag_start = parsers.less
8340
8341 parsers.html_tag_closing_start = parsers.less
8342 * parsers.slash
8343
8344 parsers.html_tag_end = parsers.html_spnl
8345 * parsers.more
8346
8347 parsers.html_empty_tag_end = parsers.html_spnl
8348 * parsers.slash
8349 * parsers.more
8350
8351 -- opening tags
8352 parsers.html_any_open_inline_tag = parsers.html_tag_start
8353 * parsers.keyword
8354 * parsers.html_attribute^0
8355 * parsers.html_tag_end
8356
8357 parsers.html_any_open_tag = parsers.html_tag_start
8358 * parsers.keyword
8359 * parsers.html_non_newline_attribute^0
8360 * parsers.html_tag_end
8361
8362 parsers.html_open_tag = parsers.html_tag_start
8363 * parsers.block_keyword
8364 * parsers.html_attribute^0
8365 * parsers.html_tag_end
8366
8367 parsers.html_open_special_tag = parsers.html_tag_start
8368 * parsers.special_block_keyword
8369 * parsers.html_attribute^0
8370 * parsers.html_tag_end
8371
8372 -- incomplete tags
8373 parsers.incomplete_tag_following = parsers.spacechar
8374 + parsers.more
8375 + parsers.slash * parsers.more
8376 + #(parsers.newline + parsers.eof)
8377
8378 parsers.incomplete_special_tag_following = parsers.spacechar
8379 + parsers.more
8380 + #(parsers.newline
8381 + parsers.eof)
8382
8383 parsers.html_incomplete_open_tag = parsers.html_tag_start
8384 * parsers.block_keyword
8385 * parsers.incomplete_tag_following

```

```

8386
8387 parsers.html_incomplete_open_special_tag
8388 = parsers.html_tag_start
8389 * parsers.special_block_keyword
8390 * parsers.incomplete_special_tag_following
8391
8392 parsers.html_incomplete_close_tag = parsers.html_tag_closing_start
8393 * parsers.block_keyword
8394 * parsers.incomplete_tag_following
8395
8396 parsers.html_incomplete_close_special_tag
8397 = parsers.html_tag_closing_start
8398 * parsers.special_block_keyword
8399 * parsers.incomplete_tag_following
8400
8401 -- closing tags
8402 parsers.html_close_tag = parsers.html_tag_closing_start
8403 * parsers.block_keyword
8404 * parsers.html_tag_end
8405
8406 parsers.html_any_close_tag = parsers.html_tag_closing_start
8407 * parsers.keyword
8408 * parsers.html_tag_end
8409
8410 parsers.html_close_special_tag = parsers.html_tag_closing_start
8411 * parsers.special_block_keyword
8412 * parsers.html_tag_end
8413
8414 -- empty tags
8415 parsers.html_any_empty_inline_tag = parsers.html_tag_start
8416 * parsers.keyword
8417 * parsers.html_attribute^0
8418 * parsers.html_empty_tag_end
8419
8420 parsers.html_any_empty_tag = parsers.html_tag_start
8421 * parsers.keyword
8422 * parsers.html_non_newline_attribute^0
8423 * parsers.optionalspace
8424 * parsers.slash
8425 * parsers.more
8426
8427 parsers.html_empty_tag = parsers.html_tag_start
8428 * parsers.block_keyword
8429 * parsers.html_attribute^0
8430 * parsers.html_empty_tag_end
8431
8432 parsers.html_empty_special_tag = parsers.html_tag_start

```

```

8433 * parsers.special_block_keyword
8434 * parsers.html_attribute^0
8435 * parsers.html_empty_tag_end
8436
8437 parsers.html_incomplete_blocks
8438 = parsers.html_incomplete_open_tag
8439 + parsers.html_incomplete_open_special_tag
8440 + parsers.html_incomplete_close_tag
8441
8442 -- parse special html blocks
8443 parsers.html_blankline_ending_special_block_opening
8444 = (parsers.html_close_special_tag
8445 + parsers.html_empty_special_tag)
8446 * #(parsers.optionalspace
8447 * (parsers.newline + parsers.eof))
8448
8449 parsers.html_blankline_ending_special_block
8450 = parsers.html_blankline_ending_special_block_opening
8451 * parsers.html_blankline_end_condition
8452
8453 parsers.html_special_block_opening
8454 = parsers.html_incomplete_open_special_tag
8455 - parsers.html_empty_special_tag
8456
8457 parsers.html_closing_special_block
8458 = parsers.html_special_block_opening
8459 * parsers.html_until_end(parsers.html_close_special_tag)
8460
8461 parsers.html_special_block
8462 = parsers.html_blankline_ending_special_block
8463 + parsers.html_closing_special_block
8464
8465 -- parse html blocks
8466 parsers.html_block_opening = parsers.html_incomplete_open_tag
8467 + parsers.html_incomplete_close_tag
8468
8469 parsers.html_block = parsers.html_block_opening
8470 * parsers.html_blankline_end_condition
8471
8472 -- parse any html blocks
8473 parsers.html_any_block_opening
8474 = (parsers.html_any_open_tag
8475 + parsers.html_any_close_tag
8476 + parsers.html_any_empty_tag)
8477 * #(parsers.optionalspace * (parsers.newline + parsers.eof))
8478
8479 parsers.html_any_block = parsers.html_any_block_opening

```



```

8480 * parsers.html_blankline_end_condition
8481
8482 parsers.html_inline_comment_full = parsers.html_comment_start
8483 * -P(">") * -P("->")
8484 * Cs((V("NoSoftLineBreakEndline")
8485 + parsers.any - P("--")
8486 - parsers.nested_breaking_blank
8487 - parsers.html_comment_end)^0)
8488 * parsers.html_comment_end
8489
8490 parsers.html_inline_tags = parsers.html_inline_comment_full
8491 + parsers.html_any_empty_inline_tag
8492 + parsers.html_inline_instruction
8493 + parsers.html_inline_cdatasection
8494 + parsers.html_inline_declaration
8495 + parsers.html_any_open_inline_tag
8496 + parsers.html_any_close_tag
8497

```

### 3.1.5.6 Parsers Used for Markdown Tags and Links

```

8498 parsers.urlchar = parsers.anyescaped
8499 - parsers.newline
8500 - parsers.more
8501
8502 parsers.auto_link_scheme_part = parsers.alphanumeric
8503 + parsers.plus
8504 + parsers.period
8505 + parsers.dash
8506
8507 parsers.auto_link_scheme = parsers.letter
8508 * parsers.auto_link_scheme_part
8509 * parsers.auto_link_scheme_part^-30
8510
8511 parsers.absolute_uri = parsers.auto_link_scheme * parsers.colon
8512 * (parsers.any - parsers.spacing
8513 - parsers.less - parsers.more)^0
8514
8515 parsers.printable_characters = S(" !#$%&'*/+=?^_`{|}~-")
8516
8517 parsers.email_address_local_part_char = parsers.alphanumeric
8518 + parsers.printable_characters
8519
8520 parsers.email_address_local_part
8521 = parsers.email_address_local_part_char^1
8522
8523 parsers.email_address_dns_label = parsers.alphanumeric

```

```

8524 * (parsers.alphanumeric
8525 + parsers.dash)^-62
8526 * B(parsers.alphanumeric)
8527
8528 parsers.email_address_domain = parsers.email_address_dns_label
8529 * (parsers.period
8530 * parsers.email_address_dns_label)^0
8531
8532 parsers.email_address = parsers.email_address_local_part
8533 * parsers.at
8534 * parsers.email_address_domain
8535
8536 parsers.auto_link_url = parsers.less
8537 * C(parsers.absolute_uri)
8538 * parsers.more
8539
8540 parsers.auto_link_email = parsers.less
8541 * C(parsers.email_address)
8542 * parsers.more
8543
8544 parsers.auto_link_relative_reference = parsers.less
8545 * C(parsers.urlchar^1)
8546 * parsers.more
8547
8548 parsers.autolink = parsers.auto_link_url
8549 + parsers.auto_link_email
8550
8551 -- content in balanced brackets, parentheses, or quotes:
8552 parsers.bracketed = P{ parsers.lbracket
8553 * ((parsers.backslash / "\"" * parsers.rbracket
8554 + parsers.any - (parsers.lbracket
8555 + parsers.rbracket
8556 + parsers.blankline^2)
8557) + V(1))^0
8558 * parsers.rbracket }
8559
8560 parsers.inparens = P{ parsers.lparent
8561 * ((parsers.anyescaped - (parsers.lparent
8562 + parsers.rparent
8563 + parsers.blankline^2)
8564) + V(1))^0
8565 * parsers.rparent }
8566
8567 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
8568 * ((parsers.anyescaped - (parsers.squote
8569 + parsers.blankline^2)
8570) + V(1))^0

```

```

8571 * parsers.squote }
8572
8573 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
8574 * ((parsers.anyescaped - (parsers.dquote
8575 + parsers.blankline^2)
8576) + V(1))^0
8577 * parsers.dquote }
8578
8579 parsers.link_text = parsers.lbracket
8580 * Cs((parsers.alphanumeric^1
8581 + parsers.bracketed
8582 + parsers.inticks
8583 + parsers.autolink
8584 + V("InlineHtml")
8585 + (parsers.backslash * parsers.backslash)
8586 + (parsers.backslash
8587 * (parsers.lbracket
8588 + parsers.rbracket)
8589 + V("NoSoftLineBreakSpace")
8590 + V("NoSoftLineBreakEndline")
8591 + (parsers.any
8592 - (parsers.newline
8593 + parsers.lbracket
8594 + parsers.rbracket
8595 + parsers.blankline^2))))^0)
8596 * parsers.rbracket
8597
8598 parsers.link_label_body = -(parsers.sp * parsers.rbracket)
8599 * #((parsers.any
8600 - parsers.rbracket)^-999
8601 * parsers.rbracket)
8602 * Cs((parsers.alphanumeric^1
8603 + parsers.inticks
8604 + parsers.autolink
8605 + V("InlineHtml")
8606 + (parsers.backslash * parsers.backslash)
8607 + (parsers.backslash
8608 * (parsers.lbracket
8609 + parsers.rbracket)
8610 + V("NoSoftLineBreakSpace")
8611 + V("NoSoftLineBreakEndline")
8612 + (parsers.any
8613 - (parsers.newline
8614 + parsers.lbracket
8615 + parsers.rbracket
8616 + parsers.blankline^2))))^1)
8617

```

```

8618 parsers.link_label = parsers.lbracket
8619 * parsers.link_label_body
8620 * parsers.rbracket
8621
8622 parsers.inparens_url = P{ parsers.lparent
8623 * ((parsers.anyescaped - (parsers.lparent
8624 + parsers.rparent
8625 + parsers.spacing)
8626) + V(1))^0
8627 * parsers.rparent }
8628
8629 -- url for markdown links, allowing nested brackets:
8630 parsers.url = parsers.less * Cs((parsers.anyescaped
8631 - parsers.newline
8632 - parsers.less
8633 - parsers.more)^0)
8634 * parsers.more
8635 + -parsers.less
8636 * Cs((parsers.inparens_url + (parsers.anyescaped
8637 - parsers.spacing
8638 - parsers.lparent
8639 - parsers.rparent))^1)
8640
8641 -- quoted text:
8642 parsers.title_s = parsers.squote
8643 * Cs((parsers.html_entities
8644 + V("NoSoftLineBreakSpace")
8645 + V("NoSoftLineBreakEndline")
8646 + (parsers.anyescaped
8647 - parsers.newline
8648 - parsers.squote
8649 - parsers.blankline^2))^0)
8650 * parsers.squote
8651
8652 parsers.title_d = parsers.dquote
8653 * Cs((parsers.html_entities
8654 + V("NoSoftLineBreakSpace")
8655 + V("NoSoftLineBreakEndline")
8656 + (parsers.anyescaped
8657 - parsers.newline
8658 - parsers.dquote
8659 - parsers.blankline^2))^0)
8660 * parsers.dquote
8661
8662 parsers.title_p = parsers.lparent
8663 * Cs((parsers.html_entities
8664 + V("NoSoftLineBreakSpace")

```

```

8665 + V("NoSoftLineBreakEndline")
8666 + (parsers.anyescaped
8667 - parsers.newline
8668 - parsers.lparent
8669 - parsers.rparent
8670 - parsers.blankline^2))^0)
8671 * parsers.rparent
8672
8673 parsers.title
8674 = parsers.title_d + parsers.title_s + parsers.title_p
8675
8676 parsers.optionaltitle
8677 = parsers.spnlc * parsers.title * parsers.spacechar^0 + Cc("")
8678

```

### 3.1.5.7 Helpers for Links and Link Reference Definitions

```

8679 -- parse a reference definition: [foo]: /bar "title"
8680 parsers.define_reference_parser = (parsers.check_trail / "")
8681 * parsers.link_label * parsers.colon
8682 * parsers.spnlc * parsers.url
8683 * (parsers.spnlc_sep * parsers.title
8684 * parsers.only_blank
8685 + Cc("") * parsers.only_blank)

```

### 3.1.5.8 Inline Elements

```

8686 parsers.Inline = V("Inline")
8687
8688 -- parse many p between starter and ender
8689 parsers.between = function(p, starter, ender)
8690 local ender2 = B(parsers.nonspacechar) * ender
8691 return (starter
8692 * #parsers.nonspacechar
8693 * Ct(p * (p - ender2)^0)
8694 * ender2)
8695 end
8696

```

### 3.1.5.9 Block Elements

```

8697 parsers.lineof = function(c)
8698 return (parsers.check_trail_no_rem
8699 * (P(c) * parsers.optionalspace)^3
8700 * (parsers.newline + parsers.eof))
8701 end
8702
8703 parsers.thematic_break_lines = parsers.lineof(parsers.asterisk)

```

```

8704 + parsers.lineof(parsers.dash)
8705 + parsers.lineof(parsers.underscore)

```

### 3.1.5.10 Headings

```

8706 -- parse Atx heading start and return level
8707 parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
8708 * -parsers.hash / length
8709
8710 -- parse setext header ending and return level
8711 parsers.heading_level
8712 = parsers.nonindentSPACE * parsers.equal^1
8713 * parsers.optionalspace * #parsers.newline * Cc(1)
8714 + parsers.nonindentSPACE * parsers.dash^1
8715 * parsers.optionalspace * #parsers.newline * Cc(2)
8716
8717 local function strip_atx_end(s)
8718 return s:gsub("%s+#+*%s*\n$", "")
8719 end
8720
8721 parsers.atx_heading = parsers.check_trail_no_rem
8722 * Cg(parsers.heading_start, "level")
8723 * (C(parsers.optionalspace
8724 * parsers.hash^0
8725 * parsers.optionalspace
8726 * parsers.newline)
8727 + parsers.spacechar^1
8728 * C(parsers.line))

```

### 3.1.6 Markdown Reader

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

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

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

```

8729 M.reader = {}
8730 function M.reader.new(writer, options)
8731 local self = {}

```

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

```
8732 self.writer = writer
8733 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.

```
8734 self.parsers = {}
8735 (function(parsers)
8736 setmetatable(self.parsers, {
8737 __index = function (_, key)
8738 return parsers[key]
8739 end
8740 })
8741 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.

```
8742 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.

```
8743 function self.normalize_tag(tag)
8744 tag = util.ropetostring(tag)
8745 tag = tag:gsub("[\n\r\t]+", " ")
8746 tag = tag:gsub("^ ", ""):gsub(" $", "")
8747 tag = uni_algos.case.casefold(tag, true, false)
8748 return tag
8749 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.

```
8750 local function iterlines(s, f)
8751 local rope = lpeg.match(Ct((parsers.line / f)^1), s)
8752 return util.ropetostring(rope)
8753 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.

```
8754 if options.preserveTabs then
8755 self.expandtabs = function(s) return s end
8756 else
8757 self.expandtabs = function(s)
8758 if s:find("\t") then
```

```

8759 return iterlines(s, util.expand_tabs_in_line)
8760 else
8761 return s
8762 end
8763 end
8764 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.

```

8765 self.parser_functions = {}
8766 self.create_parser = function(name, grammar, toplevel)
8767 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.

```

8768 if toplevel and options.stripIndent then
8769 local min_prefix_length, min_prefix = nil, ''
8770 str = iterlines(str, function(line)
8771 if lpeg.match(parsers.nonemptyline, line) == nil then
8772 return line
8773 end
8774 line = util.expand_tabs_in_line(line)
8775 local prefix = lpeg.match(C(parsers.optionalspace), line)
8776 local prefix_length = #prefix
8777 local is_shorter = min_prefix_length == nil
8778 if not is_shorter then
8779 is_shorter = prefix_length < min_prefix_length
8780 end
8781 if is_shorter then
8782 min_prefix_length, min_prefix = prefix_length, prefix
8783 end
8784 return line
8785 end)
8786 str = str:gsub('^' .. min_prefix, '')
8787 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.

```

8788 if toplevel and (options.texComments or options.hybrid) then

```



```

8789 str = lpeg.match(Ct(parsers.commented_line^1), str)
8790 str = util.ropo_to_string(str)
8791 end
8792 local res = lpeg.match(grammar(), str)
8793 if res == nil then
8794 return writer.error(
8795 format("Parser `%s` failed to process the input text.", name),
8796 format("Here are the first 20 characters of the remaining "
8797 .. "unprocessed text: `%s`.", str:sub(1,20))
8798)
8799 else
8800 return res
8801 end
8802 end
8803 end
8804
8805 self.create_parser("parse_blocks",
8806 function()
8807 return parsers.blocks
8808 end, true)
8809
8810 self.create_parser("parse_blocks_nested",
8811 function()
8812 return parsers.blocks_nested
8813 end, false)
8814
8815 self.create_parser("parse_inlines",
8816 function()
8817 return parsers.inlines
8818 end, false)
8819
8820 self.create_parser("parse_inlines_no_inline_note",
8821 function()
8822 return parsers.inlines_no_inline_note
8823 end, false)
8824
8825 self.create_parser("parse_inlines_no_html",
8826 function()
8827 return parsers.inlines_no_html
8828 end, false)
8829
8830 self.create_parser("parse_inlines_nbsp",
8831 function()
8832 return parsers.inlines_nbsp
8833 end, false)
8834 self.create_parser("parse_inlines_no_link_or_emphasis",
8835 function()

```

```
8836 return parsers.inlines_no_link_or_emphasis
8837 end, false)
```

### 3.1.6.3 Parsers Used for Indentation (local)

The following patterns represent basic building blocks of indented content.

```
8838 parsers.minimally_indented_blankline
8839 = parsers.check_minimal_indent * (parsers.blankline / "")
8840
8841 parsers.minimally_indented_block
8842 = parsers.check_minimal_indent * V("Block")
8843
8844 parsers.minimally_indented_block_or_paragraph
8845 = parsers.check_minimal_indent * V("BlockOrParagraph")
8846
8847 parsers.minimally_indented_paragraph
8848 = parsers.check_minimal_indent * V("Paragraph")
8849
8850 parsers.minimally_indented_plain
8851 = parsers.check_minimal_indent * V("Plain")
8852
8853 parsers.minimally_indented_par_or_plain
8854 = parsers.minimally_indented_paragraph
8855 + parsers.minimally_indented_plain
8856
8857 parsers.minimally_indented_par_or_plain_no_blank
8858 = parsers.minimally_indented_par_or_plain
8859 - parsers.minimally_indented_blankline
8860
8861 parsers.minimally_indented_ref
8862 = parsers.check_minimal_indent * V("Reference")
8863
8864 parsers.minimally_indented_blank
8865 = parsers.check_minimal_indent * V("Blank")
8866
8867 parsers.conditionally_indented_blankline
8868 = parsers.check_minimal_blank_indent * (parsers.blankline / "")
8869
8870 parsers.minimally_indented_ref_or_block
8871 = parsers.minimally_indented_ref
8872 + parsers.minimally_indented_block
8873 - parsers.minimally_indented_blankline
8874
8875 parsers.minimally_indented_ref_or_block_or_par
8876 = parsers.minimally_indented_ref
8877 + parsers.minimally_indented_block_or_paragraph
8878 - parsers.minimally_indented_blankline
```

8879

The following pattern parses the properly indented content that follows the initial container start.

```
8880
8881 function parsers.separator_loop(separated_block, paragraph,
8882 block_separator, paragraph_separator)
8883 return separated_block
8884 + block_separator
8885 * paragraph
8886 * separated_block
8887 + paragraph_separator
8888 * paragraph
8889 end
8890
8891 function parsers.create_loop_body_pair(separated_block, paragraph,
8892 block_separator,
8893 paragraph_separator)
8894 return {
8895 block = parsers.separator_loop(separated_block, paragraph,
8896 block_separator, block_separator),
8897 par = parsers.separator_loop(separated_block, paragraph,
8898 block_separator, paragraph_separator)
8899 }
8900 end
8901
8902 parsers.block_sep_group = function(blank)
8903 return blank^0 * parsers.eof
8904 + (blank^2 / writer.paragraphsep
8905 + blank^0 / writer.interblocksep
8906)
8907 end
8908
8909 parsers.par_sep_group = function(blank)
8910 return blank^0 * parsers.eof
8911 + blank^0 / writer.paragraphsep
8912 end
8913
8914 parsers.sep_group_no_output = function(blank)
8915 return blank^0 * parsers.eof
8916 + blank^0
8917 end
8918
8919 parsers.content_blank = parsers.minimally_indented_blankline
8920
8921 parsers.ref_or_block_separated
8922 = parsers.sep_group_no_output(parsers.content_blank)
```

```

8923 * (parsers.minimally_indented_ref
8924 - parsers.content_blank)
8925 + parsers.block_sep_group(parsers.content_blank)
8926 * (parsers.minimally_indented_block
8927 - parsers.content_blank)
8928
8929 parsers.loop_body_pair =
8930 parsers.create_loop_body_pair(
8931 parsers.ref_or_block_separated,
8932 parsers.minimally_indented_par_or_plain_no_blank,
8933 parsers.block_sep_group(parsers.content_blank),
8934 parsers.par_sep_group(parsers.content_blank))
8935
8936 parsers.content_loop = (V("Block")
8937 * parsers.loop_body_pair.block~0
8938 + (V("Paragraph") + V("Plain"))
8939 * parsers.ref_or_block_separated
8940 * parsers.loop_body_pair.block~0
8941 + (V("Paragraph") + V("Plain"))
8942 * parsers.loop_body_pair.par~0
8943 * parsers.content_blank~0
8944
8945 parsers.indented_content = function()
8946 return Ct((V("Reference") + (parsers.blankline / ""))
8947 * parsers.content_blank~0
8948 * parsers.check_minimal_indent
8949 * parsers.content_loop
8950 + (V("Reference") + (parsers.blankline / ""))
8951 * parsers.content_blank~0
8952 + parsers.content_loop)
8953 end
8954
8955 parsers.add_indent = function(pattern, name, breakable)
8956 return Cg(Cmt(Cb("indent_info")
8957 * Ct(pattern)
8958 * (#parsers.linechar -- check if starter is blank
8959 * Cc(false) + Cc(true))
8960 * Cc(name)
8961 * Cc(breakable),
8962 process_starter_indent), "indent_info")
8963 end
8964

```

### 3.1.6.4 Parsers Used for Markdown Lists (local)

```

8965 if options.hashEnumerators then
8966 parsers.dig = parsers.digit + parsers.hash

```

```

8967 else
8968 parsers.dig = parsers.digit
8969 end
8970
8971 parsers.enumerator = function(delimiter_type, interrupting)
8972 local delimiter_range
8973 local allowed_end
8974 if interrupting then
8975 delimiter_range = P("1")
8976 allowed_end = C(parsers.spacechar^1) * #parsers.linechar
8977 else
8978 delimiter_range = parsers.dig * parsers.dig^-8
8979 allowed_end = C(parsers.spacechar^1)
8980 + #(parsers.newline + parsers.eof)
8981 end
8982
8983 return parsers.check_trail
8984 * Ct(C(delimiter_range) * C(delimiter_type))
8985 * allowed_end
8986 end
8987
8988 parsers.starter = parsers.bullet(parsers.dash)
8989 + parsers.bullet(parsers.asterisk)
8990 + parsers.bullet(parsers.plus)
8991 + parsers.enumerator(parsers.period)
8992 + parsers.enumerator(parsers.rparent)
8993

```

### 3.1.6.5 Parsers Used for Blockquotes (local)

```

8994 parsers.blockquote_start
8995 = parsers.check_trail
8996 * C(parsers.more)
8997 * C(parsers.spacechar^0)
8998
8999 parsers.blockquote_body
9000 = parsers.add_indent(parsers.blockquote_start, "bq", true)
9001 * parsers.indented_content()
9002 * remove_indent("bq")
9003
9004 if not options.breakableBlockquotes then
9005 parsers.blockquote_body
9006 = parsers.add_indent(parsers.blockquote_start, "bq", false)
9007 * parsers.indented_content()
9008 * remove_indent("bq")
9009 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.

```
9010 local function parse_content_part(content_part)
9011 local rope = util.rope_to_string(content_part)
9012 local parsed
9013 = self.parser_functions.parse_inlines_no_link_or_emphasis(rope)
9014 parsed.indent_info = nil
9015 return parsed
9016 end
9017
```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
9018 local collect_emphasis_content =
9019 function(t, opening_index, closing_index)
9020 local content = {}
9021
9022 local content_part = {}
9023 for i = opening_index, closing_index do
9024 local value = t[i]
9025
9026 if value.rendered ~= nil then
9027 content[#content + 1] = parse_content_part(content_part)
9028 content_part = {}
9029 content[#content + 1] = value.rendered
9030 value.rendered = nil
9031 else
9032 if value.type == "delimiter"
9033 and value.element == "emphasis" then
9034 if value.is_active then
9035 content_part[#content_part + 1]
9036 = string.rep(value.character, value.current_count)
9037 end
9038 else
9039 content_part[#content_part + 1] = value.content
9040 end
9041 value.content = ''
9042 value.is_active = false
9043 end
9044 end
9045
9046 if next(content_part) ~= nil then
9047 content[#content + 1] = parse_content_part(content_part)
9048 end
9049
9050 return content
9051
```

```
9051 end
9052
```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as emphasis.

```
9053 local function fill_emph(t, opening_index, closing_index)
9054 local content
9055 = collect_emphasis_content(t, opening_index + 1,
9056 closing_index - 1)
9057 t[opening_index + 1].is_active = true
9058 t[opening_index + 1].rendered = writer.emphasis(content)
9059 end
9060
```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as strong emphasis.

```
9061 local function fill_strong(t, opening_index, closing_index)
9062 local content
9063 = collect_emphasis_content(t, opening_index + 1,
9064 closing_index - 1)
9065 t[opening_index + 1].is_active = true
9066 t[opening_index + 1].rendered = writer.strong(content)
9067 end
9068
```

Check whether the opening delimiter `opening_delimiter` and closing delimiter `closing_delimiter` break rule three together.

```
9069 local function breaks_three_rule(opening_delimiter, closing_delimiter)
9070 return (opening_delimiter.is_closing
9071 or closing_delimiter.is_opening)
9072 and ((opening_delimiter.original_count
9073 + closing_delimiter.original_count) % 3 == 0)
9074 and (opening_delimiter.original_count % 3 ~= 0
9075 or closing_delimiter.original_count % 3 ~= 0)
9076 end
9077
```

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`.

```
9078 local find_emphasis_opener = function(t, bottom_index, latest_index,
9079 character, closing_delimiter)
9080 for i = latest_index, bottom_index, -1 do
9081 local value = t[i]
9082 if value.is_active and
9083 value.is_opening and
9084 value.type == "delimiter" and
9085 value.element == "emphasis" and
```

```

9086 (value.character == character) and
9087 (value.current_count > 0) then
9088 if not breaks_three_rule(value, closing_delimiter) then
9089 return i
9090 end
9091 end
9092 end
9093 end
9094

```

Iterate over the delimiters in the delimiter table `t`, producing emphasis or strong emphasis macros.

```

9095 local function process_emphasis(t, opening_index, closing_index)
9096 for i = opening_index, closing_index do
9097 local value = t[i]
9098 if value.type == "delimiter" and value.element == "emphasis" then
9099 local delimiter_length = string.len(value.content)
9100 value.character = string.sub(value.content, 1, 1)
9101 value.current_count = delimiter_length
9102 value.original_count = delimiter_length
9103 end
9104 end
9105
9106 local openers_bottom = {
9107 ['*'] = {
9108 [true] = {opening_index, opening_index, opening_index},
9109 [false] = {opening_index, opening_index, opening_index}
9110 },
9111 ['_'] = {
9112 [true] = {opening_index, opening_index, opening_index},
9113 [false] = {opening_index, opening_index, opening_index}
9114 }
9115 }
9116
9117 local current_position = opening_index
9118 local max_position = closing_index
9119
9120 while current_position <= max_position do
9121 local value = t[current_position]
9122
9123 if value.type ~= "delimiter" or
9124 value.element ~= "emphasis" or
9125 not value.is_active or
9126 not value.is_closing or
9127 (value.current_count <= 0) then
9128 current_position = current_position + 1
9129 goto continue

```



```

9130 end
9131
9132 local character = value.character
9133 local is_opening = value.is_opening
9134 local closing_length_modulo_three = value.original_count % 3
9135
9136 local current_openers_bottom
9137 = openers_bottom[character][is_opening]
9138 [closing_length_modulo_three + 1]
9139
9140 local opener_position
9141 = find_emphasis_opener(t, current_openers_bottom,
9142 current_position - 1, character, value)
9143
9144 if (opener_position == nil) then
9145 openers_bottom[character][is_opening]
9146 [closing_length_modulo_three + 1]
9147 = current_position
9148 current_position = current_position + 1
9149 goto continue
9150 end
9151
9152 local opening_delimiter = t[opener_position]
9153
9154 local current_opening_count = opening_delimiter.current_count
9155 local current_closing_count = t[current_position].current_count
9156
9157 if (current_opening_count >= 2)
9158 and (current_closing_count >= 2) then
9159 opening_delimiter.current_count = current_opening_count - 2
9160 t[current_position].current_count = current_closing_count - 2
9161 fill_strong(t, opener_position, current_position)
9162 else
9163 opening_delimiter.current_count = current_opening_count - 1
9164 t[current_position].current_count = current_closing_count - 1
9165 fill_emph(t, opener_position, current_position)
9166 end
9167
9168 ::continue::
9169 end
9170 end
9171
9172 parsers.delimiter_run = function(character)
9173 return (B(parsers.backslash * character) + -B(character))
9174 * character^1
9175 * -#character
9176 end

```

```

9177
9178 parsers.left_flanking_delimiter_run = function(character)
9179 return (B(parsers.any)
9180 * (parsers.unicode.preceding_punctuation
9181 + parsers.unicode.preceding_whitespace)
9182 + -B(parsers.any))
9183 * parsers.delimiter_run(character)
9184 * parsers.unicode.following_punctuation
9185 + parsers.delimiter_run(character)
9186 * -(parsers.unicode.following_punctuation
9187 + parsers.unicode.following_whitespace
9188 + parsers.eof)
9189 end
9190
9191 parsers.right_flanking_delimiter_run = function(character)
9192 return parsers.unicode.preceding_punctuation
9193 * parsers.delimiter_run(character)
9194 * (parsers.unicode.following_punctuation
9195 + parsers.unicode.following_whitespace
9196 + parsers.eof)
9197 + (B(parsers.any)
9198 * -(parsers.unicode.preceding_punctuation
9199 + parsers.unicode.preceding_whitespace))
9200 * parsers.delimiter_run(character)
9201 end
9202
9203 if options.underscores then
9204 parsers.emph_start
9205 = parsers.left_flanking_delimiter_run(parsers.asterisk)
9206 + (-#parsers.right_flanking_delimiter_run(parsers.underscore)
9207 + (parsers.unicode.preceding_punctuation
9208 * #parsers.right_flanking_delimiter_run(parsers.underscore)))
9209 * parsers.left_flanking_delimiter_run(parsers.underscore)
9210
9211 parsers.emph_end
9212 = parsers.right_flanking_delimiter_run(parsers.asterisk)
9213 + (-#parsers.left_flanking_delimiter_run(parsers.underscore)
9214 + #(parsers.left_flanking_delimiter_run(parsers.underscore)
9215 * parsers.unicode.following_punctuation))
9216 * parsers.right_flanking_delimiter_run(parsers.underscore)
9217 else
9218 parsers.emph_start
9219 = parsers.left_flanking_delimiter_run(parsers.asterisk)
9220
9221 parsers.emph_end
9222 = parsers.right_flanking_delimiter_run(parsers.asterisk)
9223 end

```

```

9224
9225 parsers.emph_capturing_open_and_close
9226 = #parsers.emph_start * #parsers.emph_end
9227 * Ct(Cg(Cc("delimiter"), "type")
9228 * Cg(Cc("emphasis"), "element")
9229 * Cg(C(parsers.emph_start), "content")
9230 * Cg(Cc(true), "is_opening")
9231 * Cg(Cc(true), "is_closing"))
9232
9233 parsers.emph_capturing_open = Ct(Cg(Cc("delimiter"), "type")
9234 * Cg(Cc("emphasis"), "element")
9235 * Cg(C(parsers.emph_start), "content")
9236 * Cg(Cc(true), "is_opening")
9237 * Cg(Cc(false), "is_closing"))
9238
9239 parsers.emph_capturing_close = Ct(Cg(Cc("delimiter"), "type")
9240 * Cg(Cc("emphasis"), "element")
9241 * Cg(C(parsers.emph_end), "content")
9242 * Cg(Cc(false), "is_opening")
9243 * Cg(Cc(true), "is_closing"))
9244
9245 parsers.emph_open_or_close = parsers.emph_capturing_open_and_close
9246 + parsers.emph_capturing_open
9247 + parsers.emph_capturing_close
9248
9249 parsers.emph_open = parsers.emph_capturing_open_and_close
9250 + parsers.emph_capturing_open
9251
9252 parsers.emph_close = parsers.emph_capturing_open_and_close
9253 + parsers.emph_capturing_close
9254

```

### 3.1.6.7 Helpers for Links and Link Reference Definitions (local)

```

9255 -- List of references defined in the document
9256 local references
9257
9258 -- List of note references defined in the document
9259 parsers.rawnotes = {}
9260

```

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.

```

9261 function self.register_link(_, tag, url, title,
9262 attributes)
9263 local normalized_tag = self.normalize_tag(tag)
9264 if references[normalized_tag] == nil then

```

```

9265 references[normalized_tag] = {
9266 url = url,
9267 title = title,
9268 attributes = attributes
9269 }
9270 end
9271 return ""
9272 end
9273

```

The `reader->lookup_reference` method looks up a reference with link label `tag`.

```

9274 function self.lookup_reference(tag)
9275 return references[self.normalize_tag(tag)]
9276 end
9277

```

The `reader->lookup_note_reference` method looks up a note reference with label `tag`.

```

9278 function self.lookup_note_reference(tag)
9279 return parsers.rawnotes[self.normalize_tag(tag)]
9280 end
9281
9282 parsers.title_s_direct_ref = parsers.squote
9283 * Cs((parsers.html_entities
9284 + (parsers.anyescaped
9285 - parsers.squote
9286 - parsers.blankline^2))^0)
9287 * parsers.squote
9288
9289 parsers.title_d_direct_ref = parsers.dquote
9290 * Cs((parsers.html_entities
9291 + (parsers.anyescaped
9292 - parsers.dquote
9293 - parsers.blankline^2))^0)
9294 * parsers.dquote
9295
9296 parsers.title_p_direct_ref = parsers.lparent
9297 * Cs((parsers.html_entities
9298 + (parsers.anyescaped
9299 - parsers.lparent
9300 - parsers.rparent
9301 - parsers.blankline^2))^0)
9302 * parsers.rparent
9303
9304 parsers.title_direct_ref = parsers.title_s_direct_ref
9305 + parsers.title_d_direct_ref
9306 + parsers.title_p_direct_ref
9307

```

```

9308 parsers.inline_direct_ref_inside = parsers.lparent * parsers.spnl
9309 * Cg(parsers.url + Cc(""), "url")
9310 * parsers.spnl
9311 * Cg(parsers.title_direct_ref
9312 + Cc(""), "title")
9313 * parsers.spnl * parsers.rparent
9314
9315 parsers.inline_direct_ref = parsers.lparent * parsers.spnlc
9316 * Cg(parsers.url + Cc(""), "url")
9317 * parsers.spnlc
9318 * Cg(parsers.title + Cc(""), "title")
9319 * parsers.spnlc * parsers.rparent
9320
9321 parsers.empty_link = parsers.lbracket
9322 * parsers.rbracket
9323
9324 parsers.inline_link = parsers.link_text
9325 * parsers.inline_direct_ref
9326
9327 parsers.full_link = parsers.link_text
9328 * parsers.link_label
9329
9330 parsers.shortcut_link = parsers.link_label
9331 * -(parsers.empty_link + parsers.link_label)
9332
9333 parsers.collapsed_link = parsers.link_label
9334 * parsers.empty_link
9335
9336 parsers.image_opening = #(parsers.exclamation * parsers.inline_link)
9337 * Cg(Cc("inline"), "link_type")
9338 + #(parsers.exclamation * parsers.full_link)
9339 * Cg(Cc("full"), "link_type")
9340 + #(parsers.exclamation
9341 * parsers.collapsed_link)
9342 * Cg(Cc("collapsed"), "link_type")
9343 + #(parsers.exclamation * parsers.shortcut_link)
9344 * Cg(Cc("shortcut"), "link_type")
9345 + #(parsers.exclamation * parsers.empty_link)
9346 * Cg(Cc("empty"), "link_type")
9347
9348 parsers.link_opening = #parsers.inline_link
9349 * Cg(Cc("inline"), "link_type")
9350 + #parsers.full_link
9351 * Cg(Cc("full"), "link_type")
9352 + #parsers.collapsed_link
9353 * Cg(Cc("collapsed"), "link_type")
9354 + #parsers.shortcut_link

```

```

9355 * Cg(Cc("shortcut"), "link_type")
9356 + #parsers.empty_link
9357 * Cg(Cc("empty_link"), "link_type")
9358 + #parsers.link_text
9359 * Cg(Cc("link_text"), "link_type")
9360
9361 parsers.note_opening = #(parsers.circumflex * parsers.link_text)
9362 * Cg(Cc("note_inline"), "link_type")
9363
9364 parsers.raw_note_opening = #(parsers.lbracket
9365 * parsers.circumflex
9366 * parsers.link_label_body
9367 * parsers.rbracket)
9368 * Cg(Cc("raw_note"), "link_type")
9369
9370 local inline_note_element = Cg(Cc("note"), "element")
9371 * parsers.note_opening
9372 * Cg(parsers.circumflex
9373 * parsers.lbracket, "content")
9374
9375 local image_element = Cg(Cc("image"), "element")
9376 * parsers.image_opening
9377 * Cg(parsers.exclamation
9378 * parsers.lbracket, "content")
9379
9380 local note_element = Cg(Cc("note"), "element")
9381 * parsers.raw_note_opening
9382 * Cg(parsers.lbracket
9383 * parsers.circumflex, "content")
9384
9385 local link_element = Cg(Cc("link"), "element")
9386 * parsers.link_opening
9387 * Cg(parsers.lbracket, "content")
9388
9389 local opening_elements = parsers.fail
9390
9391 if options.inlineNotes then
9392 opening_elements = opening_elements + inline_note_element
9393 end
9394
9395 opening_elements = opening_elements + image_element
9396
9397 if options.notes then
9398 opening_elements = opening_elements + note_element
9399 end
9400
9401 opening_elements = opening_elements + link_element

```

```

9402
9403 parsers.link_image_opening = Ct(Cg(Cc("delimiter"), "type")
9404 * Cg(Cc(true), "is_opening")
9405 * Cg(Cc(false), "is_closing")
9406 * opening_elements)
9407
9408 parsers.link_image_closing = Ct(Cg(Cc("delimiter"), "type")
9409 * Cg(Cc("link"), "element")
9410 * Cg(Cc(false), "is_opening")
9411 * Cg(Cc(true), "is_closing")
9412 * (Cg(Cc(true), "is_direct")
9413 * Cg(parsers.rbracket
9414 * #parsers.inline_direct_ref,
9415 "content")
9416 + Cg(Cc(false), "is_direct")
9417 * Cg(parsers.rbracket, "content")))
9418
9419 parsers.link_image_open_or_close = parsers.link_image_opening
9420 + parsers.link_image_closing
9421
9422 if options.html then
9423 parsers.link_emph_precedence = parsers.inticks
9424 + parsers.autolink
9425 + parsers.html_inline_tags
9426 else
9427 parsers.link_emph_precedence = parsers.inticks
9428 + parsers.autolink
9429 end
9430
9431 parsers.link_and_emph_endline = parsers.newline
9432 * ((parsers.check_minimal_indent
9433 * -V("EndlineExceptions")
9434 + parsers.check_optional_indent
9435 * -V("EndlineExceptions")
9436 * -V("ListStarter")) / "")
9437 * parsers.spacechar^0 / "\n"
9438
9439 parsers.link_and_emph_content
9440 = Ct(Cg(Cc("content"), "type")
9441 * Cg(Cs((parsers.link_emph_precedence
9442 + parsers.backslash * parsers.linechar
9443 + parsers.link_and_emph_endline
9444 + (parsers.linechar
9445 - parsers.blankline^2
9446 - parsers.link_image_open_or_close
9447 - parsers.emph_open_or_close))^0), "content"))
9448

```

```

9449 parsers.link_and_emph_table
9450 = (parsers.link_image_opening + parsers.emph_open)
9451 * parsers.link_and_emph_content
9452 * ((parsers.link_image_open_or_close + parsers.emph_open_or_close)
9453 * parsers.link_and_emph_content)^1
9454

```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

9455 local function collect_link_content(t, opening_index, closing_index)
9456 local content = {}
9457 for i = opening_index, closing_index do
9458 content[#content + 1] = t[i].content
9459 end
9460 return util.rope_to_string(content)
9461 end
9462

```

Look for the closest potential link opener in the delimiter table `t` in the range from `bottom_index` to `latest_index`.

```

9463 local function find_link_opener(t, bottom_index, latest_index)
9464 for i = latest_index, bottom_index, -1 do
9465 local value = t[i]
9466 if value.type == "delimiter" and
9467 value.is_opening and
9468 (value.element == "link"
9469 or value.element == "image"
9470 or value.element == "note")
9471 and not value.removed then
9472 if value.is_active then
9473 return i
9474 end
9475 value.removed = true
9476 return nil
9477 end
9478 end
9479 end
9480

```

Find the position of a delimiter that closes a full link after an an index `latest_index` in the delimiter table `t`.

```

9481 local function find_next_link_closing_index(t, latest_index)
9482 for i = latest_index, #t do
9483 local value = t[i]
9484 if value.is_closing and
9485 value.element == "link" and
9486 not value.removed then
9487 return i

```



```

9488 end
9489 end
9490 end
9491

```

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.

```

9492 local function disable_previous_link_openers(t, opening_index)
9493 if t[opening_index].element == "image" then
9494 return
9495 end
9496
9497 for i = opening_index, 1, -1 do
9498 local value = t[i]
9499 if value.is_active and
9500 value.type == "delimiter" and
9501 value.is_opening and
9502 value.element == "link" then
9503 value.is_active = false
9504 end
9505 end
9506 end
9507

```

Disable the delimiters between the `opening_index` and `closing_index` in the delimiter table `t` by marking them inactive with the `is_active` property.

```

9508 local function disable_range(t, opening_index, closing_index)
9509 for i = opening_index, closing_index do
9510 local value = t[i]
9511 if value.is_active then
9512 value.is_active = false
9513 if value.type == "delimiter" then
9514 value.removed = true
9515 end
9516 end
9517 end
9518 end
9519

```

Clear the parsed content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

9520 local delete_parsed_content_in_range =
9521 function(t, opening_index, closing_index)
9522 for i = opening_index, closing_index do
9523 t[i].rendered = nil
9524 end
9525 end
9526

```

Clear the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
9527 local function empty_content_in_range(t, opening_index, closing_index)
9528 for i = opening_index, closing_index do
9529 t[i].content = ''
9530 end
9531 end
9532
```

Join the attributes from the link reference definition `reference_attributes` with the link's own attributes `own_attributes`.

```
9533 local function join_attributes(reference_attributes, own_attributes)
9534 local merged_attributes = {}
9535 for _, attribute in ipairs(reference_attributes or {}) do
9536 table.insert(merged_attributes, attribute)
9537 end
9538 for _, attribute in ipairs(own_attributes or {}) do
9539 table.insert(merged_attributes, attribute)
9540 end
9541 if next(merged_attributes) == nil then
9542 merged_attributes = nil
9543 end
9544 return merged_attributes
9545 end
9546
```

Parse content between two delimiters in the delimiter table `t`. Produce the respective link and image macros.

```
9547 local render_link_or_image =
9548 function(t, opening_index, closing_index, content_end_index,
9549 reference)
9550 process_emphasis(t, opening_index, content_end_index)
9551 local mapped = collect_emphasis_content(t, opening_index + 1,
9552 content_end_index - 1)
9553
9554 local rendered = {}
9555 if (t[opening_index].element == "link") then
9556 rendered = writer.link(mapped, reference.url,
9557 reference.title, reference.attributes)
9558 end
9559
9560 if (t[opening_index].element == "image") then
9561 rendered = writer.image(mapped, reference.url, reference.title,
9562 reference.attributes)
9563 end
9564
9565 if (t[opening_index].element == "note") then
```

```

9566 if (t[opening_index].link_type == "note_inline") then
9567 rendered = writer.note(mapped)
9568 end
9569 if (t[opening_index].link_type == "raw_note") then
9570 rendered = writer.note(reference)
9571 end
9572 end
9573
9574 t[opening_index].rendered = rendered
9575 delete_parsed_content_in_range(t, opening_index + 1,
9576 closing_index)
9577 empty_content_in_range(t, opening_index, closing_index)
9578 disable_previous_link_openers(t, opening_index)
9579 disable_range(t, opening_index, closing_index)
9580 end
9581

```

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.

```

9582 local resolve_inline_following_content =
9583 function(t, closing_index, match_reference, match_link_attributes)
9584 local content = ""
9585 for i = closing_index + 1, #t do
9586 content = content .. t[i].content
9587 end
9588
9589 local matching_content = parsers.succeed
9590
9591 if match_reference then
9592 matching_content = matching_content
9593 * parsers.inline_direct_ref_inside
9594 end
9595
9596 if match_link_attributes then
9597 matching_content = matching_content
9598 * Cg(Ct(parsers.attributes^-1), "attributes")
9599 end
9600
9601 local matched = lpeg.match(Ct(matching_content
9602 * Cg(Cp(), "end_position")), content)
9603
9604 local matched_count = matched.end_position - 1
9605 for i = closing_index + 1, #t do
9606 local value = t[i]
9607
9608 local chars_left = matched_count

```

```

9609 matched_count = matched_count - #value.content
9610
9611 if matched_count <= 0 then
9612 value.content = value.content:sub(chars_left + 1)
9613 break
9614 end
9615
9616 value.content = ''
9617 value.is_active = false
9618 end
9619
9620 local attributes = matched.attributes
9621 if attributes == nil or next(attributes) == nil then
9622 attributes = nil
9623 end
9624
9625 return {
9626 url = matched.url or "",
9627 title = matched.title or "",
9628 attributes = attributes
9629 }
9630 end
9631

```

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.

```

9632 local function resolve_inline_link(t, opening_index, closing_index)
9633 local inline_content
9634 = resolve_inline_following_content(t, closing_index, true,
9635 t.match_link_attributes)
9636 render_link_or_image(t, opening_index, closing_index,
9637 closing_index, inline_content)
9638 end
9639

```

Resolve an inline note `^[a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`.

```

9640 local resolve_note_inline_link =
9641 function(t, opening_index, closing_index)
9642 local inline_content
9643 = resolve_inline_following_content(t, closing_index,
9644 false, false)
9645 render_link_or_image(t, opening_index, closing_index,
9646 closing_index, inline_content)
9647 end
9648

```

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.

```
9649 local function resolve_shortcut_link(t, opening_index, closing_index)
9650 local content
9651 = collect_link_content(t, opening_index + 1, closing_index - 1)
9652 local r = self.lookup_reference(content)
9653
9654 if r then
9655 local inline_content
9656 = resolve_inline_following_content(t, closing_index, false,
9657 t.match_link_attributes)
9658 r.attributes
9659 = join_attributes(r.attributes, inline_content.attributes)
9660 render_link_or_image(t, opening_index, closing_index,
9661 closing_index, r)
9662 end
9663 end
9664
```

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.

```
9665 local function resolve_raw_note_link(t, opening_index, closing_index)
9666 local content
9667 = collect_link_content(t, opening_index + 1, closing_index - 1)
9668 local r = self.lookup_note_reference(content)
9669
9670 if r then
9671 local parsed_ref = self.parser_functions.parse_blocks_nested(r)
9672 render_link_or_image(t, opening_index, closing_index,
9673 closing_index, parsed_ref)
9674 end
9675 end
9676
```

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.

```
9677 local function resolve_full_link(t, opening_index, closing_index)
9678 local next_link_closing_index
9679 = find_next_link_closing_index(t, closing_index + 4)
9680 local next_link_content
9681 = collect_link_content(t, closing_index + 3,
9682 next_link_closing_index - 1)
9683 local r = self.lookup_reference(next_link_content)
9684
9685 if r then
9686 local inline_content
```

```

9687 = resolve_inline_following_content(t, next_link_closing_index,
9688 false,
9689 t.match_link_attributes)
9690 r.attributes
9691 = join_attributes(r.attributes, inline_content.attributes)
9692 render_link_or_image(t, opening_index, next_link_closing_index,
9693 closing_index, r)
9694 end
9695 end
9696

```

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.

```

9697 local function resolve_collapsed_link(t, opening_index, closing_index)
9698 local next_link_closing_index
9699 = find_next_link_closing_index(t, closing_index + 4)
9700 local content
9701 = collect_link_content(t, opening_index + 1, closing_index - 1)
9702 local r = self.lookup_reference(content)
9703
9704 if r then
9705 local inline_content
9706 = resolve_inline_following_content(t, closing_index, false,
9707 t.match_link_attributes)
9708 r.attributes
9709 = join_attributes(r.attributes, inline_content.attributes)
9710 render_link_or_image(t, opening_index, next_link_closing_index,
9711 closing_index, r)
9712 end
9713 end
9714

```

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.

```

9715 local function process_links_and_emphasis(t)
9716 for _,value in ipairs(t) do
9717 value.is_active = true
9718 end
9719
9720 for i,value in ipairs(t) do
9721 if not value.is_closing
9722 or value.type ~= "delimiter"
9723 or not (value.element == "link"
9724 or value.element == "image"

```

```

9725 or value.element == "note")
9726 or value.removed then
9727 goto continue
9728 end
9729
9730 local opener_position = find_link_opener(t, 1, i - 1)
9731 if (opener_position == nil) then
9732 goto continue
9733 end
9734
9735 local opening_delimiter = t[opener_position]
9736 opening_delimiter.removed = true
9737
9738 local link_type = opening_delimiter.link_type
9739
9740 if (link_type == "inline") then
9741 resolve_inline_link(t, opener_position, i)
9742 end
9743 if (link_type == "shortcut") then
9744 resolve_shortcut_link(t, opener_position, i)
9745 end
9746 if (link_type == "full") then
9747 resolve_full_link(t, opener_position, i)
9748 end
9749 if (link_type == "collapsed") then
9750 resolve_collapsed_link(t, opener_position, i)
9751 end
9752 if (link_type == "note_inline") then
9753 resolve_note_inline_link(t, opener_position, i)
9754 end
9755 if (link_type == "raw_note") then
9756 resolve_raw_note_link(t, opener_position, i)
9757 end
9758
9759 ::continue::
9760 end
9761
9762 t[#t].content = t[#t].content:gsub("%s*$", "")
9763
9764 process_emphasis(t, 1, #t)
9765 local final_result = collect_emphasis_content(t, 1, #t)
9766 return final_result
9767 end
9768
9769 function self.defer_link_and_emphasis_processing(delimiter_table)
9770 return writer.defer_call(function()
9771 return process_links_and_emphasis(delimiter_table)

```

```
9772 end)
9773 end
9774
```

### 3.1.6.8 Inline Elements (local)

```
9775 parsers.Str = (parsers.normalchar
9776 * (parsers.normalchar + parsers.at)^0)
9777 / writer.string
9778
9779 parsers.Symbol = (parsers.backtick^1 + V("SpecialChar"))
9780 / writer.string
9781
9782 parsers.Ellipsis = P("...") / writer.ellipsis
9783
9784 parsers.Smart = parsers.Ellipsis
9785
9786 parsers.Code = parsers.inticks / writer.code
9787
9788 if options.blankBeforeBlockquote then
9789 parsers.bqstart = parsers.fail
9790 else
9791 parsers.bqstart = parsers.blockquote_start
9792 end
9793
9794 if options.blankBeforeHeading then
9795 parsers.headerstart = parsers.fail
9796 else
9797 parsers.headerstart = parsers.atx_heading
9798 end
9799
9800 if options.blankBeforeList then
9801 parsers.interrupting_bullets = parsers.fail
9802 parsers.interrupting_enumerators = parsers.fail
9803 else
9804 parsers.interrupting_bullets
9805 = parsers.bullet(parsers.dash, true)
9806 + parsers.bullet(parsers.asterisk, true)
9807 + parsers.bullet(parsers.plus, true)
9808
9809 parsers.interrupting_enumerators
9810 = parsers.enumerator(parsers.period, true)
9811 + parsers.enumerator(parsers.rparent, true)
9812 end
9813
9814 if options.html then
9815 parsers.html_interrupting
```



```

9816 = parsers.check_trail
9817 * (parsers.html_incomplete_open_tag
9818 + parsers.html_incomplete_close_tag
9819 + parsers.html_incomplete_open_special_tag
9820 + parsers.html_comment_start
9821 + parsers.html_cdatasection_start
9822 + parsers.html_declaration_start
9823 + parsers.html_instruction_start
9824 - parsers.html_close_special_tag
9825 - parsers.html_empty_special_tag)
9826 else
9827 parsers.html_interrupting = parsers.fail
9828 end
9829
9830 parsers.ListStarter = parsers.starter
9831
9832 parsers.EndlineExceptions
9833 = parsers.blankline -- paragraph break
9834 + parsers.eof -- end of document
9835 + parsers.bqstart
9836 + parsers.thematic_break_lines
9837 + parsers.interrupting_bullets
9838 + parsers.interrupting_enumerators
9839 + parsers.headerstart
9840 + parsers.html_interrupting
9841
9842 parsers.NoSoftLineBreakEndlineExceptions = parsers.EndlineExceptions
9843
9844 parsers.endline = parsers.newline
9845 * (parsers.check_minimal_indent
9846 * -V("EndlineExceptions")
9847 + parsers.check_optional_indent
9848 * -V("EndlineExceptions")
9849 * -V("ListStarter")) / function(_) return end
9850 * parsers.spacechar~0
9851
9852 parsers.Endline = parsers.endline
9853 / writer.soft_line_break
9854
9855 parsers.EndlineNoSub = parsers.endline
9856
9857 parsers.NoSoftLineBreakEndline
9858 = parsers.newline
9859 * (parsers.check_minimal_indent
9860 * -V("NoSoftLineBreakEndlineExceptions")
9861 + parsers.check_optional_indent
9862 * -V("NoSoftLineBreakEndlineExceptions")

```

```

9863 * -V("ListStarter"))
9864 * parsers.spacechar^0
9865 / writer.space
9866
9867 parsers.EndlineBreak = parsers.backslash * parsers.endline
9868 / writer.hard_line_break
9869
9870 parsers.OptionalIndent
9871 = parsers.spacechar^1 / writer.space
9872
9873 parsers.Space = parsers.spacechar^2 * parsers.endline
9874 / writer.hard_line_break
9875 + parsers.spacechar^1
9876 * parsers.endline^-1
9877 * parsers.eof / self.expandtabs
9878 + parsers.spacechar^1 * parsers.endline
9879 / writer.soft_line_break
9880 + parsers.spacechar^1
9881 * -parsers.newline / self.expandtabs
9882 + parsers.spacechar^1
9883
9884 parsers.NoSoftLineBreakSpace
9885 = parsers.spacechar^2 * parsers.endline
9886 / writer.hard_line_break
9887 + parsers.spacechar^1
9888 * parsers.endline^-1
9889 * parsers.eof / self.expandtabs
9890 + parsers.spacechar^1 * parsers.endline
9891 / writer.soft_line_break
9892 + parsers.spacechar^1
9893 * -parsers.newline / self.expandtabs
9894 + parsers.spacechar^1
9895
9896 parsers.NonbreakingEndline
9897 = parsers.endline
9898 / writer.nbsp
9899
9900 parsers.NonbreakingSpace
9901 = parsers.spacechar^2 * parsers.endline
9902 / writer.nbsp
9903 + parsers.spacechar^1
9904 * parsers.endline^-1 * parsers.eof / ""
9905 + parsers.spacechar^1 * parsers.endline
9906 * parsers.optionalspace
9907 / writer.nbsp
9908 + parsers.spacechar^1 * parsers.optionalspace
9909 / writer.nbsp

```

9910

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.

```
9911 function self.auto_link_url(url, attributes)
9912 return writer.link(writer.escape(url),
9913 url, nil, attributes)
9914 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.

```
9915 function self.auto_link_email(email, attributes)
9916 return writer.link(writer.escape(email),
9917 "mailto:".email,
9918 nil, attributes)
9919 end
9920
9921 parsers.AutoLinkUrl = parsers.auto_link_url
9922 / self.auto_link_url
9923
9924 parsers.AutoLinkEmail
9925 = parsers.auto_link_email
9926 / self.auto_link_email
9927
9928 parsers.AutoLinkRelativeReference
9929 = parsers.auto_link_relative_reference
9930 / self.auto_link_url
9931
9932 parsers.LinkAndEmph = Ct(parsers.link_and_emph_table)
9933 / self.defer_link_and_emphasis_processing
9934
9935 parsers.EscapedChar = parsers.backslash
9936 * C(parsers.escapable) / writer.string
9937
9938 parsers.InlineHtml = Cs(parsers.html_inline_comment)
9939 / writer.inline_html_comment
9940 + Cs(parsers.html_any_empty_inline_tag
9941 + parsers.html_inline_instruction
9942 + parsers.html_inline_cdatasection
9943 + parsers.html_inline_declaration
9944 + parsers.html_any_open_inline_tag
9945 + parsers.html_any_close_tag)
9946 / writer.inline_html_tag
9947
9948 parsers.HtmlEntity = parsers.html_entities / writer.string
```

### 3.1.6.9 Block Elements (local)

```
9949 parsers.DisplayHtml = Cs(parsers.check_trail
9950 * (parsers.html_comment
9951 + parsers.html_special_block
9952 + parsers.html_block
9953 + parsers.html_any_block
9954 + parsers.html_instruction
9955 + parsers.html_cdatasection
9956 + parsers.html_declaration))
9957 / writer.block_html_element
9958
9959 parsers.indented_non_blank_line = parsers.indentedline
9960 - parsers.blankline
9961
9962 parsers.Verbatim
9963 = Cs(parsers.check_code_trail
9964 * (parsers.line - parsers.blankline)
9965 * ((parsers.check_minimal_blank_indent_and_full_code_trail
9966 * parsers.blankline)^0
9967 * ((parsers.check_minimal_indent / "")
9968 * parsers.check_code_trail
9969 * (parsers.line - parsers.blankline))^1)^0)
9970 / self.expandtabs / writer.verbatim
9971
9972 parsers.Blockquote = parsers.blockquote_body
9973 / writer.blockquote
9974
9975 parsers.ThematicBreak = parsers.thematic_break_lines
9976 / writer.thematic_break
9977
9978 parsers.Reference = parsers.define_reference_parser
9979 / self.register_link
9980
9981 parsers.Paragraph = parsers.freeze_trail
9982 * (Ct((parsers.Inline)^1)
9983 * (parsers.newline + parsers.eof)
9984 * parsers.unfreeze_trail
9985 / writer.paragraph)
9986
9987 parsers.Plain = parsers.nonindentspace * Ct(parsers.Inline^1)
9988 / writer.plain
```

### 3.1.6.10 Lists (local)

```
9989
9990 if options.taskLists then
9991 parsers.tickbox = (parsers.ticked_box
```

```

9992 + parsers.halfticked_box
9993 + parsers.unticked_box
9994) / writer.tickbox
9995 else
9996 parsers.tickbox = parsers.fail
9997 end
9998
9999 parsers.list_blank = parsers.conditionally_indented_blankline
10000
10001 parsers.ref_or_block_list_separated
10002 = parsers.sep_group_no_output(parsers.list_blank)
10003 * parsers.minimally_indented_ref
10004 + parsers.block_sep_group(parsers.list_blank)
10005 * parsers.minimally_indented_block
10006
10007 parsers.ref_or_block_non_separated
10008 = parsers.minimally_indented_ref
10009 + (parsers.succeed / writer.interblocksep)
10010 * parsers.minimally_indented_block
10011 - parsers.minimally_indented_blankline
10012
10013 parsers.tight_list_loop_body_pair =
10014 parsers.create_loop_body_pair(
10015 parsers.ref_or_block_non_separated,
10016 parsers.minimally_indented_par_or_plain_no_blank,
10017 (parsers.succeed / writer.interblocksep),
10018 (parsers.succeed / writer.paragraphsep))
10019
10020 parsers.loose_list_loop_body_pair =
10021 parsers.create_loop_body_pair(
10022 parsers.ref_or_block_list_separated,
10023 parsers.minimally_indented_par_or_plain,
10024 parsers.block_sep_group(parsers.list_blank),
10025 parsers.par_sep_group(parsers.list_blank))
10026
10027 parsers.tight_list_content_loop
10028 = V("Block")
10029 * parsers.tight_list_loop_body_pair.block^0
10030 + (V("Paragraph") + V("Plain"))
10031 * parsers.ref_or_block_non_separated
10032 * parsers.tight_list_loop_body_pair.block^0
10033 + (V("Paragraph") + V("Plain"))
10034 * parsers.tight_list_loop_body_pair.par^0
10035
10036 parsers.loose_list_content_loop
10037 = V("Block")
10038 * parsers.loose_list_loop_body_pair.block^0

```

```

10039 + (V("Paragraph") + V("Plain"))
10040 * parsers.ref_or_block_list_separated
10041 * parsers.loose_list_loop_body_pair.block^0
10042 + (V("Paragraph") + V("Plain"))
10043 * parsers.loose_list_loop_body_pair.par^0
10044
10045 parsers.list_item_tightness_condition
10046 = -(parsers.list_blank^0
10047 * parsers.minimally_indented_ref_or_block_or_par)
10048 * remove_indent("li")
10049 + remove_indent("li")
10050 * parsers.fail
10051
10052 parsers.indented_content_tight
10053 = Ct((parsers.blankline / "")
10054 * #parsers.list_blank
10055 * remove_indent("li")
10056 + ((V("Reference") + (parsers.blankline / ""))
10057 * parsers.check_minimal_indent
10058 * parsers.tight_list_content_loop
10059 + (V("Reference") + (parsers.blankline / ""))
10060 + (parsers.tickbox^-1 / writer.escape)
10061 * parsers.tight_list_content_loop
10062)
10063 * parsers.list_item_tightness_condition)
10064
10065 parsers.indented_content_loose
10066 = Ct((parsers.blankline / "")
10067 * #parsers.list_blank
10068 + ((V("Reference") + (parsers.blankline / ""))
10069 * parsers.check_minimal_indent
10070 * parsers.loose_list_content_loop
10071 + (V("Reference") + (parsers.blankline / ""))
10072 + (parsers.tickbox^-1 / writer.escape)
10073 * parsers.loose_list_content_loop))
10074
10075 parsers.TightListItem = function(starter)
10076 return -parsers.ThematicBreak
10077 * parsers.add_indent(starter, "li")
10078 * parsers.indented_content_tight
10079 end
10080
10081 parsers.LooseListItem = function(starter)
10082 return -parsers.ThematicBreak
10083 * parsers.add_indent(starter, "li")
10084 * parsers.indented_content_loose
10085 * remove_indent("li")

```

```

10086 end
10087
10088 parsers.BulletListOfType = function(bullet_type)
10089 local bullet = parsers.bullet(bullet_type)
10090 return (Ct(parsers.TightListItem(bullet)
10091 * ((parsers.check_minimal_indent / "")
10092 * parsers.TightListItem(bullet)
10093)^0
10094)
10095 * Cc(true)
10096 * -#((parsers.list_blank^0 / "")
10097 * parsers.check_minimal_indent
10098 * (bullet - parsers.ThematicBreak)
10099)
10100 + Ct(parsers.LooseListItem(bullet)
10101 * ((parsers.list_blank^0 / "")
10102 * (parsers.check_minimal_indent / "")
10103 * parsers.LooseListItem(bullet)
10104)^0
10105)
10106 * Cc(false)
10107) / writer.bulletlist
10108 end
10109
10110 parsers.BulletList = parsers.BulletListOfType(parsers.dash)
10111 + parsers.BulletListOfType(parsers.asterisk)
10112 + parsers.BulletListOfType(parsers.plus)
10113
10114 local function ordered_list(items,tight,starter)
10115 local startnum = starter[2][1]
10116 if options.startNumber then
10117 startnum = tonumber(startnum) or 1 -- fallback for '#'
10118 if startnum ~= nil then
10119 startnum = math.floor(startnum)
10120 end
10121 else
10122 startnum = nil
10123 end
10124 return writer.orderedlist(items,tight,startnum)
10125 end
10126
10127 parsers.OrderedListOfType = function(delimiter_type)
10128 local enumerator = parsers.enumerator(delimiter_type)
10129 return Cg(enumerator, "listtype")
10130 * (Ct(parsers.TightListItem(Cb("listtype"))
10131 * ((parsers.check_minimal_indent / "")
10132 * parsers.TightListItem(enumerator))^0

```

```

10133 * Cc(true)
10134 * -#((parsers.list_blank^0 / ""
10135 * parsers.check_minimal_indent * enumerator)
10136 + Ct(parsers.LooseListItem(Cb("listtype")))
10137 * ((parsers.list_blank^0 / ""
10138 * (parsers.check_minimal_indent / ""
10139 * parsers.LooseListItem(enumerator))^0)
10140 * Cc(false)
10141) * Ct(Cb("listtype")) / ordered_list
10142 end
10143
10144 parsers.OrderedList = parsers.OrderedListOfType(parsers.period)
10145 + parsers.OrderedListOfType(parsers.rparent)

```

### 3.1.6.11 Blank (local)

```

10146 parsers.Blank = parsers.blankline / ""
10147 + V("Reference")

```

### 3.1.6.12 Headings (local)

```

10148 function parsers.parse_heading_text(s)
10149 local inlines = self.parser_functions.parse_inlines(s)
10150 local flatten_inlines = self.writer.flatten_inlines
10151 self.writer.flatten_inlines = true
10152 local flat_text = self.parser_functions.parse_inlines(s)
10153 flat_text = util.ropes_to_string(flat_text)
10154 self.writer.flatten_inlines = flatten_inlines
10155 return {flat_text, inlines}
10156 end
10157
10158 -- parse atx header
10159 parsers.AtxHeading = parsers.check_trail_no_rem
10160 * Cg(parsers.heading_start, "level")
10161 * ((C(parsers.optionalspace
10162 * parsers.hash^0
10163 * parsers.optionalspace
10164 * parsers.newline)
10165 + parsers.spacechar^1
10166 * C(parsers.line))
10167 / strip_atx_end
10168 / parsers.parse_heading_text)
10169 * Cb("level")
10170 / writer.heading
10171
10172 parsers.heading_line = parsers.linechar^1
10173 - parsers.thematic_break_lines
10174

```



```

10175 parsers.heading_text = parsers.heading_line
10176 * ((V("Endline") / "\n")
10177 * (parsers.heading_line
10178 - parsers.heading_level))^0
10179 * parsers.newline^-1
10180
10181 parsers.SetextHeading = parsers.freeze_trail
10182 * parsers.check_trail_no_rem
10183 * #(parsers.heading_text
10184 * parsers.check_minimal_indent
10185 * parsers.check_trail
10186 * parsers.heading_level)
10187 * Cs(parsers.heading_text)
10188 / parsers.parse_heading_text
10189 * parsers.check_minimal_indent_and_trail
10190 * parsers.heading_level
10191 * parsers.newline
10192 * parsers.unfreeze_trail
10193 / writer.heading
10194
10195 parsers.Heading = parsers.AtxHeading + parsers.SetextHeading

```

### 3.1.6.13 Syntax Specification

Define `reader->finalize_grammar` as a function that constructs the PEG grammar of markdown, applies syntax extensions `extensions` and returns a conversion function that takes a markdown string and turns it into a plain  $\text{\TeX}$  output.

```

10196 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.

```

10197 local walkable_syntax = (function(global_walkable_syntax)
10198 local local_walkable_syntax = {}
10199 for lhs, rule in pairs(global_walkable_syntax) do
10200 local_walkable_syntax[lhs] = util.table_copy(rule)
10201 end
10202 return local_walkable_syntax
10203 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.

```

10204 local current_extension_name = nil
10205 self.insert_pattern = function(selector, pattern, pattern_name)

```

```

10206 assert(pattern_name == nil or type(pattern_name) == "string")
10207 local _, _, lhs, pos, rhs
10208 = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
10209 assert(lhs ~= nil,
10210 [[Expected selector in form]]
10211 .. [[LHS (before|after|instead of) RHS", not "]]
10212 .. selector .. ["])
10213 assert(walkable_syntax[lhs] ~= nil,
10214 [[Rule]] .. lhs
10215 .. [[-> ... does not exist in markdown grammar]])
10216 assert(pos == "before" or pos == "after" or pos == "instead of",
10217 [[Expected positional specifier "before", "after",]]
10218 .. [[or "instead of", not "]]
10219 .. pos .. ["])
10220 local rule = walkable_syntax[lhs]
10221 local index = nil
10222 for current_index, current_rhs in ipairs(rule) do
10223 if type(current_rhs) == "string" and current_rhs == rhs then
10224 index = current_index
10225 if pos == "after" then
10226 index = index + 1
10227 end
10228 break
10229 end
10230 end
10231 assert(index ~= nil,
10232 [[Rule]] .. lhs .. [[->]] .. rhs
10233 .. [[does not exist in markdown grammar]])
10234 local accountable_pattern
10235 if current_extension_name then
10236 accountable_pattern
10237 = {pattern, current_extension_name, pattern_name}
10238 else
10239 assert(type(pattern) == "string",
10240 [[reader->insert_pattern() was called outside]]
10241 .. [[an extension with]]
10242 .. [[a PEG pattern instead of a rule name]])
10243 accountable_pattern = pattern
10244 end
10245 if pos == "instead of" then
10246 rule[index] = accountable_pattern
10247 else
10248 table.insert(rule, index, accountable_pattern)
10249 end
10250 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.

```

10251 local syntax =
10252 { "Blocks",
10253
10254 Blocks = V("InitializeState")
10255 * V("ExpectedJekyllData")
10256 * 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.

```

10257 * (V("Block")
10258 * (V("Blank")^0 * parsers.eof
10259 + (V("Blank")^2 / writer.paragraphsep
10260 + V("Blank")^0 / writer.interblocksep
10261)
10262)
10263 + (V("Paragraph") + V("Plain"))
10264 * (V("Blank")^0 * parsers.eof
10265 + (V("Blank")^2 / writer.paragraphsep
10266 + V("Blank")^0 / writer.interblocksep
10267)
10268)
10269 * V("Block")
10270 * (V("Blank")^0 * parsers.eof
10271 + (V("Blank")^2 / writer.paragraphsep
10272 + V("Blank")^0 / writer.interblocksep
10273)
10274)
10275 + (V("Paragraph") + V("Plain"))
10276 * (V("Blank")^0 * parsers.eof
10277 + V("Blank")^0 / writer.paragraphsep
10278)
10279)^0,
10280
10281 ExpectedJekyllData = parsers.succeed,
10282
10283 Blank = parsers.Blank,
10284 Reference = parsers.Reference,
10285
10286 Blockquote = parsers.Blockquote,
10287 Verbatim = parsers.Verbatim,
10288 ThematicBreak = parsers.ThematicBreak,
10289 BulletList = parsers.BulletList,
10290 OrderedList = parsers.OrderedList,
10291 DisplayHtml = parsers.DisplayHtml,

```

```

10292 Heading = parsers.Heading,
10293 Paragraph = parsers.Paragraph,
10294 Plain = parsers.Plain,
10295
10296 ListStarter = parsers.ListStarter,
10297 EndlineExceptions = parsers.EndlineExceptions,
10298 NoSoftLineBreakEndlineExceptions
10299 = parsers.NoSoftLineBreakEndlineExceptions,
10300
10301 Str = parsers.Str,
10302 Space = parsers.Space,
10303 NoSoftLineBreakSpace
10304 = parsers.NoSoftLineBreakSpace,
10305 OptionalIndent = parsers.OptionalIndent,
10306 Endline = parsers.Endline,
10307 EndlineNoSub = parsers.EndlineNoSub,
10308 NoSoftLineBreakEndline
10309 = parsers.NoSoftLineBreakEndline,
10310 EndlineBreak = parsers.EndlineBreak,
10311 LinkAndEmph = parsers.LinkAndEmph,
10312 Code = parsers.Code,
10313 AutoLinkUrl = parsers.AutoLinkUrl,
10314 AutoLinkEmail = parsers.AutoLinkEmail,
10315 AutoLinkRelativeReference
10316 = parsers.AutoLinkRelativeReference,
10317 InlineHtml = parsers.InlineHtml,
10318 HtmlEntity = parsers.HtmlEntity,
10319 EscapedChar = parsers.EscapedChar,
10320 Smart = parsers.Smart,
10321 Symbol = parsers.Symbol,
10322 SpecialChar = parsers.fail,
10323 InitializeState = parsers.succeed,
10324 }

```

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]`.

```

10325 self.update_rule = function(rule_name, get_pattern)
10326 assert(current_extension_name ~= nil)
10327 assert(syntax[rule_name] ~= nil,
10328 [[Rule]] .. rule_name
10329 .. [[-> ... does not exist in markdown grammar]])
10330 local previous_pattern
10331 local extension_name
10332 if walkable_syntax[rule_name] then

```

```

10333 local previous_accountable_pattern
10334 = walkable_syntax[rule_name][1]
10335 previous_pattern = previous_accountable_pattern[1]
10336 extension_name
10337 = previous_accountable_pattern[2]
10338 .. ", " .. current_extension_name
10339 else
10340 previous_pattern = nil
10341 extension_name = current_extension_name
10342 end
10343 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

```

```

10344 if type(get_pattern) == "function" then
10345 pattern = get_pattern(previous_pattern)
10346 else
10347 assert(previous_pattern == nil,
10348 [[Rule]] .. rule_name ..
10349 [[has already been updated by]] .. extension_name)
10350 pattern = get_pattern
10351 end
10352 local accountable_pattern = { pattern, extension_name, rule_name }
10353 walkable_syntax[rule_name] = { accountable_pattern }
10354 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.

```

10355 local special_characters = {}
10356 self.add_special_character = function(c)
10357 table.insert(special_characters, c)
10358 syntax.SpecialChar = S(table.concat(special_characters, ""))
10359 end
10360
10361 self.add_special_character("*")
10362 self.add_special_character("[")
10363 self.add_special_character("]")
10364 self.add_special_character("<")

```

```

10365 self.add_special_character("!")
10366 self.add_special_character("\\")

```

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

```

10367 self.initialize_named_group = function(name, value)
10368 local pattern = Ct("")
10369 if value ~= nil then
10370 pattern = pattern / value
10371 end
10372 syntax.InitializeState = syntax.InitializeState
10373 * Cg(pattern, name)
10374 end

```

Add a named group for indentation.

```

10375 self.initialize_named_group("indent_info")

```

Apply syntax extensions.

```

10376 for _, extension in ipairs(extensions) do
10377 current_extension_name = extension.name
10378 extension.extend_writer(writer)
10379 extension.extend_reader(self)
10380 end
10381 current_extension_name = nil

```

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

```

10382 if options.debugExtensions then
10383 local sorted_lhs = {}
10384 for lhs, _ in pairs(walkable_syntax) do
10385 table.insert(sorted_lhs, lhs)
10386 end
10387 table.sort(sorted_lhs)
10388
10389 local output_lines = {"{"}
10390 for lhs_index, lhs in ipairs(sorted_lhs) do
10391 local encoded_lhs = util.encode_json_string(lhs)
10392 table.insert(output_lines, [[]] .. encoded_lhs .. [[:]])
10393 local rule = walkable_syntax[lhs]
10394 for rhs_index, rhs in ipairs(rule) do
10395 local human_readable_rhs
10396 if type(rhs) == "string" then
10397 human_readable_rhs = rhs
10398 else
10399 local pattern_name
10400 if rhs[3] then
10401 pattern_name = rhs[3]
10402 else

```

```

10403 pattern_name = "Anonymous Pattern"
10404 end
10405 local extension_name = rhs[2]
10406 human_readable_rhs = pattern_name .. [[(]]
10407 .. extension_name .. [[]]]
10408 end
10409 local encoded_rhs
10410 = util.encode_json_string(human_readable_rhs)
10411 local output_line = [[]] .. encoded_rhs
10412 if rhs_index < #rule then
10413 output_line = output_line .. ", "
10414 end
10415 table.insert(output_lines, output_line)
10416 end
10417 local output_line = "]"
10418 if lhs_index < #sorted_lhs then
10419 output_line = output_line .. ", "
10420 end
10421 table.insert(output_lines, output_line)
10422 end
10423 table.insert(output_lines, "}")
10424
10425 local output = table.concat(output_lines, "\n")
10426 local output_filename = options.debugExtensionsFileName
10427 local output_file = assert(io.open(output_filename, "w"),
10428 [[Could not open file]] .. output_filename
10429 .. [[for writing]])
10430 assert(output_file:write(output))
10431 assert(output_file:close())
10432 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`.

```

10433 for lhs, rule in pairs(walkable_syntax) do
10434 syntax[lhs] = parsers.fail
10435 for _, rhs in ipairs(rule) do
10436 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.

```

10437 if type(rhs) == "string" then
10438 pattern = V(rhs)
10439 else
10440 pattern = rhs[1]

```

```

10441 if type(pattern) == "string" then
10442 pattern = V(pattern)
10443 end
10444 end
10445 syntax[lhs] = syntax[lhs] + pattern
10446 end
10447 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.

```

10448 if options.underscores then
10449 self.add_special_character("_")
10450 end
10451
10452 if not options.codeSpans then
10453 syntax.Code = parsers.fail
10454 else
10455 self.add_special_character("`")
10456 end
10457
10458 if not options.html then
10459 syntax.DisplayHtml = parsers.fail
10460 syntax.InlineHtml = parsers.fail
10461 syntax.HtmlEntity = parsers.fail
10462 else
10463 self.add_special_character("&")
10464 end
10465
10466 if options.preserveTabs then
10467 options.stripIndent = false
10468 end
10469
10470 if not options.smartEllipses then
10471 syntax.Smart = parsers.fail
10472 else
10473 self.add_special_character(".")
10474 end
10475
10476 if not options.relativeReferences then
10477 syntax.AutoLinkRelativeReference = parsers.fail
10478 end
10479
10480 if options.contentLevel == "inline" then
10481 syntax[1] = "Inlines"
10482 syntax.Inlines = V("InitializeState")
10483 * parsers.Inline^0

```



```

10484 * (parsers.spacing^0
10485 * parsers.eof / "")
10486 syntax.Space = parsers.Space + parsers.blankline / writer.space
10487 end
10488
10489 local blocks_nested_t = util.table_copy(syntax)
10490 blocks_nested_t.ExpectedJekyllData = parsers.succeed
10491 parsers.blocks_nested = Ct(blocks_nested_t)
10492
10493 parsers.blocks = Ct(syntax)
10494
10495 local inlines_t = util.table_copy(syntax)
10496 inlines_t[1] = "Inlines"
10497 inlines_t.Inlines = V("InitializeState")
10498 * parsers.Inline^0
10499 * (parsers.spacing^0
10500 * parsers.eof / "")
10501 parsers.inlines = Ct(inlines_t)
10502
10503 local inlines_no_inline_note_t = util.table_copy(inlines_t)
10504 inlines_no_inline_note_t.InlineNote = parsers.fail
10505 parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
10506
10507 local inlines_no_html_t = util.table_copy(inlines_t)
10508 inlines_no_html_t.DisplayHtml = parsers.fail
10509 inlines_no_html_t.InlineHtml = parsers.fail
10510 inlines_no_html_t.HtmlEntity = parsers.fail
10511 parsers.inlines_no_html = Ct(inlines_no_html_t)
10512
10513 local inlines_nbsp_t = util.table_copy(inlines_t)
10514 inlines_nbsp_t.Endline = parsers.NonbreakingEndline
10515 inlines_nbsp_t.Space = parsers.NonbreakingSpace
10516 parsers.inlines_nbsp = Ct(inlines_nbsp_t)
10517
10518 local inlines_no_link_or_emphasis_t = util.table_copy(inlines_t)
10519 inlines_no_link_or_emphasis_t.LinkAndEmph = parsers.fail
10520 inlines_no_link_or_emphasis_t.EndlineExceptions
10521 = parsers.EndlineExceptions - parsers.eof
10522 parsers.inlines_no_link_or_emphasis
10523 = Ct(inlines_no_link_or_emphasis_t)

```

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

```

10524 return function(input)

```

Unicode-normalize the input.

```

10525 if options.unicodeNormalization then
10526 local form = options.unicodeNormalizationForm

```

```

10527 if form == "nfc" then
10528 input = uni_algos.normalize.NFC(input)
10529 elseif form == "nfd" then
10530 input = uni_algos.normalize.NFD(input)
10531 elseif form == "nfkc" then
10532 input = uni_algos.normalize.NFKC(input)
10533 elseif form == "nfkd" then
10534 input = uni_algos.normalize.NFKD(input)
10535 else
10536 return writer.error(
10537 format("Unknown normalization form %s.", form))
10538 end
10539 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.

```

10540 input = input:gsub("\r\n?", "\n")
10541 if input:sub(-1) ~= "\n" then
10542 input = input .. "\n"
10543 end

```

Clear the table of references.

```

10544 references = {}
10545 local document = self.parser_functions.parse_blocks(input)
10546 local output = util.rope_to_string(writer.document(document))

```

Remove block element / paragraph separators immediately followed by the output of [writer->undosep](#), possibly interleaved by section ends. Then, remove any leftover output of [writer->undosep](#).

```

10547 local undosep_start, undosep_end
10548 local potential_secend_start, secend_start
10549 local potential_sep_start, sep_start
10550 while true do
10551 -- find a `writer->undosep`
10552 undosep_start, undosep_end
10553 = output:find(writer.undosep_text, 1, true)
10554 if undosep_start == nil then break end
10555 -- skip any preceding section ends
10556 secend_start = undosep_start
10557 while true do
10558 potential_secend_start = secend_start - #writer.secend_text
10559 if potential_secend_start < 1
10560 or output:sub(potential_secend_start,
10561 secend_start - 1) ~= writer.secend_text
10562 then
10563 break
10564 end
10565 secend_start = potential_secend_start

```

```

10566 end
10567 -- find an immediately preceding
10568 -- block element / paragraph separator
10569 sep_start = secend_start
10570 potential_sep_start = sep_start - #writer.interblocksep_text
10571 if potential_sep_start >= 1
10572 and output:sub(potential_sep_start,
10573 sep_start - 1) == writer.interblocksep_text
10574 then
10575 sep_start = potential_sep_start
10576 else
10577 potential_sep_start = sep_start - #writer.paragraphsep_text
10578 if potential_sep_start >= 1
10579 and output:sub(potential_sep_start,
10580 sep_start - 1) == writer.paragraphsep_text
10581 then
10582 sep_start = potential_sep_start
10583 end
10584 end
10585 -- remove `writer->undosep` and immediately preceding
10586 -- block element / paragraph separator
10587 output = output:sub(1, sep_start - 1)
10588 .. output:sub(secend_start, undosep_start - 1)
10589 .. output:sub(undosep_end + 1)
10590 end
10591 return output
10592 end
10593 end
10594 return self
10595 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.

```
10596 M.extensions = {}
```

#### 3.1.7.1 Bracketed Spans

The `extensions.bracketed_spans` function implements the Pandoc bracketed span syntax extension.

```
10597 M.extensions.bracketed_spans = function()
10598 return {

```

```

10599 name = "built-in bracketed_spans syntax extension",
10600 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.
10601 function self.span(s, attr)
10602 if self.flatten_inlines then return s end
10603 return {"\\markdownRendererBracketedSpanAttributeContextBegin",
10604 self.attributes(attr),
10605 s,
10606 "\\markdownRendererBracketedSpanAttributeContextEnd{}}"}
10607 end
10608 end, extend_reader = function(self)
10609 local parsers = self.parsers
10610 local writer = self.writer
10611
10612 local span_label = parsers.lbracket
10613 * (Cs((parsers.alphanumeric^1
10614 + parsers.inticks
10615 + parsers.autolink
10616 + V("InlineHtml")
10617 + (parsers.backslash * parsers.backslash)
10618 + (parsers.backslash
10619 * (parsers.lbracket + parsers.rbracket)
10620 + V("Space") + V("Endline")
10621 + (parsers.any
10622 - (parsers.newline
10623 + parsers.lbracket
10624 + parsers.rbracket
10625 + parsers.blankline^2))))^1)
10626 / self.parser_functions.parse_inlines)
10627 * parsers.rbracket
10628
10629 local Span = span_label
10630 * Ct(parsers.attributes)
10631 / writer.span
10632
10633 self.insert_pattern("Inline before LinkAndEmph",
10634 Span, "Span")
10635 end
10636 }
10637 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.

```
10638 M.extensions.citations = function(citation_nbsps)
10639 return {
10640 name = "built-in citations syntax extension",
10641 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.

```
10642 function self.citations(text_cites, cites)
10643 local buffer = {}
10644 if self.flatten_inlines then
10645 for _,cite in ipairs(cites) do
10646 if cite.prenote then
10647 table.insert(buffer, {cite.prenote, " "})
10648 end
10649 table.insert(buffer, cite.name)
10650 if cite.postnote then
10651 table.insert(buffer, {" ", cite.postnote})
10652 end
10653 end
10654 else
10655 table.insert(buffer,
10656 {"\\markdownRenderer",
10657 text_cites and "TextCite" or "Cite",
10658 "{", #cites, "}"})
10659 for _,cite in ipairs(cites) do
10660 table.insert(buffer,
10661 {cite.suppress_author and "-" or "+", "{",
10662 cite.prenote or "", "}{"},
10663 {cite.postnote or "", "}{"}, cite.name, "}")
10664 end
10665 end
10666 return buffer
```

```

10667 end
10668 end, extend_reader = function(self)
10669 local parsers = self.parsers
10670 local writer = self.writer
10671
10672 local citation_chars
10673 = parsers.alphanumeric
10674 + S("#$%&-+<>~/_")
10675
10676 local citation_name
10677 = Cs(parsers.dash^-1) * parsers.at
10678 * Cs(citation_chars
10679 * (((citation_chars
10680 + parsers.internal_punctuation
10681 - parsers.comma - parsers.semicolon)
10682 * -#((parsers.internal_punctuation
10683 - parsers.comma
10684 - parsers.semicolon)^0
10685 * -(citation_chars
10686 + parsers.internal_punctuation
10687 - parsers.comma
10688 - parsers.semicolon)))^0
10689 * citation_chars)^-1)
10690
10691 local citation_body_prenote
10692 = Cs((parsers.alphanumeric^1
10693 + parsers.bracketed
10694 + parsers.inticks
10695 + parsers.autolink
10696 + V("InlineHtml")
10697 + V("Space") + V("EndlineNoSub")
10698 + (parsers.anyescaped
10699 - (parsers.newline
10700 + parsers.rbracket
10701 + parsers.blankline^2))
10702 - (parsers.spnl
10703 * parsers.dash^-1
10704 * parsers.at))^1)
10705
10706 local citation_body_postnote
10707 = Cs((parsers.alphanumeric^1
10708 + parsers.bracketed
10709 + parsers.inticks
10710 + parsers.autolink
10711 + V("InlineHtml")
10712 + V("Space") + V("EndlineNoSub")
10713 + (parsers.anyescaped

```

```

10714 - (parsers.newline
10715 + parsers.rbracket
10716 + parsers.semicolon
10717 + parsers.blankline^2))
10718 - (parsers.spnl * parsers.rbracket))^1)
10719
10720 local citation_body_chunk
10721 = (citation_body_prenote
10722 * parsers.spnlc_sep
10723 + Cc("")
10724 * parsers.spnlc
10725)
10726 * citation_name
10727 * (parsers.internal_punctuation
10728 - parsers.semicolon)^-1
10729 * (parsers.spnlc / function(_) return end
10730 * citation_body_postnote
10731 + Cc("")
10732 * parsers.spnlc
10733)
10734
10735 local citation_body
10736 = citation_body_chunk
10737 * (parsers.semicolon
10738 * parsers.spnlc
10739 * citation_body_chunk
10740)^0
10741
10742 local citation_headless_body_postnote
10743 = Cs((parsers.alphanumeric^1
10744 + parsers.bracketed
10745 + parsers.inticks
10746 + parsers.autolink
10747 + V("InlineHtml")
10748 + V("Space") + V("Endline")
10749 + (parsers.anyescaped
10750 - (parsers.newline
10751 + parsers.rbracket
10752 + parsers.at
10753 + parsers.semicolon + parsers.blankline^2))
10754 - (parsers.spnl * parsers.rbracket))^0)
10755
10756 local citation_headless_body
10757 = citation_headless_body_postnote
10758 * (parsers.semicolon
10759 * parsers.spnlc
10760 * citation_body_chunk

```

```

10761)^0
10762
10763 local citations
10764 = function(text_cites, raw_cites)
10765 local function normalize(str)
10766 if str == "" then
10767 str = nil
10768 else
10769 str = (citation_nbsps and
10770 self.parser_functions.parse_inlines_nbsp or
10771 self.parser_functions.parse_inlines)(str)
10772 end
10773 return str
10774 end
10775
10776 local cites = {}
10777 for i = 1,#raw_cites,4 do
10778 cites[#cites+1] = {
10779 prenote = normalize(raw_cites[i]),
10780 suppress_author = raw_cites[i+1] == "-",
10781 name = writer.identifier(raw_cites[i+2]),
10782 postnote = normalize(raw_cites[i+3]),
10783 }
10784 end
10785 return writer.citations(text_cites, cites)
10786 end
10787
10788 local TextCitations
10789 = Ct((parsers.spnlc
10790 * Cc("")
10791 * citation_name
10792 * ((parsers.spnlc
10793 * parsers.lbracket
10794 * citation_headless_body
10795 * parsers.rbracket) + Cc("")))^1)
10796 / function(raw_cites)
10797 return citations(true, raw_cites)
10798 end
10799
10800 local ParenthesizedCitations
10801 = Ct((parsers.spnlc
10802 * parsers.lbracket
10803 * citation_body
10804 * parsers.rbracket)^1)
10805 / function(raw_cites)
10806 return citations(false, raw_cites)
10807 end

```



```

10808
10809 local Citations = TextCitations + ParenthesizedCitations
10810
10811 self.insert_pattern("Inline before LinkAndEmph",
10812 Citations, "Citations")
10813
10814 self.add_special_character("@")
10815 self.add_special_character("-")
10816 end
10817 }
10818 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.

```

10819 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.

```

10820 local languages_json = (function()
10821 local base, prev, curr
10822 for _, pathname in ipairs{kpse.lookup(language_map,
10823 {all=true})} do
10824 local file = io.open(pathname, "r")
10825 if not file then goto continue end
10826 local input = assert(file:read("*a"))
10827 assert(file:close())
10828 local json = input:gsub('[^\n]-:', '[%1]=')
10829 curr = load("_ENV = {}; return"..json)()
10830 if type(curr) == "table" then
10831 if base == nil then
10832 base = curr
10833 else
10834 setmetatable(prev, { __index = curr })
10835 end
10836 prev = curr
10837 end
10838 ::continue::
10839 end
10840 return base or {}
10841 end)()
10842
10843 return {

```

```

10844 name = "built-in content_blocks syntax extension",
10845 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.

```

10846 function self.contentblock(src,suf,type,tit)
10847 if not self.is_writing then return "" end
10848 src = src.." "..suf
10849 suf = suf:lower()
10850 if type == "onlineimage" then
10851 return {"\\markdownRendererContentBlockOnlineImage{" ,suf,"} ",
10852 {" ,self.string(src),"} ",
10853 {" ,self.uri(src),"} ",
10854 {" ,self.string(tit or ""),"} "}
10855 elseif languages_json[suf] then
10856 return {"\\markdownRendererContentBlockCode{" ,suf,"} ",
10857 {" ,self.string(languages_json[suf]),"} ",
10858 {" ,self.string(src),"} ",
10859 {" ,self.uri(src),"} ",
10860 {" ,self.string(tit or ""),"} "}
10861 else
10862 return {"\\markdownRendererContentBlock{" ,suf,"} ",
10863 {" ,self.string(src),"} ",
10864 {" ,self.uri(src),"} ",
10865 {" ,self.string(tit or ""),"} "}
10866 end
10867 end
10868 end, extend_reader = function(self)
10869 local parsers = self.parsers
10870 local writer = self.writer
10871
10872 local contentblock_tail
10873 = parsers.optionaltitle
10874 * (parsers.newline + parsers.eof)
10875
10876 -- case insensitive online image suffix:
10877 local onlineimagesuffix
10878 = (function(...)
10879 local parser = nil
10880 for _, suffix in ipairs({...}) do
10881 local pattern=nil
10882 for i=1,#suffix do
10883 local char=suffix:sub(i,i)
10884 char = S(char:lower()..char:upper())
10885 if pattern == nil then

```

```

10886 pattern = char
10887 else
10888 pattern = pattern * char
10889 end
10890 end
10891 if parser == nil then
10892 parser = pattern
10893 else
10894 parser = parser + pattern
10895 end
10896 end
10897 return parser
10898 end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
10899
10900 -- online image url for iA Writer content blocks with
10901 -- mandatory suffix, allowing nested brackets:
10902 local onlineimageurl
10903 = (parsers.less
10904 * Cs((parsers.anyescaped
10905 - parsers.more
10906 - parsers.spacing
10907 - #(parsers.period
10908 * onlineimagesuffix
10909 * parsers.more
10910 * contentblock_tail))^0)
10911 * parsers.period
10912 * Cs(onlineimagesuffix)
10913 * parsers.more
10914 + (Cs((parsers.inparens
10915 + (parsers.anyescaped
10916 - parsers.spacing
10917 - parsers.rparent
10918 - #(parsers.period
10919 * onlineimagesuffix
10920 * contentblock_tail))))^0)
10921 * parsers.period
10922 * Cs(onlineimagesuffix))
10923) * Cc("onlineimage")
10924
10925 -- filename for iA Writer content blocks with mandatory suffix:
10926 local localfilepath
10927 = parsers.slash
10928 * Cs((parsers.anyescaped
10929 - parsers.tab
10930 - parsers.newline
10931 - #(parsers.period
10932 * parsers.alphanumeric^1

```

```

10933 * contentblock_tail))^1)
10934 * parsers.period
10935 * Cs(parsers.alphanumeric^1)
10936 * Cc("localfile")
10937
10938 local ContentBlock
10939 = parsers.check_trail_no_rem
10940 * (localfilepath + onlineimageurl)
10941 * contentblock_tail
10942 / writer.contentblock
10943
10944 self.insert_pattern("Block before Blockquote",
10945 ContentBlock, "ContentBlock")
10946 end
10947 }
10948 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.

```

10949 M.extensions.definition_lists = function(tight_lists)
10950 return {
10951 name = "built-in definition_lists syntax extension",
10952 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.

```

10953 local function dlistem(term, defs)
10954 local retVal = {"\\markdownRendererDlItem{",term,""}
10955 for _, def in ipairs(defs) do
10956 retVal[#retVal+1]
10957 = {"\\markdownRendererDlDefinitionBegin ",def,
10958 "\\markdownRendererDlDefinitionEnd "}
10959 end
10960 retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
10961 return retVal
10962 end
10963
10964 function self.definitionlist(items,tight)
10965 if not self.is_writing then return "" end
10966 local buffer = {}
10967 for _,item in ipairs(items) do
10968 buffer[#buffer + 1] = dlistem(item.term, item.definitions)

```

```

10969 end
10970 if tight and tight_lists then
10971 return {"\\markdownRendererDlBeginTight\n", buffer,
10972 "\\markdownRendererDlEndTight"}
10973 else
10974 return {"\\markdownRendererDlBegin\n", buffer,
10975 "\\markdownRendererDlEnd"}
10976 end
10977 end
10978 end, extend_reader = function(self)
10979 local parsers = self.parsers
10980 local writer = self.writer
10981
10982 local defstartchar = S("~:")
10983
10984 local defstart
10985 = parsers.check_trail_length(0) * defstartchar
10986 * #parsers.spacing
10987 * (parsers.tab + parsers.space^-3)
10988 + parsers.check_trail_length(1)
10989 * defstartchar * #parsers.spacing
10990 * (parsers.tab + parsers.space^-2)
10991 + parsers.check_trail_length(2)
10992 * defstartchar * #parsers.spacing
10993 * (parsers.tab + parsers.space^-1)
10994 + parsers.check_trail_length(3)
10995 * defstartchar * #parsers.spacing
10996
10997 local indented_line
10998 = (parsers.check_minimal_indent / "")
10999 * parsers.check_code_trail * parsers.line
11000
11001 local blank
11002 = parsers.check_minimal_blank_indent_and_any_trail
11003 * parsers.optionalspace * parsers.newline
11004
11005 local dlchunk = Cs(parsers.line * (indented_line - blank)^0)
11006
11007 local indented_blocks = function(bl)
11008 return Cs(bl
11009 * (blank^1 * (parsers.check_minimal_indent / ""))
11010 * parsers.check_code_trail * -parsers.blankline * bl)^0
11011 * (blank^1 + parsers.eof))
11012 end
11013
11014 local function definition_list_item(term, defs, _)
11015 return { term = self.parser_functions.parse_inlines(term),

```

```

11016 definitions = defs }
11017 end
11018
11019 local DefinitionListItemLoose
11020 = C(parsers.line) * blank^0
11021 * Ct((parsers.check_minimal_indent * (defstart
11022 * indented_blocks(dlchunk)
11023 / self.parser_functions.parse_blocks_nested))^1)
11024 * Cc(false) / definition_list_item
11025
11026 local DefinitionListItemTight
11027 = C(parsers.line)
11028 * Ct((parsers.check_minimal_indent * (defstart * dlchunk
11029 / self.parser_functions.parse_blocks_nested))^1)
11030 * Cc(true) / definition_list_item
11031
11032 local DefinitionList
11033 = (Ct(DefinitionListItemLoose^1) * Cc(false)
11034 + Ct(DefinitionListItemTight^1)
11035 * (blank^0
11036 * -DefinitionListItemLoose * Cc(true))
11037) / writer.definitionlist
11038
11039 self.insert_pattern("Block after Heading",
11040 DefinitionList, "DefinitionList")
11041 end
11042 }
11043 end

```

### 3.1.7.5 Fancy Lists

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

```

11044 M.extensions.fancy_lists = function()
11045 return {
11046 name = "built-in fancy_lists syntax extension",
11047 extend_writer = function(self)
11048 local options = self.options
11049

```

Define `writer->fancylist` as a function that will transform an input ordered list to the output format, where:

- `items` is an array of the list items,
- `tight` specifies, whether the list is tight or not,
- `startnum` is the number of the first list item,
- `numstyle` is the style of the list item labels from among the following:

- `Decimal` – decimal arabic numbers,
  - `LowerRoman` – lower roman numbers,
  - `UpperRoman` – upper roman numbers,
  - `LowerAlpha` – lower ASCII alphabetic characters, and
  - `UpperAlpha` – upper ASCII alphabetic characters, and
- `numdelim` is the style of delimiters between list item labels and texts from among the following:
    - `Default` – default style,
    - `OneParen` – parentheses, and
    - `Period` – periods.

```

11050 function self.fancylist(items,tight,startnum,numstyle,numdelim)
11051 if not self.is_writing then return "" end
11052 local buffer = {}
11053 local num = startnum
11054 for _,item in ipairs(items) do
11055 if item ~= "" then
11056 buffer[#buffer + 1] = self.fancyitem(item,num)
11057 end
11058 if num ~= nil and item ~= "" then
11059 num = num + 1
11060 end
11061 end
11062 local contents = util.intersperse(buffer,"\n")
11063 if tight and options.tightLists then
11064 return {"\markdownRendererFancy01BeginTight{",
11065 numstyle,"}{",numdelim,"}",contents,
11066 "\n\markdownRendererFancy01EndTight "}
11067 else
11068 return {"\markdownRendererFancy01Begin{",
11069 numstyle,"}{",numdelim,"}",contents,
11070 "\n\markdownRendererFancy01End "}
11071 end
11072 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.

```

11073 function self.fancyitem(s,num)
11074 if num ~= nil then
11075 return {"\markdownRendererFancy01ItemWithNumber{",num,"}",s,
11076 "\n\markdownRendererFancy01ItemEnd "}
11077 else
11078 return {"\markdownRendererFancy01Item ",s,
11079 "\n\markdownRendererFancy01ItemEnd "}

```

```

11080 end
11081 end
11082 end, extend_reader = function(self)
11083 local parsers = self.parsers
11084 local options = self.options
11085 local writer = self.writer
11086
11087 local function combine_markers_and_delims(markers, delims)
11088 local markers_table = {}
11089 for _,marker in ipairs(markers) do
11090 local start_marker
11091 local continuation_marker
11092 if type(marker) == "table" then
11093 start_marker = marker[1]
11094 continuation_marker = marker[2]
11095 else
11096 start_marker = marker
11097 continuation_marker = marker
11098 end
11099 for _,delim in ipairs(delims) do
11100 table.insert(markers_table,
11101 {start_marker, continuation_marker, delim})
11102 end
11103 end
11104 return markers_table
11105 end
11106
11107 local function join_table_with_func(func, markers_table)
11108 local pattern = func(table.unpack(markers_table[1]))
11109 for i = 2, #markers_table do
11110 pattern = pattern + func(table.unpack(markers_table[i]))
11111 end
11112 return pattern
11113 end
11114
11115 local lowercase_letter_marker = R("az")
11116 local uppercase_letter_marker = R("AZ")
11117
11118 local roman_marker = function(chars)
11119 local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])
11120 local l, x, v, i
11121 = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])
11122 return m^-3
11123 * (c*m + c*d + d^-1 * c^-3)
11124 * (x*c + x*l + l^-1 * x^-3)
11125 * (i*x + i*v + v^-1 * i^-3)
11126 end

```



```

11127
11128 local lowercase_roman_marker
11129 = roman_marker({"m", "d", "c", "l", "x", "v", "i"})
11130 local uppercase_roman_marker
11131 = roman_marker({"M", "D", "C", "L", "X", "V", "I"})
11132
11133 local lowercase_opening_roman_marker = P("i")
11134 local uppercase_opening_roman_marker = P("I")
11135
11136 local digit_marker = parsers.dig * parsers.dig-8
11137
11138 local markers = {
11139 {lowercase_opening_roman_marker, lowercase_roman_marker},
11140 {uppercase_opening_roman_marker, uppercase_roman_marker},
11141 lowercase_letter_marker,
11142 uppercase_letter_marker,
11143 lowercase_roman_marker,
11144 uppercase_roman_marker,
11145 digit_marker
11146 }
11147
11148 local delims = {
11149 parsers.period,
11150 parsers.rparent
11151 }
11152
11153 local markers_table = combine_markers_and_delims(markers, delims)
11154
11155 local function enumerator(start_marker, _,
11156 delimiter_type, interrupting)
11157 local delimiter_range
11158 local allowed_end
11159 if interrupting then
11160 delimiter_range = P("1")
11161 allowed_end = C(parsers.spacechar1) * #parsers.linechar
11162 else
11163 delimiter_range = start_marker
11164 allowed_end = C(parsers.spacechar1)
11165 + #(parsers.newline + parsers.eof)
11166 end
11167
11168 return parsers.check_trail
11169 * Ct(C(delimiter_range) * C(delimiter_type))
11170 * allowed_end
11171 end
11172
11173 local starter = join_table_with_func(enumerator, markers_table)

```

```

11174
11175 local TightListItem = function(starter)
11176 return parsers.add_indent(starter, "li")
11177 * parsers.indented_content_tight
11178 end
11179
11180 local LooseListItem = function(starter)
11181 return parsers.add_indent(starter, "li")
11182 * parsers.indented_content_loose
11183 * remove_indent("li")
11184 end
11185
11186 local function roman2number(roman)
11187 local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100,
11188 ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
11189 local numeral = 0
11190
11191 local i = 1
11192 local len = string.len(roman)
11193 while i < len do
11194 local z1, z2 = romans[string.sub(roman, i, i)],
11195 romans[string.sub(roman, i+1, i+1)]
11196 if z1 < z2 then
11197 numeral = numeral + (z2 - z1)
11198 i = i + 2
11199 else
11200 numeral = numeral + z1
11201 i = i + 1
11202 end
11203 end
11204 if i <= len then
11205 numeral = numeral + romans[string.sub(roman,i,i)]
11206 end
11207 return numeral
11208 end
11209
11210 local function sniffstyle(numstr, delimend)
11211 local numdelim
11212 if delimend == ")" then
11213 numdelim = "OneParen"
11214 elseif delimend == "." then
11215 numdelim = "Period"
11216 else
11217 numdelim = "Default"
11218 end
11219
11220 local num

```

```

11221 num = numstr:match("^([I])$")
11222 if num then
11223 return roman2number(num), "UpperRoman", numdelim
11224 end
11225 num = numstr:match("^([i])$")
11226 if num then
11227 return roman2number(string.upper(num)), "LowerRoman", numdelim
11228 end
11229 num = numstr:match("^([A-Z])$")
11230 if num then
11231 return string.byte(num) - string.byte("A") + 1,
11232 "UpperAlpha", numdelim
11233 end
11234 num = numstr:match("^([a-z])$")
11235 if num then
11236 return string.byte(num) - string.byte("a") + 1,
11237 "LowerAlpha", numdelim
11238 end
11239 num = numstr:match("^([IVXLCDM]+)")
11240 if num then
11241 return roman2number(num), "UpperRoman", numdelim
11242 end
11243 num = numstr:match("^([ivxlcdm]+)")
11244 if num then
11245 return roman2number(string.upper(num)), "LowerRoman", numdelim
11246 end
11247 return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
11248 end
11249
11250 local function fancylist(items,tight,start)
11251 local startnum, numstyle, numdelim
11252 = sniffstyle(start[2][1], start[2][2])
11253 return writer.fancylist(items,tight,
11254 options.startNumber and startnum or 1,
11255 numstyle or "Decimal",
11256 numdelim or "Default")
11257 end
11258
11259 local FancyListOfType
11260 = function(start_marker, continuation_marker, delimiter_type)
11261 local enumerator_start
11262 = enumerator(start_marker, continuation_marker,
11263 delimiter_type)
11264 local enumerator_cont
11265 = enumerator(continuation_marker, continuation_marker,
11266 delimiter_type)
11267 return Cg(enumerator_start, "listtype")

```

```

11268 * (Ct(TightListItem(Cb("listtype"))
11269 * ((parsers.check_minimal_indent / "")
11270 * TightListItem(enumerator_cont))^0)
11271 * Cc(true)
11272 * -#((parsers.conditionally_indented_blankline^0 / "")
11273 * parsers.check_minimal_indent * enumerator_cont)
11274 + Ct(LooseListItem(Cb("listtype"))
11275 * ((parsers.conditionally_indented_blankline^0 / "")
11276 * (parsers.check_minimal_indent / "")
11277 * LooseListItem(enumerator_cont))^0)
11278 * Cc(false)
11279) * Ct(Cb("listtype")) / fancylist
11280 end
11281
11282 local FancyList
11283 = join_table_with_func(FancyListOfType, markers_table)
11284
11285 local ListStarter = starter
11286
11287 self.update_rule("OrderedList", FancyList)
11288 self.update_rule("ListStarter", ListStarter)
11289 end
11290 }
11291 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.

```

11292 M.extensions.fenced_code = function(blank_before_code_fence,
11293 allow_attributes,
11294 allow_raw_blocks)
11295 return {
11296 name = "built-in fenced_code syntax extension",
11297 extend_writer = function(self)
11298 local options = self.options
11299

```

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.

```

11300 function self.fencedCode(s, i, attr)

```

```

11301 if not self.is_writing then return "" end
11302 s = s:gsub("\n$", "")
11303 local buf = {}
11304 if attr ~= nil then
11305 table.insert(buf,
11306 {"\\markdownRendererFencedCodeAttributeContextBegin",
11307 self.attributes(attr)})
11308 end
11309 local name = util.cache_verbatim(options.cacheDir, s)
11310 table.insert(buf,
11311 {"\\markdownRendererInputFencedCode{" ,
11312 name,"}{" ,self.string(i),"}{" ,self.infostring(i),"}"}")
11313 if attr ~= nil then
11314 table.insert(buf,
11315 "\\markdownRendererFencedCodeAttributeContextEnd{")
11316 end
11317 return buf
11318 end
11319

```

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

```

11320 if allow_raw_blocks then
11321 function self.rawBlock(s, attr)
11322 if not self.is_writing then return "" end
11323 s = s:gsub("\n$", "")
11324 local name = util.cache_verbatim(options.cacheDir, s)
11325 return {"\\markdownRendererInputRawBlock{" ,
11326 name,"}{" , self.string(attr),"}"}
11327 end
11328 end
11329 end, extend_reader = function(self)
11330 local parsers = self.parsers
11331 local writer = self.writer
11332
11333 local function captures_geq_length(_,i,a,b)
11334 return #a >= #b and i
11335 end
11336
11337 local function strip_enclosing_whitespaces(str)
11338 return str:gsub("^%s*(.)%s*$", "%1")
11339 end
11340
11341 local tilde_infostring = Cs(Cs((V("HtmlEntity")
11342 + parsers.anyescaped
11343 - parsers.newline)^0)
11344 / strip_enclosing_whitespaces)

```

```

11345
11346 local backtick_infostring
11347 = Cs(Cs((V("HtmlEntity")
11348 + (-#(parsers.backslash * parsers.backtick)
11349 * parsers.anyescaped)
11350 - parsers.newline
11351 - parsers.backtick)^0)
11352 / strip_enclosing_whitespaces)
11353
11354 local fenceindent
11355
11356 local function has_trail(indent_table)
11357 return indent_table ~= nil and
11358 indent_table.trail ~= nil and
11359 next(indent_table.trail) ~= nil
11360 end
11361
11362 local function has_indents(indent_table)
11363 return indent_table ~= nil and
11364 indent_table.indents ~= nil and
11365 next(indent_table.indents) ~= nil
11366 end
11367
11368 local function get_last_indent_name(indent_table)
11369 if has_indents(indent_table) then
11370 return indent_table.indents[#indent_table.indents].name
11371 end
11372 end
11373
11374 local count_fenced_start_indent =
11375 function(_, _, indent_table, trail)
11376 local last_indent_name = get_last_indent_name(indent_table)
11377 fenceindent = 0
11378 if last_indent_name ~= "li" then
11379 fenceindent = #trail
11380 end
11381 return true
11382 end
11383
11384 local fencehead = function(char, infostring)
11385 return Cmt(Cb("indent_info")
11386 * parsers.check_trail, count_fenced_start_indent)
11387 * Cg(char^3, "fencelength")
11388 * parsers.optionalspace
11389 * infostring
11390 * (parsers.newline + parsers.eof)
11391 end

```

```

11392
11393 local fencetail = function(char)
11394 return parsers.check_trail_no_rem
11395 * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
11396 * parsers.optionalspace * (parsers.newline + parsers.eof)
11397 + parsers.eof
11398 end
11399
11400 local process_fenced_line =
11401 function(s, i, -- luacheck: ignore s i
11402 indent_table, line_content, is_blank)
11403 local remainder = ""
11404 if has_trail(indent_table) then
11405 remainder = indent_table.trail.internal_remainder
11406 end
11407
11408 if is_blank
11409 and get_last_indent_name(indent_table) == "li" then
11410 remainder = ""
11411 end
11412
11413 local str = remainder .. line_content
11414 local index = 1
11415 local remaining = fenceindent
11416
11417 while true do
11418 local c = str:sub(index, index)
11419 if c == " " and remaining > 0 then
11420 remaining = remaining - 1
11421 index = index + 1
11422 elseif c == "\t" and remaining > 3 then
11423 remaining = remaining - 4
11424 index = index + 1
11425 else
11426 break
11427 end
11428 end
11429
11430 return true, str:sub(index)
11431 end
11432
11433 local fencedline = function(char)
11434 return Cmt(Cb("indent_info")
11435 * C(parsers.line - fencetail(char))
11436 * Cc(false), process_fenced_line)
11437 end
11438

```

```

11439 local blankfencedline
11440 = Cmt(Cb("indent_info")
11441 * C(parsers.blankline)
11442 * Cc(true), process_fenced_line)
11443
11444 local TildeFencedCode
11445 = fencehead(parsers.tilde, tilde_infostring)
11446 * Cs((parsers.check_minimal_blank_indent / "")
11447 * blankfencedline
11448 + (parsers.check_minimal_indent / "")
11449 * fencedline(parsers.tilde))^0)
11450 * ((parsers.check_minimal_indent / "")
11451 * fencetail(parsers.tilde) + parsers.succeed)
11452
11453 local BacktickFencedCode
11454 = fencehead(parsers.backtick, backtick_infostring)
11455 * Cs((parsers.check_minimal_blank_indent / "")
11456 * blankfencedline
11457 + (parsers.check_minimal_indent / "")
11458 * fencedline(parsers.backtick))^0)
11459 * ((parsers.check_minimal_indent / "")
11460 * fencetail(parsers.backtick) + parsers.succeed)
11461
11462 local infostring_with_attributes
11463 = Ct(C((parsers.linechar
11464 - (parsers.optionalspace
11465 * parsers.attributes))^0)
11466 * parsers.optionalspace
11467 * Ct(parsers.attributes))
11468
11469 local FencedCode
11470 = ((TildeFencedCode + BacktickFencedCode)
11471 / function(infostring, code)
11472 local expanded_code = self.expandtabs(code)
11473
11474 if allow_raw_blocks then
11475 local raw_attr = lpeg.match(parsers.raw_attribute,
11476 infostring)
11477
11478 if raw_attr then
11479 return writer.rawBlock(expanded_code, raw_attr)
11480 end
11481 end
11482
11483 local attr = nil
11484 if allow_attributes then
11485 local match = lpeg.match(infostring_with_attributes,
11486 infostring)

```



```

11486 if match then
11487 infostring, attr = table.unpack(match)
11488 end
11489 end
11490 return writer.fencedCode(expanded_code, infostring, attr)
11491 end)
11492
11493 self.insert_pattern("Block after Verbatim",
11494 FencedCode, "FencedCode")
11495
11496 local fencestart
11497 if blank_before_code_fence then
11498 fencestart = parsers.fail
11499 else
11500 fencestart = fencehead(parsers.backtick, backtick_infostring)
11501 + fencehead(parsers.tilde, tilde_infostring)
11502 end
11503
11504 self.update_rule("EndlineExceptions", function(previous_pattern)
11505 if previous_pattern == nil then
11506 previous_pattern = parsers.EndlineExceptions
11507 end
11508 return previous_pattern + fencestart
11509 end)
11510
11511 self.add_special_character("`")
11512 self.add_special_character("~")
11513 end
11514 }
11515 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.

```

11516 M.extensions.fenced_divs = function(blank_before_div_fence)
11517 return {
11518 name = "built-in fenced_divs syntax extension",
11519 extend_writer = function(self)

```

Define `writer->div_begin` as a function that will transform the beginning of an input fenced div with with attributes `attributes` to the output format.

```

11520 function self.div_begin(attributes)
11521 local start_output
11522 = {"\\markdownRendererFencedDivAttributeContextBegin\n",

```

```

11523 self.attributes(attributes)}
11524 local end_output
11525 = {"\\markdownRendererFencedDivAttributeContextEnd{}}"}
11526 return self.push_attributes(
11527 "div", attributes, start_output, end_output)
11528 end

```

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

```

11529 function self.div_end()
11530 return self.pop_attributes("div")
11531 end
11532 end, extend_reader = function(self)
11533 local parsers = self.parsers
11534 local writer = self.writer

```

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

```

11535 local fenced_div_infostring
11536 = C((parsers.linechar
11537 - (parsers.spacechar^1
11538 * parsers.colon^1))^1)
11539
11540 local fenced_div_begin = parsers.nonindentspace
11541 * parsers.colon^3
11542 * parsers.optionalspace
11543 * fenced_div_infostring
11544 * (parsers.spacechar^1
11545 * parsers.colon^1)^0
11546 * parsers.optionalspace
11547 * (parsers.newline + parsers.eof)
11548
11549 local fenced_div_end = parsers.nonindentspace
11550 * parsers.colon^3
11551 * parsers.optionalspace
11552 * (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.

```

11553 self.initialize_named_group("fenced_div_level", "0")
11554 self.initialize_named_group("fenced_div_num_opening_indents")
11555
11556 local function increment_div_level()
11557 local push_indent_table =
11558 function(s, i, indent_table, -- luacheck: ignore s i

```

```

11559 fenced_div_num_opening_indents, fenced_div_level)
11560 fenced_div_level = tonumber(fenced_div_level) + 1
11561 local num_opening_indents = 0
11562 if indent_table.indents ~= nil then
11563 num_opening_indents = #indent_table.indents
11564 end
11565 fenced_div_num_opening_indents[fenced_div_level]
11566 = num_opening_indents
11567 return true, fenced_div_num_opening_indents
11568 end
11569
11570 local increment_level =
11571 function(s, i, fenced_div_level) -- luacheck: ignore s i
11572 fenced_div_level = tonumber(fenced_div_level) + 1
11573 return true, tostring(fenced_div_level)
11574 end
11575
11576 return Cg(Cmt(Cb("indent_info")
11577 * Cb("fenced_div_num_opening_indents")
11578 * Cb("fenced_div_level"), push_indent_table)
11579 , "fenced_div_num_opening_indents")
11580 * Cg(Cmt(Cb("fenced_div_level"), increment_level)
11581 , "fenced_div_level")
11582 end
11583
11584 local function decrement_div_level()
11585 local pop_indent_table =
11586 function(s, i, -- luacheck: ignore s i
11587 fenced_div_indent_table, fenced_div_level)
11588 fenced_div_level = tonumber(fenced_div_level)
11589 fenced_div_indent_table[fenced_div_level] = nil
11590 return true, tostring(fenced_div_level - 1)
11591 end
11592
11593 return Cg(Cmt(Cb("fenced_div_num_opening_indents")
11594 * Cb("fenced_div_level"), pop_indent_table)
11595 , "fenced_div_level")
11596 end
11597
11598
11599 local non_fenced_div_block
11600 = parsers.check_minimal_indent * V("Block")
11601 - parsers.check_minimal_indent_and_trail * fenced_div_end
11602
11603 local non_fenced_div_paragraph
11604 = parsers.check_minimal_indent * V("Paragraph")
11605 - parsers.check_minimal_indent_and_trail * fenced_div_end

```

```

11606
11607 local blank = parsers.minimally_indented_blank
11608
11609 local block_separated = parsers.block_sep_group(blank)
11610 * non_fenced_div_block
11611
11612 local loop_body_pair
11613 = parsers.create_loop_body_pair(block_separated,
11614 non_fenced_div_paragraph,
11615 parsers.block_sep_group(blank),
11616 parsers.par_sep_group(blank))
11617
11618 local content_loop = (non_fenced_div_block
11619 * loop_body_pair.block^0
11620 + non_fenced_div_paragraph
11621 * block_separated
11622 * loop_body_pair.block^0
11623 + non_fenced_div_paragraph
11624 * loop_body_pair.par^0)
11625 * blank^0
11626
11627 local FencedDiv = fenced_div_begin
11628 / function (infostring)
11629 local attr
11630 = lpeg.match(Ct(parsers.attributes),
11631 infostring)
11632 if attr == nil then
11633 attr = {"." .. infostring}
11634 end
11635 return attr
11636 end
11637 / writer.div_begin
11638 * increment_div_level()
11639 * parsers.skipblanklines
11640 * Ct(content_loop)
11641 * parsers.minimally_indented_blank^0
11642 * parsers.check_minimal_indent_and_trail
11643 * fenced_div_end
11644 * decrement_div_level()
11645 * (Cc("") / writer.div_end)
11646
11647 self.insert_pattern("Block after Verbatim",
11648 FencedDiv, "FencedDiv")
11649
11650 self.add_special_character(":")
11651

```

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.

```
11652 local function is_inside_div()
11653 local check_div_level =
11654 function(s, i, fenced_div_level) -- luacheck: ignore s i
11655 fenced_div_level = tonumber(fenced_div_level)
11656 return fenced_div_level > 0
11657 end
11658
11659 return Cmt(Cb("fenced_div_level"), check_div_level)
11660 end
11661
11662 local function check_indent()
11663 local compare_indent =
11664 function(s, i, indent_table, -- luacheck: ignore s i
11665 fenced_div_num_opening_indents, fenced_div_level)
11666 fenced_div_level = tonumber(fenced_div_level)
11667 local num_current_indents
11668 = (indent_table.current_line_indents ~= nil and
11669 #indent_table.current_line_indents) or 0
11670 local num_opening_indents
11671 = fenced_div_num_opening_indents[fenced_div_level]
11672 return num_current_indents == num_opening_indents
11673 end
11674
11675 return Cmt(Cb("indent_info")
11676 * Cb("fenced_div_num_opening_indents")
11677 * Cb("fenced_div_level"), compare_indent)
11678 end
11679
11680 local fencestart = is_inside_div()
11681 * fenced_div_end
11682 * check_indent()
11683
11684 if not blank_before_div_fence then
11685 self.update_rule("EndlineExceptions", function(previous_pattern)
11686 if previous_pattern == nil then
11687 previous_pattern = parsers.EndlineExceptions
11688 end
11689 return previous_pattern + fencestart
11690 end)
11691 end
11692 end
11693 }
11694 end
```

### 3.1.7.8 Header Attributes

The `extensions.header_attributes` function implements the Pandoc header attribute syntax extension.

```
11695 M.extensions.header_attributes = function()
11696 return {
11697 name = "built-in header_attributes syntax extension",
11698 extend_writer = function()
11699 end, extend_reader = function(self)
11700 local parsers = self.parsers
11701 local writer = self.writer
11702
11703 local function strip_atx_end(s)
11704 return s:gsub("%s+##%s*$", "")
11705 end
11706
11707 local AtxHeading = Cg(parsers.heading_start, "level")
11708 * parsers.optionalspace
11709 * (C(((parsers.linechar
11710 - (parsers.attributes
11711 * parsers.optionalspace
11712 * parsers.newline))
11713 * (parsers.linechar
11714 - parsers.lbrace)^0)^1)
11715 / strip_atx_end
11716 / parsers.parse_heading_text)
11717 * Cg(Ct(parsers.newline
11718 + (parsers.attributes
11719 * parsers.optionalspace
11720 * parsers.newline)), "attributes")
11721 * Cb("level")
11722 * Cb("attributes")
11723 / writer.heading
11724
11725 local function strip_trailing_spaces(s)
11726 return s:gsub("%s*$", "")
11727 end
11728
11729 local heading_line = (parsers.linechar
11730 - (parsers.attributes
11731 * parsers.optionalspace
11732 * parsers.newline))^1
11733 - parsers.thematic_break_lines
11734
11735 local heading_text
11736 = heading_line
11737 * ((V("Endline") / "\n")
11738 * (heading_line - parsers.heading_level))^0
```

```

11739 * parsers.newline^-1
11740
11741 local SetextHeading
11742 = parsers.freeze_trail * parsers.check_trail_no_rem
11743 * #(heading_text
11744 * (parsers.attributes
11745 * parsers.optionalspace
11746 * parsers.newline)^-1
11747 * parsers.check_minimal_indent
11748 * parsers.check_trail
11749 * parsers.heading_level)
11750 * Cs(heading_text) / strip_trailing_spaces
11751 / parsers.parse_heading_text
11752 * Cg(Ct((parsers.attributes
11753 * parsers.optionalspace
11754 * parsers.newline)^-1), "attributes")
11755 * parsers.check_minimal_indent_and_trail * parsers.heading_level
11756 * Cb("attributes")
11757 * parsers.newline
11758 * parsers.unfreeze_trail
11759 / writer.heading
11760
11761 local Heading = AtxHeading + SetextHeading
11762 self.update_rule("Heading", Heading)
11763 end
11764 }
11765 end

```

### 3.1.7.9 Inline Code Attributes

The `extensions.inline_code_attributes` function implements the Pandoc inline code attribute syntax extension.

```

11766 M.extensions.inline_code_attributes = function()
11767 return {
11768 name = "built-in inline_code_attributes syntax extension",
11769 extend_writer = function()
11770 end, extend_reader = function(self)
11771 local writer = self.writer
11772
11773 local CodeWithAttributes = parsers.inticks
11774 * Ct(parsers.attributes)
11775 / writer.code
11776
11777 self.insert_pattern("Inline before Code",
11778 CodeWithAttributes,
11779 "CodeWithAttributes")
11780 end

```

```

11781 }
11782 end

```

### 3.1.7.10 Line Blocks

The `extensions.line_blocks` function implements the Pandoc line block syntax extension.

```

11783 M.extensions.line_blocks = function()
11784 return {
11785 name = "built-in line_blocks syntax extension",
11786 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.

```

11787 function self.lineblock(lines)
11788 if not self.is_writing then return "" end
11789 local buffer = {}
11790 for i = 1, #lines - 1 do
11791 buffer[#buffer + 1] = { lines[i], self.hard_line_break }
11792 end
11793 buffer[#buffer + 1] = lines[#lines]
11794
11795 return {"\\markdownRendererLineBlockBegin\n"
11796 ,buffer,
11797 "\n\\markdownRendererLineBlockEnd "}
11798 end
11799 end, extend_reader = function(self)
11800 local parsers = self.parsers
11801 local writer = self.writer
11802
11803 local LineBlock
11804 = Ct((Cs(((parsers.pipe * parsers.space) / ""
11805 * ((parsers.space)/entities.char_entity("nbsp"))^0
11806 * parsers.linechar^0 * (parsers.newline/"")
11807 * (-parsers.pipe
11808 * (parsers.space^1/" ")
11809 * parsers.linechar^1
11810 * (parsers.newline/"")
11811)^0
11812 * (parsers.blankline/"")^0)
11813 / self.parser_functions.parse_inlines)^1)
11814 / writer.lineblock
11815
11816 self.insert_pattern("Block after Blockquote",
11817 LineBlock, "LineBlock")
11818 end
11819 }
11820 end

```



### 3.1.7.11 Marked text

The `extensions.mark` function implements the Pandoc mark syntax extension.

```
11821 M.extensions.mark = function()
11822 return {
11823 name = "built-in mark syntax extension",
11824 extend_writer = function(self)
```

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

```
11825 function self.mark(s)
11826 if self.flatten_inlines then return s end
11827 return {"\\markdownRendererMark{" , s, "}"}
11828 end
11829 end, extend_reader = function(self)
11830 local parsers = self.parsers
11831 local writer = self.writer
11832
11833 local doubleequals = P("==")
11834
11835 local Mark
11836 = parsers.between(V("Inline"), doubleequals, doubleequals)
11837 / function (inlines) return writer.mark(inlines) end
11838
11839 self.add_special_character("=")
11840 self.insert_pattern("Inline before LinkAndEmph",
11841 Mark, "Mark")
11842 end
11843 }
11844 end
```

### 3.1.7.12 Link Attributes

The `extensions.link_attributes` function implements the Pandoc link attribute syntax extension.

```
11845 M.extensions.link_attributes = function()
11846 return {
11847 name = "built-in link_attributes syntax extension",
11848 extend_writer = function()
11849 end, extend_reader = function(self)
11850 local parsers = self.parsers
11851 local options = self.options
11852
```

The following patterns define link reference definitions with attributes.

```
11853 local define_reference_parser
11854 = (parsers.check_trail / "")
11855 * parsers.link_label
11856 * parsers.colon
```

```

11857 * parsers.spnlc * parsers.url
11858 * (parsers.spnlc_sep * parsers.title
11859 * (parsers.spnlc * Ct(parsers.attributes))
11860 * parsers.only_blank
11861 + parsers.spnlc_sep * parsers.title * parsers.only_blank
11862 + Cc("") * (parsers.spnlc * Ct(parsers.attributes))
11863 * parsers.only_blank
11864 + Cc("") * parsers.only_blank)
11865
11866 local ReferenceWithAttributes = define_reference_parser
11867 / self.register_link
11868
11869 self.update_rule("Reference", ReferenceWithAttributes)
11870

```

The following patterns define direct and indirect links with attributes.

```

11871
11872 local LinkWithAttributesAndEmph
11873 = Ct(parsers.link_and_emph_table * Cg(Cc(true),
11874 "match_link_attributes"))
11875 / self.defer_link_and_emphasis_processing
11876
11877 self.update_rule("LinkAndEmph", LinkWithAttributesAndEmph)
11878

```

The following patterns define autolinks with attributes.

```

11879 local AutoLinkUrlWithAttributes
11880 = parsers.auto_link_url
11881 * Ct(parsers.attributes)
11882 / self.auto_link_url
11883
11884 self.insert_pattern("Inline before AutoLinkUrl",
11885 AutoLinkUrlWithAttributes,
11886 "AutoLinkUrlWithAttributes")
11887
11888 local AutoLinkEmailWithAttributes
11889 = parsers.auto_link_email
11890 * Ct(parsers.attributes)
11891 / self.auto_link_email
11892
11893 self.insert_pattern("Inline before AutoLinkEmail",
11894 AutoLinkEmailWithAttributes,
11895 "AutoLinkEmailWithAttributes")
11896
11897 if options.relativeReferences then
11898
11899 local AutoLinkRelativeReferenceWithAttributes
11900 = parsers.auto_link_relative_reference

```

```

11901 * Ct(parsers.attributes)
11902 / self.auto_link_url
11903
11904 self.insert_pattern(
11905 "Inline before AutoLinkRelativeReference",
11906 AutoLinkRelativeReferenceWithAttributes,
11907 "AutoLinkRelativeReferenceWithAttributes")
11908
11909 end
11910
11911 end
11912 }
11913 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.

```

11914 M.extensions.notes = function(notes, inline_notes)
11915 assert(notes or inline_notes)
11916 return {
11917 name = "built-in notes syntax extension",
11918 extend_writer = function(self)

```

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

```

11919 function self.note(s)
11920 if self.flatten_inlines then return "" end
11921 return {"\\markdownRendererNote{",s,""}
11922 end
11923 end, extend_reader = function(self)
11924 local parsers = self.parsers
11925 local writer = self.writer
11926
11927 local rawnotes = parsers.rawnotes
11928
11929 if inline_notes then
11930 local InlineNote
11931 = parsers.circumflex
11932 * (parsers.link_label
11933 / self.parser_functions.parse_inlines_no_inline_note)
11934 / writer.note
11935
11936 self.insert_pattern("Inline after LinkAndEmph",
11937 InlineNote, "InlineNote")

```

```

11938 end
11939 if notes then
11940 local function strip_first_char(s)
11941 return s:sub(2)
11942 end
11943
11944 local RawNoteRef
11945 = #(parsers.lbracket * parsers.circumflex)
11946 * parsers.link_label / strip_first_char
11947
11948 -- like indirect_link
11949 local function lookup_note(ref)
11950 return writer.defer_call(function()
11951 local found = rawnotes[self.normalize_tag(ref)]
11952 if found then
11953 return writer.note(
11954 self.parser_functions.parse_blocks_nested(found))
11955 else
11956 return {"[",
11957 self.parser_functions.parse_inlines("^" .. ref), "]" }
11958 end
11959 end)
11960 end
11961
11962 local function register_note(ref,rawnote)
11963 local normalized_tag = self.normalize_tag(ref)
11964 if rawnotes[normalized_tag] == nil then
11965 rawnotes[normalized_tag] = rawnote
11966 end
11967 return ""
11968 end
11969
11970 local NoteRef = RawNoteRef / lookup_note
11971
11972 local optionally_indented_line
11973 = parsers.check_optional_indent_and_any_trail * parsers.line
11974
11975 local blank
11976 = parsers.check_optional_blank_indent_and_any_trail
11977 * parsers.optionalspace * parsers.newline
11978
11979 local chunk
11980 = Cs(parsers.line
11981 * (optionally_indented_line - blank)^0)
11982
11983 local indented_blocks = function(bl)
11984 return Cs(bl

```

```

11985 * (blank^1 * (parsers.check_optional_indent / "")
11986 * parsers.check_code_trail
11987 * -parsers.blankline * bl)^0)
11988 end
11989
11990 local NoteBlock
11991 = parsers.check_trail_no_rem
11992 * RawNoteRef * parsers.colon
11993 * parsers.spnlc * indented_blocks(chunk)
11994 / register_note
11995
11996 self.update_rule("Reference", function(previous_pattern)
11997 if previous_pattern == nil then
11998 previous_pattern = parsers.Reference
11999 end
12000 return NoteBlock + previous_pattern
12001 end)
12002
12003 self.insert_pattern("Inline before LinkAndEmph",
12004 NoteRef, "NoteRef")
12005 end
12006
12007 self.add_special_character("^")
12008 end
12009 }
12010 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.

```

12011 M.extensions.pipe_tables = function(table_captions, table_attributes)
12012
12013 local function make_pipe_table_rectangular(rows)
12014 local num_columns = #rows[2]
12015 local rectangular_rows = {}
12016 for i = 1, #rows do
12017 local row = rows[i]
12018 local rectangular_row = {}
12019 for j = 1, num_columns do
12020 rectangular_row[j] = row[j] or ""
12021 end
12022 table.insert(rectangular_rows, rectangular_row)

```

```

12023 end
12024 return rectangular_rows
12025 end
12026
12027 local function pipe_table_row(allow_empty_first_column
12028 , nonempty_column
12029 , column_separator
12030 , column)
12031 local row_beginning
12032 if allow_empty_first_column then
12033 row_beginning = -- empty first column
12034 #(parsers.spacechar^4
12035 * column_separator)
12036 * parsers.optionalspace
12037 * column
12038 * parsers.optionalspace
12039 -- non-empty first column
12040 + parsers.nonindentspace
12041 * nonempty_column^-1
12042 * parsers.optionalspace
12043 else
12044 row_beginning = parsers.nonindentspace
12045 * nonempty_column^-1
12046 * parsers.optionalspace
12047 end
12048
12049 return Ct(row_beginning
12050 * (-- single column with no leading pipes
12051 #(column_separator
12052 * parsers.optionalspace
12053 * parsers.newline)
12054 * column_separator
12055 * parsers.optionalspace
12056 -- single column with leading pipes or
12057 -- more than a single column
12058 + (column_separator
12059 * parsers.optionalspace
12060 * column
12061 * parsers.optionalspace)^1
12062 * (column_separator
12063 * parsers.optionalspace)^-1))
12064 end
12065
12066 return {
12067 name = "built-in pipe_tables syntax extension",
12068 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.

```

12069 function self.table(rows, caption, attributes)
12070 if not self.is_writing then return "" end
12071 local buffer = {}
12072 if attributes ~= nil then
12073 table.insert(buffer,
12074 "\\markdownRendererTableAttributeContextBegin\n")
12075 table.insert(buffer, self.attributes(attributes))
12076 end
12077 table.insert(buffer,
12078 {"\\markdownRendererTable{",
12079 caption or "", "}{" , #rows - 1, "}{" ,
12080 #rows[1], "}")})
12081 local temp = rows[2] -- put alignments on the first row
12082 rows[2] = rows[1]
12083 rows[1] = temp
12084 for i, row in ipairs(rows) do
12085 table.insert(buffer, "{")
12086 for _, column in ipairs(row) do
12087 if i > 1 then -- do not use braces for alignments
12088 table.insert(buffer, "{")
12089 end
12090 table.insert(buffer, column)
12091 if i > 1 then
12092 table.insert(buffer, "}")
12093 end
12094 end
12095 table.insert(buffer, "}")
12096 end
12097 if attributes ~= nil then
12098 table.insert(buffer,
12099 "\\markdownRendererTableAttributeContextEnd{")
12100 end
12101 return buffer
12102 end
12103 end, extend_reader = function(self)
12104 local parsers = self.parsers
12105 local writer = self.writer
12106
12107 local table_hline_separator = parsers.pipe + parsers.plus
12108
12109 local table_hline_column = (parsers.dash
12110 - #(parsers.dash
12111 * (parsers.spacechar
12112 + table_hline_separator
12113 + parsers.newline)))^1

```

```

12114 * (parsers.colon * Cc("r")
12115 + parsers.dash * Cc("d"))
12116 + parsers.colon
12117 * (parsers.dash
12118 - #(parsers.dash
12119 * (parsers.spacechar
12120 + table_hline_separator
12121 + parsers.newline)))^1
12122 * (parsers.colon * Cc("c")
12123 + parsers.dash * Cc("l"))
12124
12125 local table_hline = pipe_table_row(false
12126 , table_hline_column
12127 , table_hline_separator
12128 , table_hline_column)
12129
12130 local table_caption_beginning
12131 = (parsers.check_minimal_blank_indent_and_any_trail_no_rem
12132 * parsers.optionalspace * parsers.newline)^0
12133 * parsers.check_minimal_indent_and_trail
12134 * (P("Table")^-1 * parsers.colon)
12135 * parsers.optionalspace
12136
12137 local function strip_trailing_spaces(s)
12138 return s:gsub("%s*$","")
12139 end
12140
12141 local table_row
12142 = pipe_table_row(true
12143 , (C((parsers.linechar - parsers.pipe)^1)
12144 / strip_trailing_spaces
12145 / self.parser_functions.parse_inlines)
12146 , parsers.pipe
12147 , (C((parsers.linechar - parsers.pipe)^0)
12148 / strip_trailing_spaces
12149 / self.parser_functions.parse_inlines))
12150
12151 local table_caption
12152 if table_captions then
12153 table_caption = #table_caption_beginning
12154 * table_caption_beginning
12155 if table_attributes then
12156 table_caption = table_caption
12157 * (C(((parsers.linechar
12158 - (parsers.attributes
12159 * parsers.optionalspace
12160 * parsers.newline

```



```

12161 * -(parsers.optionalspace
12162 * parsers.linechar)))
12163 + (parsers.newline
12164 * #(parsers.optionalspace
12165 * parsers.linechar)
12166 * C(parsers.optionalspace)
12167 / writer.space))
12168 * (parsers.linechar
12169 - parsers.lbrace)^0^1)
12170 / self.parser_functions.parse_inlines)
12171 * (parsers.newline
12172 + (Ct(parsers.attributes)
12173 * parsers.optionalspace
12174 * parsers.newline))
12175 else
12176 table_caption = table_caption
12177 * C((parsers.linechar
12178 + (parsers.newline
12179 * #(parsers.optionalspace
12180 * parsers.linechar)
12181 * C(parsers.optionalspace)
12182 / writer.space))^1)
12183 / self.parser_functions.parse_inlines
12184 * parsers.newline
12185 end
12186 else
12187 table_caption = parsers.fail
12188 end
12189
12190 local PipeTable
12191 = Ct(table_row * parsers.newline
12192 * (parsers.check_minimal_indent_and_trail / {})
12193 * table_hline * parsers.newline
12194 * ((parsers.check_minimal_indent / {})
12195 * table_row * parsers.newline)^0)
12196 / make_pipe_table_rectangular
12197 * table_caption^-1
12198 / writer.table
12199
12200 self.insert_pattern("Block after Blockquote",
12201 PipeTable, "PipeTable")
12202 end
12203 }
12204 end

```

### 3.1.7.15 Raw Attributes

The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```
12205 M.extensions.raw_inline = function()
12206 return {
12207 name = "built-in raw_inline syntax extension",
12208 extend_writer = function(self)
12209 local options = self.options
12210
```

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

```
12211 function self.rawInline(s, attr)
12212 if not self.is_writing then return "" end
12213 if self.flatten_inlines then return s end
12214 local name = util.cache_verbatim(options.cacheDir, s)
12215 return {"\\markdownRendererInputRawInline{" ,
12216 name,"}{" , self.string(attr),"}"}
12217 end
12218 end, extend_reader = function(self)
12219 local writer = self.writer
12220
12221 local RawInline = parsers.inticks
12222 * parsers.raw_attribute
12223 / writer.rawInline
12224
12225 self.insert_pattern("Inline before Code",
12226 RawInline, "RawInline")
12227 end
12228 }
12229 end
```

### 3.1.7.16 Strike-Through

The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```
12230 M.extensions.strike_through = function()
12231 return {
12232 name = "built-in strike_through syntax extension",
12233 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.

```
12234 function self.strike_through(s)
12235 if self.flatten_inlines then return s end
12236 return {"\\markdownRendererStrikeThrough{" ,s,"}"}
12237 end
12238 end, extend_reader = function(self)
```

```

12239 local parsers = self.parsers
12240 local writer = self.writer
12241
12242 local StrikeThrough = (
12243 parsers.between(parsers.Inline, parsers.doubletildes,
12244 parsers.doubletildes)
12245) / writer.strike_through
12246
12247 self.insert_pattern("Inline after LinkAndEmph",
12248 StrikeThrough, "StrikeThrough")
12249
12250 self.add_special_character("~")
12251 end
12252 }
12253 end

```

### 3.1.7.17 Subscripts

The `extensions.subscripts` function implements the Pandoc subscript syntax extension.

```

12254 M.extensions.subscripts = function()
12255 return {
12256 name = "built-in subscripts syntax extension",
12257 extend_writer = function(self)

```

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

```

12258 function self.subscript(s)
12259 if self.flatten_inlines then return s end
12260 return {"\\markdownRendererSubscript{" ,s, "}"}
12261 end
12262 end, extend_reader = function(self)
12263 local parsers = self.parsers
12264 local writer = self.writer
12265
12266 local Subscript = (
12267 parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
12268) / writer.subscript
12269
12270 self.insert_pattern("Inline after LinkAndEmph",
12271 Subscript, "Subscript")
12272
12273 self.add_special_character("~")
12274 end
12275 }
12276 end

```

### 3.1.7.18 Superscripts

The `extensions.superscripts` function implements the Pandoc superscript syntax extension.

```
12277 M.extensions.superscripts = function()
12278 return {
12279 name = "built-in superscripts syntax extension",
12280 extend_writer = function(self)
```

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

```
12281 function self.superscript(s)
12282 if self.flatten_inlines then return s end
12283 return {"\\markdownRendererSuperscript{" ,s,"}"}
12284 end
12285 end, extend_reader = function(self)
12286 local parsers = self.parsers
12287 local writer = self.writer
12288
12289 local Superscript = (
12290 parsers.between(parsers.Str, parsers.circumflex,
12291 parsers.circumflex)
12292) / writer.superscript
12293
12294 self.insert_pattern("Inline after LinkAndEmph",
12295 Superscript, "Superscript")
12296
12297 self.add_special_character("^")
12298 end
12299 }
12300 end
```

### 3.1.7.19 TeX Math

The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```
12301 M.extensions.tex_math = function(tex_math_dollars,
12302 tex_math_single_backslash,
12303 tex_math_double_backslash)
12304 return {
12305 name = "built-in tex_math syntax extension",
12306 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.

```
12307 function self.display_math(s)
12308 if self.flatten_inlines then return s end
12309 return {"\\markdownRendererDisplayMath{" ,self.math(s),"}"}

```

12310           end

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

```
12311 function self.inline_math(s)
12312 if self.flatten_inlines then return s end
12313 return {"\\markdownRendererInlineMath{" ,self.math(s),"}"}
12314 end
12315 end, extend_reader = function(self)
12316 local parsers = self.parsers
12317 local writer = self.writer
12318
12319 local function between(p, starter, ender)
12320 return (starter * Cs(p * (p - ender)^0) * ender)
12321 end
12322
12323 local function strip_preceding_whitespaces(str)
12324 return str:gsub("^%s*(.)$", "%1")
12325 end
12326
12327 local allowed_before_closing
12328 = B(parsers.backslash * parsers.any
12329 + parsers.any * (parsers.any - parsers.backslash))
12330
12331 local allowed_before_closing_no_space
12332 = B(parsers.backslash * parsers.any
12333 + parsers.any * (parsers.nonspacechar - parsers.backslash))
12334
```

The following patterns implement the Pandoc dollar math syntax extension.

```
12335 local dollar_math_content
12336 = (parsers.newline * (parsers.check_optional_indent / "")
12337 + parsers.backslash^-1
12338 * parsers.linechar)
12339 - parsers.blankline^2
12340 - parsers.dollar
12341
12342 local inline_math_opening_dollars = parsers.dollar
12343 * #(parsers.nonspacechar)
12344
12345 local inline_math_closing_dollars
12346 = allowed_before_closing_no_space
12347 * parsers.dollar
12348 * -#(parsers.digit)
12349
12350 local inline_math_dollars = between(Cs(dollar_math_content),
12351 inline_math_opening_dollars,
12352 inline_math_closing_dollars)
```

```

12353
12354 local display_math_opening_dollars = parsers.dollar
12355 * parsers.dollar
12356
12357 local display_math_closing_dollars = parsers.dollar
12358 * parsers.dollar
12359
12360 local display_math_dollars = between(Cs(dollar_math_content),
12361 display_math_opening_dollars,
12362 display_math_closing_dollars)

```

The following patterns implement the Pandoc single and double backslash math syntax extensions.

```

12363 local backslash_math_content
12364 = (parsers.newline * (parsers.check_optional_indent / ""))
12365 + parsers.linechar)
12366 - parsers.blankline^2

```

The following patterns implement the Pandoc double backslash math syntax extension.

```

12367 local inline_math_opening_double = parsers.backslash
12368 * parsers.backslash
12369 * parsers.lparent
12370
12371 local inline_math_closing_double = allowed_before_closing
12372 * parsers.spacechar^0
12373 * parsers.backslash
12374 * parsers.backslash
12375 * parsers.rparent
12376
12377 local inline_math_double = between(Cs(backslash_math_content),
12378 inline_math_opening_double,
12379 inline_math_closing_double)
12380 / strip_preceding_whitespaces
12381
12382 local display_math_opening_double = parsers.backslash
12383 * parsers.backslash
12384 * parsers.lbracket
12385
12386 local display_math_closing_double = allowed_before_closing
12387 * parsers.spacechar^0
12388 * parsers.backslash
12389 * parsers.backslash
12390 * parsers.rbracket
12391
12392 local display_math_double = between(Cs(backslash_math_content),
12393 display_math_opening_double,
12394 display_math_closing_double)

```

```

12395 / strip_preceding_whitespaces
The following patterns implement the Pandoc single backslash math syntax extension.
12396 local inline_math_opening_single = parsers.backslash
12397 * parsers.lparent
12398
12399 local inline_math_closing_single = allowed_before_closing
12400 * parsers.spacechar^0
12401 * parsers.backslash
12402 * parsers.rparent
12403
12404 local inline_math_single = between(Cs(backslash_math_content),
12405 inline_math_opening_single,
12406 inline_math_closing_single)
12407 / strip_preceding_whitespaces
12408
12409 local display_math_opening_single = parsers.backslash
12410 * parsers.lbracket
12411
12412 local display_math_closing_single = allowed_before_closing
12413 * parsers.spacechar^0
12414 * parsers.backslash
12415 * parsers.rbracket
12416
12417 local display_math_single = between(Cs(backslash_math_content),
12418 display_math_opening_single,
12419 display_math_closing_single)
12420 / strip_preceding_whitespaces
12421
12422 local display_math = parsers.fail
12423
12424 local inline_math = parsers.fail
12425
12426 if tex_math_dollars then
12427 display_math = display_math + display_math_dollars
12428 inline_math = inline_math + inline_math_dollars
12429 end
12430
12431 if tex_math_double_backslash then
12432 display_math = display_math + display_math_double
12433 inline_math = inline_math + inline_math_double
12434 end
12435
12436 if tex_math_single_backslash then
12437 display_math = display_math + display_math_single
12438 inline_math = inline_math + inline_math_single
12439 end
12440

```

```

12441 local TexMath = display_math / writer.display_math
12442 + inline_math / writer.inline_math
12443
12444 self.insert_pattern("Inline after LinkAndEmph",
12445 TexMath, "TexMath")
12446
12447 if tex_math_dollars then
12448 self.add_special_character("$")
12449 end
12450
12451 if tex_math_single_backslash or tex_math_double_backslash then
12452 self.add_special_character("\\")
12453 self.add_special_character("[")
12454 self.add_special_character("]")
12455 self.add_special_character("(")
12456 self.add_special_character("(")
12457 end
12458 end
12459 }
12460 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.

```

12461 M.extensions.jekyll_data = function(expect_jekyll_data,
12462 ensure_jekyll_data)
12463 return {
12464 name = "built-in jekyll_data syntax extension",
12465 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.

```

12466 function self.jekyllData(d, t, p)
12467 if not self.is_writing then return "" end
12468
12469 local buf = {}
12470

```



```

12471 local keys = {}
12472 for k, _ in pairs(d) do
12473 table.insert(keys, k)
12474 end

```

For reproducibility, sort the keys. For mixed string-and-numeric keys, sort numeric keys before string keys.

```

12475 table.sort(keys, function(first, second)
12476 if type(first) ~= type(second) then
12477 return type(first) < type(second)
12478 else
12479 return first < second
12480 end
12481 end)
12482
12483 if not p then
12484 table.insert(buf, "\\markdownRendererJekyllDataBegin")
12485 end
12486
12487 local is_sequence = false
12488 if #d > 0 and #d == #keys then
12489 for i=1, #d do
12490 if d[i] == nil then
12491 goto not_a_sequence
12492 end
12493 end
12494 is_sequence = true
12495 end
12496 ::not_a_sequence::
12497
12498 if is_sequence then
12499 table.insert(buf,
12500 "\\markdownRendererJekyllDataSequenceBegin{")
12501 table.insert(buf, self.identifier(p or "null"))
12502 table.insert(buf, "{")
12503 table.insert(buf, #keys)
12504 table.insert(buf, "}")
12505 else
12506 table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
12507 table.insert(buf, self.identifier(p or "null"))
12508 table.insert(buf, "{")
12509 table.insert(buf, #keys)
12510 table.insert(buf, "}")
12511 end
12512
12513 for _, k in ipairs(keys) do
12514 local v = d[k]

```

```

12515 local typ = type(v)
12516 k = tostring(k or "null")
12517 if typ == "table" and next(v) ~= nil then
12518 table.insert(
12519 buf,
12520 self.jekyllData(v, t, k)
12521)
12522 else
12523 k = self.identifier(k)
12524 v = tostring(v)
12525 if typ == "boolean" then
12526 table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
12527 table.insert(buf, k)
12528 table.insert(buf, "}{")
12529 table.insert(buf, v)
12530 table.insert(buf, "}")
12531 elseif typ == "number" then
12532 table.insert(buf, "\\markdownRendererJekyllDataNumber{")
12533 table.insert(buf, k)
12534 table.insert(buf, "}{")
12535 table.insert(buf, v)
12536 table.insert(buf, "}")
12537 elseif typ == "string" then
12538 table.insert(buf,
12539 "\\markdownRendererJekyllDataProgrammaticString{")
12540 table.insert(buf, k)
12541 table.insert(buf, "}{")
12542 table.insert(buf, self.identifier(v))
12543 table.insert(buf, "}")
12544 table.insert(buf,
12545 "\\markdownRendererJekyllDataTypographicString{")
12546 table.insert(buf, k)
12547 table.insert(buf, "}{")
12548 table.insert(buf, t(v))
12549 table.insert(buf, "}")
12550 elseif typ == "table" then
12551 table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
12552 table.insert(buf, k)
12553 table.insert(buf, "}")
12554 else
12555 local error = self.error(format(
12556 "Unexpected type %s for value of "
12557 .. "YAML key %s.", typ, k))
12558 table.insert(buf, error)
12559 end
12560 end
12561 end

```

```

12562
12563 if is_sequence then
12564 table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
12565 else
12566 table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
12567 end
12568
12569 if not p then
12570 table.insert(buf, "\\markdownRendererJekyllDataEnd")
12571 end
12572
12573 return buf
12574 end
12575 end, extend_reader = function(self)
12576 local parsers = self.parsers
12577 local writer = self.writer
12578
12579 local JekyllData
12580 = Cmt(C((parsers.line - P("---") - P("..."))^0)
12581 , function(s, i, text) -- luacheck: ignore s i
12582 local data
12583 local ran_ok, _ = pcall(function()
12584 local tinyyaml = require("tinyyaml")
12585 data = tinyyaml.parse(text, {timestamps=false})
12586 end)
12587 if ran_ok and data ~= nil then
12588 return true, writer.jekyllData(data, function(s)
12589 return self.parser_functions.parse_blocks_nested(s)
12590 end, nil)
12591 else
12592 return false
12593 end
12594 end
12595)
12596
12597 local UnexpectedJekyllData
12598 = P("---")
12599 * parsers.blankline / 0
12600 -- if followed by blank, it's thematic break
12601 * #(-parsers.blankline)
12602 * JekyllData
12603 * (P("---") + P("..."))
12604
12605 local ExpectedJekyllData
12606 = (P("---")
12607 * parsers.blankline / 0
12608 -- if followed by blank, it's thematic break

```

```

12609 * #(-parsers.blankline)
12610)^-1
12611 * JekyllData
12612 * (P("----") + P("..."))^-1
12613
12614 if ensure_jekyll_data then
12615 ExpectedJekyllData = ExpectedJekyllData
12616 * parsers.eof
12617 else
12618 ExpectedJekyllData = (ExpectedJekyllData
12619 * (V("Blank")^0 / writer.interblocksep)
12620)^-1
12621 end
12622
12623 self.insert_pattern("Block before Blockquote",
12624 UnexpectedJekyllData, "UnexpectedJekyllData")
12625 if expect_jekyll_data then
12626 self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
12627 end
12628 end
12629 }
12630 end

```

### 3.1.8 Conversion from Markdown to Plain TeX

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`.

```

12631 function M.new(options)

```

Make the `options` table inherit from the `defaultOptions` table.

```

12632 options = options or {}
12633 setmetatable(options, { __index = function (_, key)
12634 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.

```

12635 local parser_convert = nil
12636 return function(input, include_flat_output)
12637 local function convert(input)
12638 if parser_convert == nil then

```

Lazy-load `markdown-parser.lua` and check that it originates from the same version of the Markdown package.

```

12639 local parser = require("markdown-parser")

```

```

12640 if metadata.version ~= parser.metadata.version then
12641 warn("markdown.lua " .. metadata.version .. " used with " ..
12642 "markdown-parser.lua " .. parser.metadata.version .. ".")
12643 end
12644 parser_convert = parser.new(options)
12645 end
12646 return parser_convert(input)
12647 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).

```

12648 local raw_output, flat_output
12649 if options.eagerCache or options.finalizeCache then
12650 local salt = util.salt(options)
12651 local name, result = util.cache(options.cacheDir, input, salt,
12652 convert, ".md.tex")
12653 raw_output = [[\input{}}] .. name .. [[\relax]]
12654 flat_output = function()
12655 if result == nil then
12656 local input_file = assert(io.open(name, "r"),
12657 [[Could not open file "]] .. name .. [[for reading]])
12658 result = assert(input_file:read("*a"))
12659 assert(input_file:close())
12660 end
12661 return result
12662 end

```

Otherwise, return the result of the conversion directly.

```

12663 else
12664 raw_output = convert(input)
12665 flat_output = function()
12666 return raw_output
12667 end
12668 end

```

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

```

12669 if options.finalizeCache then
12670 local file, mode
12671 if options.frozenCacheCounter > 0 then
12672 mode = "a"
12673 else
12674 mode = "w"
12675 end

```

```

12676 file = assert(io.open(options.frozenCacheFileName, mode),
12677 [[Could not open file "]] .. options.frozenCacheFileName
12678 .. [[" for writing]])
12679 assert(file:write(
12680 [[\expandafter\global\expandafter\def\csname]]
12681 .. [[markdownFrozenCache]] .. options.frozenCacheCounter
12682 .. [[\endcsname{}}] .. raw_output .. [{}]] .. "\n"))
12683 assert(file:close())
12684 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.

```

12685 if include_flat_output then
12686 return raw_output, flat_output
12687 else
12688 return raw_output
12689 end
12690 end
12691 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.

```

12692 function M.new(options)

```

Make the `options` table inherit from the `defaultOptions` table.

```

12693 options = options or {}
12694 setmetatable(options, { __index = function (_, key)
12695 return defaultOptions[key] end })

```

If the singleton cache contains a conversion function for the same `options`, reuse it.

```

12696 if options.singletonCache and singletonCache.convert then
12697 for k, v in pairs(defaultOptions) do
12698 if type(v) == "table" then
12699 for i = 1, math.max(#singletonCache.options[k], #options[k]) do
12700 if singletonCache.options[k][i] ~= options[k][i] then
12701 goto miss
12702 end
12703 end

```

The `cacheDir` option is disregarded.

```

12704 elseif k ~= "cacheDir"
12705 and singletonCache.options[k] ~= options[k] then
12706 goto miss
12707 end
12708 end
12709 return singletonCache.convert
12710 end
12711 ::miss::

```

Apply built-in syntax extensions based on `options`.

```
12712 local extensions = {}
12713
12714 if options.bracketedSpans then
12715 local bracketed_spans_extension = M.extensions.bracketed_spans()
12716 table.insert(extensions, bracketed_spans_extension)
12717 end
12718
12719 if options.contentBlocks then
12720 local content_blocks_extension = M.extensions.content_blocks(
12721 options.contentBlocksLanguageMap)
12722 table.insert(extensions, content_blocks_extension)
12723 end
12724
12725 if options.definitionLists then
12726 local definition_lists_extension = M.extensions.definition_lists(
12727 options.tightLists)
12728 table.insert(extensions, definition_lists_extension)
12729 end
12730
12731 if options.fencedCode then
12732 local fenced_code_extension = M.extensions.fenced_code(
12733 options.blankBeforeCodeFence,
12734 options.fencedCodeAttributes,
12735 options.rawAttribute)
12736 table.insert(extensions, fenced_code_extension)
12737 end
12738
12739 if options.fencedDivs then
12740 local fenced_div_extension = M.extensions.fenced_divs(
12741 options.blankBeforeDivFence)
12742 table.insert(extensions, fenced_div_extension)
12743 end
12744
12745 if options.headerAttributes then
12746 local header_attributes_extension = M.extensions.header_attributes()
12747 table.insert(extensions, header_attributes_extension)
12748 end
12749
12750 if options.inlineCodeAttributes then
12751 local inline_code_attributes_extension =
12752 M.extensions.inline_code_attributes()
12753 table.insert(extensions, inline_code_attributes_extension)
12754 end
12755
12756 if options.jekyllData then
12757 local jekyll_data_extension = M.extensions.jekyll_data(
```

```

12758 options.expectJekyllData, options.ensureJekyllData)
12759 table.insert(extensions, jekyll_data_extension)
12760 end
12761
12762 if options.linkAttributes then
12763 local link_attributes_extension =
12764 M.extensions.link_attributes()
12765 table.insert(extensions, link_attributes_extension)
12766 end
12767
12768 if options.lineBlocks then
12769 local line_block_extension = M.extensions.line_blocks()
12770 table.insert(extensions, line_block_extension)
12771 end
12772
12773 if options.mark then
12774 local mark_extension = M.extensions.mark()
12775 table.insert(extensions, mark_extension)
12776 end
12777
12778 if options.pipeTables then
12779 local pipe_tables_extension = M.extensions.pipe_tables(
12780 options.tableCaptions, options.tableAttributes)
12781 table.insert(extensions, pipe_tables_extension)
12782 end
12783
12784 if options.rawAttribute then
12785 local raw_inline_extension = M.extensions.raw_inline()
12786 table.insert(extensions, raw_inline_extension)
12787 end
12788
12789 if options.strikeThrough then
12790 local strike_through_extension = M.extensions.strike_through()
12791 table.insert(extensions, strike_through_extension)
12792 end
12793
12794 if options.subscripts then
12795 local subscript_extension = M.extensions.subscripts()
12796 table.insert(extensions, subscript_extension)
12797 end
12798
12799 if options.superscripts then
12800 local superscript_extension = M.extensions.superscripts()
12801 table.insert(extensions, superscript_extension)
12802 end
12803
12804 if options.texMathDollars or

```



```

12805 options.texMathSingleBackslash or
12806 options.texMathDoubleBackslash then
12807 local tex_math_extension = M.extensions.tex_math(
12808 options.texMathDollars,
12809 options.texMathSingleBackslash,
12810 options.texMathDoubleBackslash)
12811 table.insert(extensions, tex_math_extension)
12812 end
12813
12814 if options.notes or options.inlineNotes then
12815 local notes_extension = M.extensions.notes(
12816 options.notes, options.inlineNotes)
12817 table.insert(extensions, notes_extension)
12818 end
12819
12820 if options.citations then
12821 local citations_extension
12822 = M.extensions.citations(options.citationNbsps)
12823 table.insert(extensions, citations_extension)
12824 end
12825
12826 if options.fancyLists then
12827 local fancy_lists_extension = M.extensions.fancy_lists()
12828 table.insert(extensions, fancy_lists_extension)
12829 end

```

Apply user-defined syntax extensions based on `options.extensions`.

```

12830 for _, user_extension_filename in ipairs(options.extensions) do
12831 local user_extension = (function(filename)

```

First, load and compile the contents of the user-defined syntax extension.

```

12832 local pathname = assert(kpse.find_file(filename),
12833 [[Could not locate user-defined syntax extension "]]
12834 .. filename)
12835 local input_file = assert(io.open(pathname, "r"),
12836 [[Could not open user-defined syntax extension "]]
12837 .. pathname .. [{" for reading}]]
12838 local input = assert(input_file:read("*a"))
12839 assert(input_file:close())
12840 local user_extension, err = load([[
12841 local sandbox = {}
12842 setmetatable(sandbox, {__index = _G})
12843 _ENV = sandbox
12844]] .. input)()
12845 assert(user_extension,
12846 [[Failed to compile user-defined syntax extension "]]
12847 .. pathname .. [{": }]] .. (err or [{"}]))

```

Then, validate the user-defined syntax extension.

```
12848 assert(user_extension.api_version ~= nil,
12849 [[User-defined syntax extension]] .. pathname
12850 .. [[" does not specify mandatory field "api_version"]])
12851 assert(type(user_extension.api_version) == "number",
12852 [[User-defined syntax extension]] .. pathname
12853 .. [[" specifies field "api_version" of type "]]
12854 .. type(user_extension.api_version)
12855 .. [[" but "number" was expected]])
12856 assert(user_extension.api_version > 0
12857 and user_extension.api_version
12858 <= metadata.user_extension_api_version,
12859 [[User-defined syntax extension]] .. pathname
12860 .. [[" uses syntax extension API version "]]
12861 .. user_extension.api_version .. [[but markdown.lua]]
12862 .. metadata.version .. [[uses API version]]
12863 .. metadata.user_extension_api_version
12864 .. [[, which is incompatible]])
12865
12866 assert(user_extension.grammar_version ~= nil,
12867 [[User-defined syntax extension]] .. pathname
12868 .. [[" does not specify mandatory field "grammar_version"]])
12869 assert(type(user_extension.grammar_version) == "number",
12870 [[User-defined syntax extension]] .. pathname
12871 .. [[" specifies field "grammar_version" of type "]]
12872 .. type(user_extension.grammar_version)
12873 .. [[" but "number" was expected]])
12874 assert(user_extension.grammar_version == metadata.grammar_version,
12875 [[User-defined syntax extension]] .. pathname
12876 .. [[" uses grammar version "]]
12877 .. user_extension.grammar_version
12878 .. [[but markdown.lua]] .. metadata.version
12879 .. [[uses grammar version]] .. metadata.grammar_version
12880 .. [[, which is incompatible]])
12881
12882 assert(user_extension.finalize_grammar ~= nil,
12883 [[User-defined syntax extension]] .. pathname
12884 .. [[" does not specify mandatory "finalize_grammar" field]])
12885 assert(type(user_extension.finalize_grammar) == "function",
12886 [[User-defined syntax extension]] .. pathname
12887 .. [[" specifies field "finalize_grammar" of type "]]
12888 .. type(user_extension.finalize_grammar)
12889 .. [[" 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.)

```
12890 local extension = {
```

```

12891 name = [[user-defined "]] .. pathname .. [{" syntax extension}],
12892 extend_reader = user_extension.finalize_grammar,
12893 extend_writer = function() end,
12894 }
12895 return extension
12896 end)(user_extension_filename)
12897 table.insert(extensions, user_extension)
12898 end

```

Produce a conversion function from markdown to plain  $\TeX$ .

```

12899 local writer = M.writer.new(options)
12900 local reader = M.reader.new(writer, options)
12901 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`.

```

12902 collectgarbage("collect")

```

Update the singleton cache.

```

12903 if options.singletonCache then
12904 local singletonCacheOptions = {}
12905 for k, v in pairs(options) do
12906 singletonCacheOptions[k] = v
12907 end
12908 setmetatable(singletonCacheOptions,
12909 { __index = function (_, key)
12910 return defaultOptions[key] end })
12911 singletonCache.options = singletonCacheOptions
12912 singletonCache.convert = convert
12913 end

```

Return the conversion function from markdown to plain  $\TeX$ .

```

12914 return convert
12915 end
12916 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.

```

12917
12918 local input
12919 if input_filename then
12920 local input_file = assert(io.open(input_filename, "r"),
12921 [[Could not open file]] .. input_filename .. [{" for reading}])
12922 input = assert(input_file:read("*a"))
12923 assert(input_file:close())
12924 else

```

```

12925 input = assert(io.read("*a"))
12926 end
12927

```

First, ensure that the `options.cacheDir` directory exists.

```

12928 local lfs = require("lfs")
12929 if options.cacheDir and not lfs.isdir(options.cacheDir) then
12930 assert(lfs.mkdir(options["cacheDir"]))
12931 end

```

If Kpathsea has not been loaded before or if LuaTeX has not yet been initialized, configure Kpathsea on top of loading it.

```

12932 local kpse
12933 (function()
12934 local should_initialize = package.loaded.kpse == nil
12935 or tex.initialize ~= nil
12936 kpse = require("kpse")
12937 if should_initialize then
12938 kpse.set_program_name("luatex")
12939 end
12940 end)()
12941 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.

```

12942 if metadata.version ~= md.metadata.version then
12943 warn("markdown-cli.lua " .. metadata.version .. " used with " ..
12944 "markdown.lua " .. md.metadata.version .. ".")
12945 end
12946
12947 local convert = md.new(options)
12948 local raw_output, flat_output = convert(input, true)
12949 local output
12950 if flat_output == nil then
12951 if options.eagerCache then
12952 warn("markdown.lua has not produced flat output, so I am using " ..
12953 "backwards-compatible raw output instead. This may cause " ..
12954 "the conversion result to be hidden behind "\\input'.")
12955 end
12956 output = raw_output
12957 else
12958 output = flat_output()
12959 end
12960
12961 if output_filename then
12962 local output_file = assert(io.open(output_filename, "w"),
12963 [[Could not open file]] .. output_filename .. [[for writing]])
12964 assert(output_file:write(output))

```

```

12965 assert(output_file:close())
12966 else
12967 assert(io.write(output))
12968 end

```

Remove the `options.cacheDir` directory if it is empty.

```

12969 if options.cacheDir then
12970 lfs.rmdir(options.cacheDir)
12971 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

```

12972 \ExplSyntaxOn
12973 \cs_if_free:NT
12974 \markdownInfo
12975 {
12976 \cs_new:Npn
12977 \markdownInfo #1
12978 {
12979 \msg_info:nne
12980 { markdown }
12981 { generic-message }
12982 { #1 }
12983 }
12984 }
12985 \cs_if_free:NT
12986 \markdownWarning
12987 {
12988 \cs_new:Npn
12989 \markdownWarning #1
12990 {
12991 \msg_warning:nne
12992 { markdown }
12993 { generic-message }
12994 { #1 }
12995 }
12996 }
12997 \cs_if_free:NT
12998 \markdownError
12999 {

```

```

13000 \cs_new:Npn
13001 \markdownError #1 #2
13002 {
13003 \msg_error:nnee
13004 { markdown }
13005 { generic-message-with-help-text }
13006 { #1 }
13007 { #2 }
13008 }
13009 }
13010 \msg_new:nnn
13011 { markdown }
13012 { generic-message }
13013 { #1 }
13014 \msg_new:nnnn
13015 { markdown }
13016 { generic-message-with-help-text }
13017 { #1 }
13018 { #2 }
13019 \cs_generate_variant:Nn
13020 \msg_info:nnn
13021 { nne }
13022 \cs_generate_variant:Nn
13023 \msg_warning:nnn
13024 { nne }
13025 \cs_generate_variant:Nn
13026 \msg_error:nnnn
13027 { nnee }
13028 \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.

```

13029 \ExplSyntaxOn
13030 \prop_new:N \g_@@_plain_tex_loaded_themes_linenos_prop
13031 \prop_new:N \g_@@_plain_tex_loaded_themes_versions_prop
13032 \cs_new:Nn
13033 \@@_plain_tex_load_theme:nnn
13034 {
13035 \prop_get:NnNTF
13036 \g_@@_plain_tex_loaded_themes_linenos_prop
13037 { #1 }
13038 \l_tmpa_tl
13039 {
13040 \prop_get:NnN

```

```

13041 \g_@@_plain_tex_loaded_themes_versions_prop
13042 { #1 }
13043 \l_tmpb_tl
13044 \str_if_eq:nVTF
13045 { #2 }
13046 \l_tmpb_tl
13047 {
13048 \msg_warning:nnnVn
13049 { markdown }
13050 { repeatedly-loaded-plain-tex-theme }
13051 { #1 }
13052 \l_tmpa_tl
13053 { #2 }
13054 }
13055 {
13056 \msg_error:nnnnVV
13057 { markdown }
13058 { different-versions-of-plain-tex-theme }
13059 { #1 }
13060 { #2 }
13061 \l_tmpb_tl
13062 \l_tmpa_tl
13063 }
13064 }
13065 {
13066 \prop_gput:Nnx
13067 \g_@@_plain_tex_loaded_themes_linenos_prop
13068 { #1 }
13069 { \tex_the:D \tex_inputlineno:D } % noqa: W200
13070 \prop_gput:Nnn
13071 \g_@@_plain_tex_loaded_themes_versions_prop
13072 { #1 }
13073 { #2 }

```

Load built-in plain T<sub>E</sub>X themes from the prop `\g_@@_plain_tex_built_in_themes_prop` and from the filesystem otherwise.

```

13074 \prop_if_in:NnTF
13075 \g_@@_plain_tex_built_in_themes_prop
13076 { #1 }
13077 {
13078 \msg_info:nnnn
13079 { markdown }
13080 { loading-built-in-plain-tex-theme }
13081 { #1 }
13082 { #2 }
13083 }
13084 \prop_item:Nn
13085 \g_@@_plain_tex_built_in_themes_prop

```

```

13085 { #1 }
13086 }
13087 {
13088 \msg_info:nnnn
13089 { markdown }
13090 { loading-plain-tex-theme }
13091 { #1 }
13092 { #2 }
13093 \file_input:n
13094 { markdown theme #3 }
13095 }
13096 }
13097 }
13098 \msg_new:nnn
13099 { markdown }
13100 { loading-plain-tex-theme }
13101 { Loading~version~#2~of~plain~TeX~Markdown~theme~#1 }
13102 \msg_new:nnn
13103 { markdown }
13104 { loading-built-in-plain-tex-theme }
13105 { Loading~version~#2~of~built-in~plain~TeX~Markdown~theme~#1 }
13106 \msg_new:nnn
13107 { markdown }
13108 { repeatedly-loaded-plain-tex-theme }
13109 {
13110 Version~#3~of~plain~TeX~Markdown~theme~#1~was~previously~
13111 loaded~on~line~#2,~not~loading~it~again
13112 }
13113 \msg_new:nnn
13114 { markdown }
13115 { different-versions-of-plain-tex-theme }
13116 {
13117 Tried~to~load~version~#2~of~plain~TeX~Markdown~theme~#1~
13118 but~version~#3~has~already~been~loaded~on~line~#4
13119 }
13120 \cs_generate_variant:Nn
13121 \prop_gput:Nnn
13122 { Nnx }
13123 \cs_gset_eq:NN
13124 \@@_load_theme:nnn
13125 \@@_plain_tex_load_theme:nnn
13126 \cs_generate_variant:Nn
13127 \@@_load_theme:nnn
13128 { VeV }
13129 \cs_generate_variant:Nn
13130 \msg_error:nnnnnn
13131 { nnnnVV }

```



```

13132 \cs_generate_variant:Nn
13133 \msg_warning:nnnnn
13134 { nnnVn }

```

Developers can use the `\markdownLoadPlainTeXTheme` macro to load a corresponding plain TeX theme from within themes for higher-level TeX formats such as L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt.

```

13135 \cs_new:Npn
13136 \markdownLoadPlainTeXTheme
13137 {

```

First, we extract the name of the current theme from the `\g_@@_current_theme_tl` macro.

```

13138 \tl_set:NV
13139 \l_tmpa_tl
13140 \g_@@_current_theme_tl
13141 \tl_reverse:N
13142 \l_tmpa_tl
13143 \tl_set:Ne
13144 \l_tmpb_tl
13145 {
13146 \tl_tail:V
13147 \l_tmpa_tl
13148 }
13149 \tl_reverse:N
13150 \l_tmpb_tl

```

Next, we munge the theme name.

```

13151 \str_set:NV
13152 \l_tmpa_str
13153 \l_tmpb_tl
13154 \str_replace_all:Nnn
13155 \l_tmpa_str
13156 { / }
13157 { _ }

```

Finally, we load the plain TeX theme.

```

13158 \@@_plain_tex_load_theme:VeV
13159 \l_tmpb_tl
13160 { \markdownThemeVersion }
13161 \l_tmpa_str
13162 }
13163 \cs_generate_variant:Nn
13164 \tl_set:Nn
13165 { Ne }
13166 \cs_generate_variant:Nn
13167 \@@_plain_tex_load_theme:nnn
13168 { VeV }

```

The `witiko/dot` theme nags users that they should use the name `witiko/diagrams@v1` instead.

```

13169 \prop_gput:Nnn
13170 \g_@@_plain_tex_built_in_themes_prop
13171 { witiko / dot }
13172 {
13173 \str_if_eq:enF
13174 { \markdownThemeVersion }
13175 { silent }
13176 {
13177 \markdownWarning
13178 {
13179 The~theme~name~"witiko/dot"~has~been~soft-deprecated.
13180 \iow_newline:
13181 Consider~changing~the~name~to~"witiko/diagrams@v1".
13182 }
13183 }

```

We enable the `fencedCode` Lua option.

```

13184 \markdownSetup { fencedCode }

```

We store the previous definition of the fenced code token renderer prototype:

```

13185 \cs_set_eq:NN
13186 \@@_dot_previous_definition:nnn
13187 \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:

```

13188 \regex_const:Nn
13189 \c_@@_dot_infostring_regex
13190 { ^dot(\s+(.+))? }
13191 \seq_new:N
13192 \l_@@_dot_matches_seq
13193 \markdownSetup {
13194 rendererPrototypes = {
13195 inputFencedCode = {
13196 \regex_extract_once:NnNTF
13197 \c_@@_dot_infostring_regex
13198 { #2 }
13199 \l_@@_dot_matches_seq
13200 {
13201 \@@_if_option:nF
13202 { frozenCache }
13203 {
13204 \sys_shell_now:n
13205 {

```

```

13206 if~!~test~-e~#1.pdf.source~
13207 ||~!~diff~#1~#1.pdf.source;
13208 then~
13209 dot~-Tpdf~-o~#1.pdf~#1;
13210 cp~#1~#1.pdf.source;
13211 fi
13212 }
13213 }

```

We include the typeset image using the image token renderer:

```

13214 \exp_args:NNne
13215 \exp_last_unbraced:No
13216 \markdownRendererImage
13217 {
13218 { Graphviz~image }
13219 { #1.pdf }
13220 { #1.pdf }
13221 }
13222 {
13223 \seq_item:Nn
13224 \l_@@_dot_matches_seq
13225 { 3 }
13226 }
13227 }

```

If the infostring does not start with `dot ...`, we use the previous definition of the fenced code token renderer prototype:

```

13228 {
13229 \@@_dot_previous_definition:nnn
13230 { #1 }
13231 { #2 }
13232 { #3 }
13233 }
13234 },
13235 },
13236 }
13237 }

```

The theme `witiko/diagrams` loads either the theme `witiko/dot` for version `v1` or the theme `witiko/diagrams/v2` for version `v2`.

```

13238 \prop_gput:Nnn
13239 \g_@@_plain_tex_built_in_themes_prop
13240 { witiko / diagrams }
13241 {
13242 \str_case:enF
13243 { \markdownThemeVersion }
13244 {
13245 { latest }

```

```

13246 {
13247 \markdownWarning
13248 {
13249 Write~"witiko/diagrams@v2"~to~pin~version~"v2"~of~the~
13250 theme~"witiko/diagrams".~This~will~keep~your~documents~
13251 from~suddenly~breaking~when~we~have~released~future~
13252 versions~of~the~theme~with~backwards~incompatible~
13253 syntax~and~behavior.
13254 }
13255 \markdownSetup
13256 {
13257 import = witiko/diagrams/v2,
13258 }
13259 }
13260 { v2 }
13261 {
13262 \markdownSetup
13263 {
13264 import = witiko/diagrams/v2,
13265 }
13266 }
13267 { v1 }
13268 {
13269 \markdownSetup
13270 {
13271 import = witiko/dot@silent,
13272 }
13273 }
13274 }
13275 {
13276 \msg_error:nnen
13277 { markdown }
13278 { unknown-theme-version }
13279 { witiko/diagrams }
13280 { \markdownThemeVersion }
13281 { v1 }
13282 }
13283 }
13284 \cs_generate_variant:Nn
13285 \msg_error:nnnnn
13286 { nnnen }
13287 \msg_new:nnnn
13288 { markdown }
13289 { unknown-theme-version }
13290 { Unknown~version~"#2"~of~theme~"#1"~has~been~requested. }
13291 { Known~versions~are:~#3 }

```

Next, we implement the theme `witiko/diagrams/v2`.

```
13292 \prop_gput:Nnn
13293 \g_@@_plain_tex_built_in_themes_prop
13294 { witiko / diagrams / v2 }
13295 {
```

We enable the `fencedCode` and `fencedCodeAttributes` Lua option.

```
13296 \@@_setup:n
13297 {
13298 fencedCode = true,
13299 fencedCodeAttributes = true,
13300 }
```

Store the previous fenced code token renderer prototype.

```
13301 \cs_set_eq:NN
13302 \@@_diagrams_previous_fenced_code:nnn
13303 \markdownRendererInputFencedCodePrototype
```

Store the caption of the diagram.

```
13304 \tl_new:N
13305 \l_@@_diagrams_caption_tl
13306 \@@_setup:n
13307 {
13308 rendererPrototypes = {
```

Route attributes on fenced code blocks to the image attribute renderer prototypes.

```
13309 fencedCodeAttributeContextBegin = {
13310 \group_begin:
13311 \markdownRendererImageAttributeContextBegin
13312 \cs_set_eq:NN
13313 \@@_diagrams_previous_key_value:nn
13314 \markdownRendererAttributeKeyValuePrototype
13315 \@@_setup:n
13316 {
13317 rendererPrototypes = {
13318 attributeKeyValue = {
13319 \str_if_eq:nnTF
13320 { ##1 }
13321 { caption }
13322 {
13323 \tl_set:Nn
13324 \l_@@_diagrams_caption_tl
13325 { ##2 }
13326 }
13327 }
13328 \@@_diagrams_previous_key_value:nn
13329 { ##1 }
13330 { ##2 }
13331 }

```

```

13332 },
13333 },
13334 }
13335 },
13336 fencedCodeAttributeContextEnd = {
13337 \markdownRendererImageAttributeContextEnd
13338 \group_end:
13339 },
13340 },
13341 }
13342 \cs_new:Nn
13343 \@@_diagrams_render_diagram:nnnn
13344 {
13345 \@@_if_option:nF
13346 { frozenCache }
13347 {
13348 \sys_shell_now:n
13349 {
13350 if~!~test~-e~#2.source~
13351 ||~!~diff~#1~#2.source;
13352 then~
13353 (#3);
13354 cp~#1~#2.source;
13355 fi
13356 }
13357 \exp_args:NNnV
13358 \exp_last_unbraced:No
13359 \markdownRendererImage
13360 {
13361 { #4 }
13362 { #2 }
13363 { #2 }
13364 }
13365 \l_@@_diagrams_caption_tl
13366 }
13367 }

```

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.

```

13368 \prop_new:N
13369 \g_markdown_diagrams_infostrings_prop

```

If we know a processing function for a given infostring, use it.

```

13370 \@@_setup:n
13371 {
13372 rendererPrototypes = {
13373 inputFencedCode = {

```

```

13374 \prop_get:NnNTF
13375 \g_markdown_diagrams_infostrings_prop
13376 { #2 }
13377 \l_tmpa_tl
13378 {
13379 \cs_set:NV
13380 \@@_diagrams_infostrings_current:n
13381 \l_tmpa_tl
13382 \@@_diagrams_infostrings_current:n
13383 { #1 }
13384 }

```

Otherwise, use the previous fenced code token renderer prototype.

```

13385 {
13386 \@@_diagrams_previous_fenced_code:nnn
13387 { #1 }
13388 { #2 }
13389 { #3 }
13390 }
13391 },
13392 },
13393 }
13394 \cs_generate_variant:Nn
13395 \cs_set:Nn
13396 { NV }

```

Typeset fenced code with infostring `dot` using the command `dot` from the package `Graphviz`.

```

13397 \cs_set:Nn
13398 \@@_diagrams_infostrings_current:n
13399 {
13400 \@@_diagrams_render_diagram:nnnn
13401 { #1 }
13402 { #1.pdf }
13403 { dot~--Tpdf~-o~#1.pdf~#1 }
13404 { Graphviz~image }
13405 }
13406 \@@_tl_set_from_cs:NNn
13407 \l_tmpa_tl
13408 \@@_diagrams_infostrings_current:n
13409 { 1 }
13410 \prop_gput:NnV
13411 \g_markdown_diagrams_infostrings_prop
13412 { dot }
13413 \l_tmpa_tl

```

Typeset fenced code with infostring `mermaid` using the command `mmdc` from the npm package `@mermaid-js/mermaid-cli`.

```

13414 \cs_set:Nn
13415 \@@_diagrams_infostrings_current:n
13416 {
13417 \@@_diagrams_render_diagram:n
13418 { #1 }
13419 { #1.pdf }
13420 { mmdc---pdfFit~i~#1~o~#1.pdf }
13421 { Mermaid~image }
13422 }
13423 \@@_tl_set_from_cs:NNn
13424 \l_tmpa_tl
13425 \@@_diagrams_infostrings_current:n
13426 { 1 }
13427 \prop_gput:NnV
13428 \g_markdown_diagrams_infostrings_prop
13429 { mermaid }
13430 \l_tmpa_tl

```

Typeset fenced code with infostring `plantuml` using the command `plantuml` from the package PlantUML.

```

13431 \regex_const:Nn
13432 \c_@@_diagrams_filename_suffix_regex
13433 { \.[^\.]*$ }
13434 \cs_set:Nn
13435 \@@_diagrams_infostrings_current:n
13436 {
13437 \tl_set:Nn
13438 \l_tmpa_tl
13439 { #1 }
13440 \regex_replace_once:NnN
13441 \c_@@_diagrams_filename_suffix_regex
13442 { .pdf }
13443 \l_tmpa_tl
13444 \@@_diagrams_render_diagram:nVnn
13445 { #1 }
13446 \l_tmpa_tl
13447 { plantuml~tpdf~#1 }
13448 { PlantUML~image }
13449 }
13450 \cs_generate_variant:Nn
13451 \@@_diagrams_render_diagram:n
13452 { nVnn }
13453 \@@_tl_set_from_cs:NNn
13454 \l_tmpa_tl
13455 \@@_diagrams_infostrings_current:n
13456 { 1 }
13457 \prop_gput:NnV

```



```

13458 \g_markdown_diagrams_infostrings_prop
13459 { plantuml }
13460 \l_tmpa_tl
13461 }

```

We locally change the category code of percent signs, so that we can use them in the shell code:

```

13462 \group_begin:
13463 \char_set_catcode_other:N \%

```

The [witiko/graphicx/http](#) theme stores the previous definition of the image token renderer prototype:

```

13464 \prop_gput:Nnn
13465 \g_@@_plain_tex_built_in_themes_prop
13466 { witiko / graphicx / http }
13467 {
13468 \cs_set_eq:NN
13469 \@@_graphicx_http_previous_definition:nnnn
13470 \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.

```

13471 \int_new:N
13472 \g_@@_graphicx_http_image_number_int
13473 \int_gset:Nn
13474 \g_@@_graphicx_http_image_number_int
13475 { 0 }
13476 \cs_new:Nn
13477 \@@_graphicx_http_filename:
13478 {
13479 \markdownOptionCacheDir
13480 / witiko_graphicx_http .
13481 \int_use:N
13482 \g_@@_graphicx_http_image_number_int
13483 }

```

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.

```

13484 \cs_new:Nn
13485 \@@_graphicx_http_download:nn
13486 {
13487 wget~-0~#2~#1~
13488 ||~curl~---location~-o~#2~#1~
13489 ||~rm~-f~#2
13490 }

```

We redefine the image token renderer prototype, so that it tries to download an online image.

```

13491 \str_new:N
13492 \l_@@_graphicx_http_filename_str
13493 \ior_new:N
13494 \g_@@_graphicx_http_filename_ior
13495 \markdownSetup {
13496 rendererPrototypes = {
13497 image = {
13498 \@@_if_option:nF
13499 { frozenCache }
13500 {

```

The image will be downloaded only if the image URL has the http or https protocols and the `frozenCache` plain `TEX` option is disabled:

```

13501 \sys_shell_now:e
13502 {
13503 mkdir~-p~" \markdownOptionCacheDir ";
13504 if~printf~'%s'~"#3"~|~grep~-q~-E~'^https?:';
13505 then~

```

The image will be downloaded to the pathname `cacheDir/⟨the MD5 digest of the image URL⟩.⟨the suffix of the image URL⟩`:

```

13506 OUTPUT_PREFIX=" \markdownOptionCacheDir ";
13507 OUTPUT_BODY="$(printf~'%s'~'#3'
13508 |~md5sum~|~cut~-d~'~'~-f1)";
13509 OUTPUT_SUFFIX="$(printf~'%s'~'#3'
13510 |~sed~'s/.*[.]//')";
13511 OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";

```

The image will be downloaded only if it has not already been downloaded:

```

13512 if~!~[~-e~"$OUTPUT"~];
13513 then~
13514 \@@_graphicx_http_download:nn
13515 { '#3' }
13516 { "$OUTPUT" } ;
13517 printf~'%s'~"$OUTPUT"~
13518 >~" \@@_graphicx_http_filename: ";
13519 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:

```

13520 else~
13521 printf~'%s'~'#3'~
13522 >~" \@@_graphicx_http_filename: ";
13523 fi
13524 }
13525 }

```

We load the pathname of the downloaded image and we typeset the image using the previous definition of the image renderer prototype:

```

13526 \ior_open:Ne
13527 \g_@@_graphicx_http_filename_ior
13528 { \@@_graphicx_http_filename: }
13529 \ior_str_get:NN
13530 \g_@@_graphicx_http_filename_ior
13531 \l_@@_graphicx_http_filename_str
13532 \ior_close:N
13533 \g_@@_graphicx_http_filename_ior
13534 \@@_graphicx_http_previous_definition:nnVn
13535 { #1 }
13536 { #2 }
13537 \l_@@_graphicx_http_filename_str
13538 { #4 }
13539 \int_gincr:N
13540 \g_@@_graphicx_http_image_number_int
13541 }
13542 }
13543 }
13544 \cs_generate_variant:Nn
13545 \ior_open:Nn
13546 { Ne }
13547 \cs_generate_variant:Nn
13548 \@@_graphicx_http_previous_definition:nnnn
13549 { nnVn }
13550 }
13551 \group_end:

```

The [witiko/tilde](#) theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```

13552 \prop_gput:Nnn
13553 \g_@@_plain_tex_built_in_themes_prop
13554 { witiko / tilde }
13555 {
13556 \markdownSetup {
13557 rendererPrototypes = {
13558 tilde = {-},
13559 },
13560 }
13561 }

```

The themes [witiko/example/foo](#) and [witiko/example/bar](#) are supposed to be used in code examples. They don't do anything.

```

13562 \clist_map_inline:nn
13563 { foo, bar }
13564 {

```

```

13565 \prop_gput:Nnn
13566 \g_@@_plain_tex_built_in_themes_prop
13567 { witiko / example / #1 }
13568 {
13569 \markdownWarning
13570 {
13571 The~theme~witiko/example/#1~is~supposed~to~be~used~in~code~
13572 examples.~Using~it~in~actual~code~has~no~effect,~except~
13573 this~warning~message,~and~is~usually~a~mistake.
13574 }
13575 }
13576 }
13577 \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.

```

13578 \def\markdownRendererInterblockSeparatorPrototype{\par}%
13579 \def\markdownRendererParagraphSeparatorPrototype{%
13580 \markdownRendererInterblockSeparator}%
13581 \def\markdownRendererHardLineBreakPrototype{\hfil\break}%
13582 \def\markdownRendererSoftLineBreakPrototype{ }%
13583 \let\markdownRendererEllipsisPrototype\dots
13584 \def\markdownRendererNbspPrototype{~}%
13585 \def\markdownRendererLeftBracePrototype{\char` \{ }%
13586 \def\markdownRendererRightBracePrototype{\char` \} }%
13587 \def\markdownRendererDollarSignPrototype{\char` $ }%
13588 \def\markdownRendererPercentSignPrototype{\char` \% }%
13589 \def\markdownRendererAmpersandPrototype{\& }%
13590 \def\markdownRendererUnderscorePrototype{\char` _ }%
13591 \def\markdownRendererHashPrototype{\char` \# }%
13592 \def\markdownRendererCircumflexPrototype{\char` ^ }%
13593 \def\markdownRendererBackslashPrototype{\char` \ }%
13594 \def\markdownRendererTildePrototype{\char` ~ }%
13595 \def\markdownRendererPipePrototype{| }%
13596 \def\markdownRendererCodeSpanPrototype#1{\tt#1}%
13597 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
13598 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
13599 \markdownInput{#3}}%
13600 \def\markdownRendererContentBlockOnlineImagePrototype{%
13601 \markdownRendererImage}%
13602 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
13603 \markdownRendererInputFencedCode{#3}{#2}{#2}}%
13604 \def\markdownRendererImagePrototype#1#2#3#4{#2}%

```

```

13605 \def\markdownRendererUlBeginPrototype{}%
13606 \def\markdownRendererUlBeginTightPrototype{}%
13607 \def\markdownRendererUlItemPrototype{}%
13608 \def\markdownRendererUlItemEndPrototype{}%
13609 \def\markdownRendererUlEndPrototype{}%
13610 \def\markdownRendererUlEndTightPrototype{}%
13611 \def\markdownRendererOlBeginPrototype{}%
13612 \def\markdownRendererOlBeginTightPrototype{}%
13613 \def\markdownRendererFancyOlBeginPrototype#1#2{%
13614 \markdownRendererOlBegin}%
13615 \def\markdownRendererFancyOlBeginTightPrototype#1#2{%
13616 \markdownRendererOlBeginTight}%
13617 \def\markdownRendererOlItemPrototype{}%
13618 \def\markdownRendererOlItemWithNumberPrototype#1{}%
13619 \def\markdownRendererOlItemEndPrototype{}%
13620 \def\markdownRendererFancyOlItemPrototype{\markdownRendererOlItem}%
13621 \def\markdownRendererFancyOlItemWithNumberPrototype{%
13622 \markdownRendererOlItemWithNumber}%
13623 \def\markdownRendererFancyOlItemEndPrototype{}%
13624 \def\markdownRendererOlEndPrototype{}%
13625 \def\markdownRendererOlEndTightPrototype{}%
13626 \def\markdownRendererFancyOlEndPrototype{\markdownRendererOlEnd}%
13627 \def\markdownRendererFancyOlEndTightPrototype{%
13628 \markdownRendererOlEndTight}%
13629 \def\markdownRendererDlBeginPrototype{}%
13630 \def\markdownRendererDlBeginTightPrototype{}%
13631 \def\markdownRendererDlItemPrototype#1{#1}%
13632 \def\markdownRendererDlItemEndPrototype{}%
13633 \def\markdownRendererDlDefinitionBeginPrototype{}%
13634 \def\markdownRendererDlDefinitionEndPrototype{\par}%
13635 \def\markdownRendererDlEndPrototype{}%
13636 \def\markdownRendererDlEndTightPrototype{}%
13637 \def\markdownRendererEmphasisPrototype#1{\it#1}%
13638 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
13639 \def\markdownRendererBlockQuoteBeginPrototype{\begingroup\it}%
13640 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
13641 \def\markdownRendererLineBlockBeginPrototype{\begingroup\parindent=0pt}%
13642 \def\markdownRendererLineBlockEndPrototype{\endgroup}%
13643 \def\markdownRendererInputVerbatimPrototype#1{%
13644 \par{\tt\input#1\relax}\par}%
13645 \def\markdownRendererInputFencedCodePrototype#1#2#3{%
13646 \markdownRendererInputVerbatim{#1}}%
13647 \def\markdownRendererHeadingOnePrototype#1{#1}%
13648 \def\markdownRendererHeadingTwoPrototype#1{#1}%
13649 \def\markdownRendererHeadingThreePrototype#1{#1}%
13650 \def\markdownRendererHeadingFourPrototype#1{#1}%
13651 \def\markdownRendererHeadingFivePrototype#1{#1}%

```

```

13652 \def\markdownRendererHeadingSixPrototype#1{#1}%
13653 \def\markdownRendererThematicBreakPrototype{}%
13654 \def\markdownRendererNotePrototype#1{#1}%
13655 \def\markdownRendererCitePrototype#1{}%
13656 \def\markdownRendererTextCitePrototype#1{}%
13657 \def\markdownRendererTickedBoxPrototype{[X]}%
13658 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
13659 \def\markdownRendererUntickedBoxPrototype{[]}%
13660 \def\markdownRendererStrikeThroughPrototype#1{#1}%
13661 \def\markdownRendererSuperscriptPrototype#1{#1}%
13662 \def\markdownRendererSubscriptPrototype#1{#1}%
13663 \def\markdownRendererDisplayMathPrototype#1{##1$}$%
13664 \def\markdownRendererInlineMathPrototype#1{##1$}%
13665 \ExplSyntaxOn
13666 \cs_gset:Npn
13667 \markdownRendererHeaderAttributeContextBeginPrototype
13668 {
13669 \group_begin:
13670 \color_group_begin:
13671 }
13672 \cs_gset:Npn
13673 \markdownRendererHeaderAttributeContextEndPrototype
13674 {
13675 \color_group_end:
13676 \group_end:
13677 }
13678 \cs_gset_eq:NN
13679 \markdownRendererBracketedSpanAttributeContextBeginPrototype
13680 \markdownRendererHeaderAttributeContextBeginPrototype
13681 \cs_gset_eq:NN
13682 \markdownRendererBracketedSpanAttributeContextEndPrototype
13683 \markdownRendererHeaderAttributeContextEndPrototype
13684 \cs_gset_eq:NN
13685 \markdownRendererFencedDivAttributeContextBeginPrototype
13686 \markdownRendererHeaderAttributeContextBeginPrototype
13687 \cs_gset_eq:NN
13688 \markdownRendererFencedDivAttributeContextEndPrototype
13689 \markdownRendererHeaderAttributeContextEndPrototype
13690 \cs_gset_eq:NN
13691 \markdownRendererFencedCodeAttributeContextBeginPrototype
13692 \markdownRendererHeaderAttributeContextBeginPrototype
13693 \cs_gset_eq:NN
13694 \markdownRendererFencedCodeAttributeContextEndPrototype
13695 \markdownRendererHeaderAttributeContextEndPrototype
13696 \cs_gset:Npn
13697 \markdownRendererReplacementCharacterPrototype
13698 { \codepoint_str_generate:n { fffd } }

```

```

13699 \ExplSyntaxOff
13700 \def\markdownRendererSectionBeginPrototype{}%
13701 \def\markdownRendererSectionEndPrototype{}%
13702 \ExplSyntaxOn
13703 \cs_gset:Npn
13704 \markdownRendererWarningPrototype
13705 #1#2#3#4
13706 {
13707 \tl_set:Nn
13708 \l_tmpa_tl
13709 { #2 }
13710 \tl_if_empty:nF
13711 { #4 }
13712 {
13713 \tl_put_right:Nn
13714 \l_tmpa_tl
13715 { \iow_newline: #4 }
13716 }
13717 \exp_args:NV
13718 \markdownWarning
13719 \l_tmpa_tl
13720 }
13721 \ExplSyntaxOff
13722 \def\markdownRendererErrorPrototype#1#2#3#4{%
13723 \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.

```

13724 \ExplSyntaxOn
13725 \cs_new:Nn
13726 \@@_plain_tex_default_input_raw_inline:nn
13727 {
13728 \str_case:nn
13729 { #2 }
13730 {
13731 { md } { \markdownInput{#1} }
13732 { tex } { \markdownEscape{#1} \unskip }
13733 }
13734 }
13735 \cs_new:Nn
13736 \@@_plain_tex_default_input_raw_block:nn
13737 {
13738 \str_case:nn
13739 { #2 }

```

```

13740 {
13741 { md } { \markdownInput{#1} }
13742 { tex } { \markdownEscape{#1} }
13743 }
13744 }
13745 \cs_gset:Npn
13746 \markdownRendererInputRawInlinePrototype#1#2
13747 {
13748 \@@_plain_tex_default_input_raw_inline:nn
13749 { #1 }
13750 { #2 }
13751 }
13752 \cs_gset:Npn
13753 \markdownRendererInputRawBlockPrototype#1#2
13754 {
13755 \@@_plain_tex_default_input_raw_block:nn
13756 { #1 }
13757 { #2 }
13758 }
13759 \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  $p$ .

```

13760 \ExplSyntaxOn
13761 \seq_new:N \g_@@_jekyll_data_datatypes_seq
13762 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
13763 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
13764 \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.



```

13765 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
13766 \cs_new:Nn \@@_jekyll_data_push_address_segment:n
13767 {
13768 \seq_if_empty:NF
13769 \g_@@_jekyll_data_datatypes_seq
13770 {
13771 \seq_get_right:NN
13772 \g_@@_jekyll_data_datatypes_seq
13773 \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.

```

13774 \str_if_eq:NNTF
13775 \l_tmpa_tl
13776 \c_@@_jekyll_data_sequence_tl
13777 {
13778 \seq_put_right:Nn
13779 \g_@@_jekyll_data_wildcard_absolute_address_seq
13780 { * }
13781 }
13782 {
13783 \seq_put_right:Nn
13784 \g_@@_jekyll_data_wildcard_absolute_address_seq
13785 { #1 }
13786 }
13787 }
13788 }

```

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.

```
13789 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
13790 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
13791 \cs_new:Nn \@@_jekyll_data_concatenate_address:NN
13792 {
13793 \seq_pop_left:NN #1 \l_tmpa_tl
13794 \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
13795 \seq_put_left:NV #1 \l_tmpa_tl
13796 }
13797 \cs_new:Nn \@@_jekyll_data_update_address_tls:
13798 {
13799 \@@_jekyll_data_concatenate_address:NN
13800 \g_@@_jekyll_data_wildcard_absolute_address_seq
13801 \g_@@_jekyll_data_wildcard_absolute_address_tl
13802 \seq_get_right:NN
13803 \g_@@_jekyll_data_wildcard_absolute_address_seq
13804 \g_@@_jekyll_data_wildcard_relative_address_tl
13805 }
```

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.

```
13806 \cs_new:Nn \@@_jekyll_data_push:nN
13807 {
13808 \@@_jekyll_data_push_address_segment:n
13809 { #1 }
13810 \seq_put_right:NV
13811 \g_@@_jekyll_data_datatypes_seq
13812 #2
13813 \@@_jekyll_data_update_address_tls:
13814 }
13815 \cs_new:Nn \@@_jekyll_data_pop:
13816 {
13817 \seq_pop_right:NN
13818 \g_@@_jekyll_data_wildcard_absolute_address_seq
13819 \l_tmpa_tl
13820 \seq_pop_right:NN
13821 \g_@@_jekyll_data_datatypes_seq
13822 \l_tmpa_tl
13823 \@@_jekyll_data_update_address_tls:
13824 }
```

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.

```

13825 \cs_new:Nn \@@_jekyll_data_set_keyval_known:nn
13826 {
13827 \keys_set_known:nn
13828 { markdown/jekyllData }
13829 { { #1 } = { #2 } }
13830 }
13831 \cs_generate_variant:Nn
13832 \@@_jekyll_data_set_keyval_known:nn
13833 { Vn }
13834 \cs_new:Nn \@@_jekyll_data_set_keyvals_known:nn
13835 {
13836 \@@_jekyll_data_push:nN
13837 { #1 }
13838 \c_@@_jekyll_data_scalar_tl
13839 \@@_jekyll_data_set_keyval_known:Vn
13840 \g_@@_jekyll_data_wildcard_absolute_address_tl
13841 { #2 }
13842 \@@_jekyll_data_set_keyval_known:Vn
13843 \g_@@_jekyll_data_wildcard_relative_address_tl
13844 { #2 }
13845 \@@_jekyll_data_pop:
13846 }

```

Finally, we will register our macros as token renderer prototypes to be able to react to the traversal of a YAML document.

```

13847 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
13848 \@@_jekyll_data_push:nN
13849 { #1 }
13850 \c_@@_jekyll_data_sequence_tl
13851 }
13852 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
13853 \@@_jekyll_data_push:nN
13854 { #1 }
13855 \c_@@_jekyll_data_mapping_tl
13856 }
13857 \def\markdownRendererJekyllDataSequenceEndPrototype{
13858 \@@_jekyll_data_pop:
13859 }
13860 \def\markdownRendererJekyllDataMappingEndPrototype{
13861 \@@_jekyll_data_pop:
13862 }
13863 \def\markdownRendererJekyllDataBooleanPrototype#1#2{
13864 \@@_jekyll_data_set_keyvals_known:nn
13865 { #1 }

```

```

13866 { #2 }
13867 }
13868 \def\markdownRendererJekyllDataEmptyPrototype#1{}
13869 \def\markdownRendererJekyllDataNumberPrototype#1#2{
13870 \@@_jekyll_data_set_keyvals_known:n
13871 { #1 }
13872 { #2 }
13873 }

```

We will process all string scalar values assuming that they may contain markdown markup and are intended for typesetting.

```

13874 \def\markdownRendererJekyllDataProgrammaticStringPrototype#1#2{}
13875 \def\markdownRendererJekyllDataTypographicStringPrototype#1#2{
13876 \@@_jekyll_data_set_keyvals_known:n
13877 { #1 }
13878 { #2 }
13879 }
13880 \ExplSyntaxOff

```

### 3.2.3.3 Complex YAML Metadata Renderer Prototypes

In this section, we implement the high-level interface for routing complex YAML metadata to expl3 key-values using the option `jekyllDataKeyValue=<module>`. See also Section 2.2.6.1.

```

13881 \ExplSyntaxOn
13882 \@@_with_various_cases:n
13883 { jekyllDataKeyValue }
13884 {
13885 \keys_define:n
13886 { markdown/options }
13887 {
13888 #1 .code:n = {
13889 \@@_route_jekyll_data_to_key_values:n
13890 { ##1 }
13891 },

```

When no `<module>` has been provided, assume that the YAML metadata specify absolute paths to key-values.

```

13892 #1 .default:n = { },
13893 }
13894 }
13895 \seq_new:N
13896 \l_@@_jekyll_data_current_position_seq
13897 \tl_new:N
13898 \l_@@_jekyll_data_current_position_tl
13899 \cs_new:Nn
13900 \@@_route_jekyll_data_to_key_values:n
13901 {

```

```

13902 \markdownSetup
13903 {
13904 renderers = {
13905 jekyllData(Sequence|Mapping)Begin = {
13906 \bool_lazy_and:nnTF
13907 {
13908 \seq_if_empty_p:N
13909 \l_@@_jekyll_data_current_position_seq
13910 }
13911 {
13912 \str_if_eq_p:nn
13913 { ##1 }
13914 { null }
13915 }
13916 {
13917 \tl_if_empty:nF
13918 { #1 }
13919 {
13920 \seq_put_right:Nn
13921 \l_@@_jekyll_data_current_position_seq
13922 { #1 }
13923 }
13924 }
13925 {
13926 \seq_put_right:Nn
13927 \l_@@_jekyll_data_current_position_seq
13928 { ##1 }
13929 }
13930 },
13931 jekyllData(Sequence|Mapping)End = {
13932 \seq_pop_right:NN
13933 \l_@@_jekyll_data_current_position_seq
13934 \l_tmpa_tl
13935 },

```

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**.

```

13936 jekyllDataBoolean = {
13937 \tl_set:Nx
13938 \l_@@_jekyll_data_current_position_tl
13939 {
13940 \seq_use:Nn
13941 \l_@@_jekyll_data_current_position_seq
13942 { / }
13943 }

```

```

13944 \keys_if_exist:VnTF
13945 \l_@@_jekyll_data_current_position_tl
13946 { ##1 / boolean }
13947 {
13948 \@@_keys_set:xn
13949 {
13950 \tl_use:N
13951 \l_@@_jekyll_data_current_position_tl
13952 / ##1 / boolean
13953 }
13954 { ##2 }
13955 }
13956 {
13957 \@@_keys_set:xn
13958 {
13959 \tl_use:N
13960 \l_@@_jekyll_data_current_position_tl
13961 / ##1
13962 }
13963 { ##2 }
13964 }
13965 },
13966 jekyllDataNumber = {
13967 \tl_set:Nx
13968 \l_@@_jekyll_data_current_position_tl
13969 {
13970 \seq_use:Nn
13971 \l_@@_jekyll_data_current_position_seq
13972 { / }
13973 }
13974 \keys_if_exist:VnTF
13975 \l_@@_jekyll_data_current_position_tl
13976 { ##1 / number }
13977 {
13978 \@@_keys_set:xn
13979 {
13980 \tl_use:N
13981 \l_@@_jekyll_data_current_position_tl
13982 / ##1 / number
13983 }
13984 { ##2 }
13985 }
13986 {
13987 \@@_keys_set:xn
13988 {
13989 \tl_use:N
13990 \l_@@_jekyll_data_current_position_tl

```

```

13991 / ##1
13992 }
13993 { ##2 }
13994 }
13995 },

```

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`.

```

13996 jekyllDataEmpty = {
13997 \tl_set:Nx
13998 \l_@@_jekyll_data_current_position_tl
13999 {
14000 \seq_use:Nn
14001 \l_@@_jekyll_data_current_position_seq
14002 { / }
14003 }
14004 \keys_if_exist:VnTF
14005 \l_@@_jekyll_data_current_position_tl
14006 { ##1 / empty }
14007 {
14008 \keys_set:xn
14009 {
14010 \tl_use:N
14011 \l_@@_jekyll_data_current_position_tl
14012 / ##1
14013 }
14014 { empty }
14015 }
14016 {
14017 \keys_set:Vn
14018 \l_@@_jekyll_data_current_position_tl
14019 { ##1 }
14020 }
14021 },

```

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.

```

14022 jekyllDataTypographicString = {
14023 \tl_set:Nx

```

```

14024 \l_@@_jekyll_data_current_position_tl
14025 {
14026 \seq_use:Nn
14027 \l_@@_jekyll_data_current_position_seq
14028 { / }
14029 }
14030 \keys_if_exist:VnTF
14031 \l_@@_jekyll_data_current_position_tl
14032 { ##1 / typographicString }
14033 {
14034 \@@_keys_set:xn
14035 {
14036 \tl_use:N
14037 \l_@@_jekyll_data_current_position_tl
14038 / ##1 / typographicString
14039 }
14040 { ##2 }
14041 }
14042 {
14043 \keys_if_exist:VnTF
14044 \l_@@_jekyll_data_current_position_tl
14045 { ##1 / programmaticString }
14046 {
14047 \@@_keys_set_known:xn
14048 {
14049 \tl_use:N
14050 \l_@@_jekyll_data_current_position_tl
14051 / ##1
14052 }
14053 { ##2 }
14054 }
14055 {
14056 \@@_keys_set:xn
14057 {
14058 \tl_use:N
14059 \l_@@_jekyll_data_current_position_tl
14060 / ##1
14061 }
14062 { ##2 }
14063 }
14064 }
14065 },
14066 jekyllDataProgrammaticString = {
14067 \tl_set:Nx
14068 \l_@@_jekyll_data_current_position_tl
14069 {
14070 \seq_use:Nn

```



```

14071 \l_@@_jekyll_data_current_position_seq
14072 { / }
14073 }
14074 \keys_if_exist:VnT
14075 \l_@@_jekyll_data_current_position_tl
14076 { ##1 / programmaticString }
14077 {
14078 \@@_keys_set:xn
14079 {
14080 \tl_use:N
14081 \l_@@_jekyll_data_current_position_tl
14082 / ##1 / programmaticString
14083 }
14084 { ##2 }
14085 }
14086 },
14087 },
14088 }
14089 }
14090 \cs_new:Nn
14091 \@@_keys_set:nn
14092 {
14093 \keys_set:nn
14094 { }
14095 { { #1 } = { #2 } }
14096 }
14097 \cs_new:Nn
14098 \@@_keys_set_known:nn
14099 {
14100 \keys_set_known:nn
14101 { }
14102 { { #1 } = { #2 } }
14103 }
14104 \cs_generate_variant:Nn
14105 \@@_keys_set:nn
14106 { xn }
14107 \cs_generate_variant:Nn
14108 \@@_keys_set_known:nn
14109 { xn }
14110 \cs_generate_variant:Nn
14111 \keys_set:nn
14112 { xn, Vn }
14113 \prg_generate_conditional_variant:Nnn
14114 \keys_if_exist:nn
14115 { Vn }
14116 { T, TF }
14117 \ExplSyntaxOff

```

If plain  $\TeX$  is the top layer, we load the [witiko/markdown/defaults](#) plain  $\TeX$  theme with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```

14118 \ExplSyntaxOn
14119 \str_if_eq:VVT
14120 \c_@@_top_layer_tl
14121 \c_@@_option_layer_plain_tex_tl
14122 {
14123 \use:c
14124 { ExplSyntaxOff }
14125 \@@_if_option:nF
14126 { noDefaults }
14127 {
14128 \@@_if_option:nTF
14129 { experimental }
14130 {
14131 \@@_setup:n
14132 { theme = witiko/markdown/defaults@experimental }
14133 }
14134 {
14135 \@@_setup:n
14136 { theme = witiko/markdown/defaults }
14137 }
14138 }
14139 \use:c
14140 { ExplSyntaxOn }
14141 }
14142 \ExplSyntaxOff

```

### 3.2.4 Lua Snippets

After the `\markdownPrepareLuaOptions` macro has been fully expanded, the `\markdownLuaOptions` macro will expand to a Lua table that contains the plain  $\TeX$  options (see Section 2.2.2) in a format recognized by Lua (see Section 2.1.3).

```

14143 \ExplSyntaxOn
14144 \tl_new:N \g_@@_formatted_lua_options_tl
14145 \cs_new:Nn \@@_format_lua_options:
14146 {
14147 \tl_gclear:N
14148 \g_@@_formatted_lua_options_tl
14149 \seq_map_function:NN
14150 \g_@@_lua_options_seq
14151 \@@_format_lua_option:n
14152 }
14153 \cs_new:Nn \@@_format_lua_option:n
14154 {

```

```

14155 \@@_typecheck_option:n
14156 { #1 }
14157 \@@_get_option_type:nN
14158 { #1 }
14159 \l_tmpa_tl
14160 \bool_case_true:nF
14161 {
14162 {
14163 \str_if_eq_p:VV
14164 \l_tmpa_tl
14165 \c_@@_option_type_boolean_tl ||
14166 \str_if_eq_p:VV
14167 \l_tmpa_tl
14168 \c_@@_option_type_number_tl ||
14169 \str_if_eq_p:VV
14170 \l_tmpa_tl
14171 \c_@@_option_type_counter_tl
14172 }
14173 {
14174 \@@_get_option_value:nN
14175 { #1 }
14176 \l_tmpa_tl
14177 \tl_gput_right:Nx
14178 \g_@@_formatted_lua_options_tl
14179 { #1~::~ \l_tmpa_tl ,~ }
14180 }
14181 }
14182 \str_if_eq_p:VV
14183 \l_tmpa_tl
14184 \c_@@_option_type_clist_tl
14185 }
14186 {
14187 \@@_get_option_value:nN
14188 { #1 }
14189 \l_tmpa_tl
14190 \tl_gput_right:Nx
14191 \g_@@_formatted_lua_options_tl
14192 { #1~::~\c_left_brace_str }
14193 \clist_map_inline:Vn
14194 \l_tmpa_tl
14195 {
14196 \@@_lua_escape:xN
14197 { ##1 }
14198 \l_tmpb_tl
14199 \tl_gput_right:Nn
14200 \g_@@_formatted_lua_options_tl
14201 { " }

```

```

14202 \tl_gput_right:NV
14203 \g_@@_formatted_lua_options_tl
14204 \l_tmpb_tl
14205 \tl_gput_right:Nn
14206 \g_@@_formatted_lua_options_tl
14207 { " ,~ }
14208 }
14209 \tl_gput_right:Nx
14210 \g_@@_formatted_lua_options_tl
14211 { \c_right_brace_str ,~ }
14212 }
14213 }
14214 {
14215 \@@_get_option_value:nN
14216 { #1 }
14217 \l_tmpa_tl
14218 \@@_lua_escape:xN
14219 { \l_tmpa_tl }
14220 \l_tmpb_tl
14221 \tl_gput_right:Nn
14222 \g_@@_formatted_lua_options_tl
14223 { #1~== " }
14224 \tl_gput_right:NV
14225 \g_@@_formatted_lua_options_tl
14226 \l_tmpb_tl
14227 \tl_gput_right:Nn
14228 \g_@@_formatted_lua_options_tl
14229 { " ,~ }
14230 }
14231 }
14232 \cs_generate_variant:Nn
14233 \clist_map_inline:nn
14234 { Vn }
14235 \let
14236 \markdownPrepareLuaOptions
14237 \@@_format_lua_options:
14238 \def
14239 \markdownLuaOptions
14240 {
14241 {
14242 \g_@@_formatted_lua_options_tl
14243 }
14244 }
14245 \sys_if_engine luatex:TF
14246 {
14247 \cs_new:Nn
14248 \@@_lua_escape:nN

```

```

14249 {
14250 \tl_set:Nx
14251 #2
14252 {
14253 \lua_escape:n
14254 { #1 }
14255 }
14256 }
14257 }
14258 {
14259 \regex_const:Nn
14260 \c_@@_lua_escape_regex
14261 { [\\"'] }
14262 \cs_new:Nn
14263 \@@_lua_escape:nN
14264 {
14265 \tl_set:Nn
14266 #2
14267 { #1 }
14268 \regex_replace_all:NnN
14269 \c_@@_lua_escape_regex
14270 { \u { c_backslash_str } \0 }
14271 #2
14272 }
14273 }
14274 \cs_generate_variant:Nn
14275 \@@_lua_escape:nN
14276 { 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.

```

14277 \tl_new:N
14278 \markdownInputFilename
14279 \cs_new:Npn
14280 \markdownPrepareInputFilename
14281 #1
14282 {
14283 \@@_lua_escape:xN
14284 { #1 }
14285 \markdownInputFilename
14286 \tl_gset:Nx
14287 \markdownInputFilename
14288 { " \markdownInputFilename " }
14289 }

```

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.

```
14290 \cs_new:Npn
14291 \markdownPrepare
14292 {
14293 local-lfs = require("lfs")
14294 local-options = \markdownLuaOptions
14295 if-not-lfs.isdir(options.cacheDir) then~
14296 assert(lfs.mkdir(options.cacheDir))
14297 end~
```

First, ensure that the `cacheDir` directory exists.

```
14293 local-lfs = require("lfs")
14294 local-options = \markdownLuaOptions
14295 if-not-lfs.isdir(options.cacheDir) then~
14296 assert(lfs.mkdir(options.cacheDir))
14297 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.

```
14298 local-md = require("markdown")
14299 local-convert = md.new(options)
14300 }
```

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.

```
14301 \cs_new:Npn
14302 \markdownConvert
14303 {
14304 local-filename = \markdownInputFilename
14305 local-file = assert(io.open(filename, "r"),
14306 [[Could-not-open-file~]] .. filename .. [[~for~reading]])
14307 local-input = assert(file:read("*a"))
14308 assert(file:close())
14309 print(convert(input))
14310 }
```

```
14311 \ExplSyntaxOff
```

The `\markdownCleanup` macro contains the Lua code that is executed after any conversion from markdown to plain T<sub>E</sub>X.

```
14312 \def\markdownCleanup{%
```

Remove the `options.cacheDir` directory if it is empty.

```
14313 if options.cacheDir then
14314 lfs.rmdir(options.cacheDir)
14315 end
14316 }%
```

### 3.2.5 Buffering Block-Level Markdown Input

The macros `\markdownInputStream` and `\markdownOutputStream` contain the number of the input and output file streams that will be used for the IO operations of the package.

```
14317 \csname newread\endcsname\markdownInputStream
14318 \csname newwrite\endcsname\markdownOutputStream
```

The `\markdownReadAndConvertTab` macro contains the tab character literal.

```
14319 \begingroup
14320 \catcode`\^^I=12%
14321 \gdef\markdownReadAndConvertTab{^^I}%
14322 \endgroup
```

The `\markdownReadAndConvert` macro is largely a rewrite of the  $\text{\LaTeX 2}\epsilon$  `\filecontents` macro to plain  $\text{\TeX}$ .

```
14323 \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.

```
14324 \catcode`\^^M=13%
14325 \catcode`\^^I=13%
14326 \catcode`|=0%
14327 \catcode`\=12%
14328 |catcode`%=14%
14329 |catcode`|=12@
14330 |gdef|markdownReadAndConvert#1#2{@
14331 |begingroup@
```

If we are not reading markdown documents from the frozen cache, open the `inputTempFileName` file for writing.

```
14332 |markdownIfOption{frozenCache}{-}{@
14333 |immediate|openout|markdownOutputStream@
14334 |markdownOptionInputTempFileName|relax@
14335 |markdownInfo{@
14336 |Buffering block-level markdown input into the temporary @
14337 |input file "|markdownOptionInputTempFileName" and scanning @
14338 |for the closing token sequence "#1"}@
14339 }@
```

Locally change the category of the special plain  $\text{\TeX}$  characters to *other* in order to prevent unwanted interpretation of the input. Change also the category of the space character, so that we can retrieve it unaltered.

```
14340 |def|do##1{|catcode`##1=12}|dospecials@
14341 |catcode`|=12@
```

```
14342 |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 ( $\sim$ M) are produced.

```
14343 |def|markdownReadAndConvertStripPercentSign##1{@
14344 |markdownIfOption{stripPercentSigns}{@
14345 |if##1%{@
14346 |expandafter|expandafter|expandafter@
14347 |markdownReadAndConvertProcessLine@
14348 |else@
14349 |expandafter|expandafter|expandafter@
14350 |markdownReadAndConvertProcessLine@
14351 |expandafter|expandafter|expandafter##1@
14352 |fi@
14353 }{@
14354 |expandafter@
14355 |markdownReadAndConvertProcessLine@
14356 |expandafter##1@
14357 }@
14358 }@
```

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 ( $\sim$ M) are produced.

```
14359 |def|markdownReadAndConvertProcessLine##1##2##3|relax{@
```

If we are not reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, store the line in the `inputTempFileName` file. If we are reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, gobble the line.

```
14360 |ifx|relax##3|relax@
14361 |markdownIfOption{frozenCache}{}{@
14362 |immediate|write|markdownOutputFileStream{##1}@
14363 }@
14364 |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.

```
14365 |def \sim M{@
14366 |markdownInfo{The ending token sequence was found}@
14367 |markdownIfOption{frozenCache}{}{@
14368 |immediate|closeout|markdownOutputFileStream@
14369 }@
```



```

14370 |endgroup@
14371 |markdownInput{@
14372 |markdownOptionOutputDir@
14373 /|markdownOptionInputTempFileName@
14374 }@
14375 #2}@
14376 |fi@

```

Repeat with the next line.

```
14377 ^^M}@
```

Make the tab character active at expansion time and make it expand to a literal tab character.

```

14378 |catcode`\^^I=13@
14379 |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.

```

14380 |catcode`\^^M=13@
14381 |def^^M##1^^M{@
14382 |def^^M###1^^M{@
14383 |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
14384 ^^M}@
14385 ^^M}@

```

Reset the character categories back to the former state.

```
14386 |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.

```

14387 \ExplSyntaxOn
14388 \cs_new:Npn
14389 \markdownLuaExecute
14390 #1
14391 {
14392 \int_compare:nNnT
14393 { \g_luabridge_method_int }
14394 =
14395 { \c_luabridge_method_shell_int }
14396 {
14397 \sys_if_shell_unrestricted:F
14398 {
14399 \sys_if_shell:TF
14400 {
14401 \msg_error:nn
14402 { markdown }
14403 { restricted-shell-access }
14404 }

```

```

14405 {
14406 \msg_error:nn
14407 { markdown }
14408 { disabled-shell-access }
14409 }
14410 }
14411 }
14412 \str_gset:NV
14413 \g_luabridge_output_dirname_str
14414 \markdownOptionOutputDir
14415 \luabridge_now:e
14416 { #1 }
14417 }
14418 \cs_generate_variant:Nn
14419 \msg_new:nnnn
14420 { nnnV }
14421 \tl_set:Nn
14422 \l_tmpa_tl
14423 {
14424 You~may~need~to~run~TeX~with~the~---shell-escape~or~the~
14425 --enable-write18~flag,~or~write~shell_escape=t~in~the~
14426 texmf.cnf~file.
14427 }
14428 \msg_new:nnnV
14429 { markdown }
14430 { restricted-shell-access }
14431 { Shell-escape-is-restricted }
14432 \l_tmpa_tl
14433 \msg_new:nnnV
14434 { markdown }
14435 { disabled-shell-access }
14436 { Shell-escape-is-disabled }
14437 \l_tmpa_tl
14438 \ExplSyntaxOff

```

### 3.2.6 Buffering Inline Markdown Input

This section describes the implementation of the macro `\markinline`.

```

14439 \ExplSyntaxOn
14440 \tl_new:N
14441 \g_@@_after_markinline_tl
14442 \tl_gset:Nn
14443 \g_@@_after_markinline_tl
14444 { \unskip }
14445 \cs_new:Npn
14446 \markinline
14447 {

```

Locally change the category of the special plain T<sub>E</sub>X characters to *other* in order to prevent unwanted interpretation of the input markdown text as T<sub>E</sub>X code.

```
14448 \group_begin:
14449 \cctab_select:N
14450 \c_other_cctab
```

Unless we are reading markdown documents from the frozen cache, open the file `inputTempFileName` for writing.

```
14451 \@@_if_option:nF
14452 { frozenCache }
14453 {
14454 \immediate
14455 \openout
14456 \markdownOutputFileStream
14457 \markdownOptionInputTempFileName
14458 \relax
14459 \msg_info:nne
14460 { markdown }
14461 { buffering-markinline }
14462 { \markdownOptionInputTempFileName }
14463 }
```

Peek ahead and extract the inline markdown text.

```
14464 \peek_regex_replace_once:nnF
14465 { { (.*) } }
14466 {
```

Unless we are reading markdown documents from the frozen cache, store the text in the file `inputTempFileName` and close it.

```
14467 \c { @@_if_option:nF }
14468 \cB { frozenCache \cE }
14469 \cB {
14470 \c { immediate }
14471 \c { write }
14472 \c { markdownOutputFileStream }
14473 \cB { \1 \cE }
14474 \c { immediate }
14475 \c { closeout }
14476 \c { markdownOutputFileStream }
14477 \cE }
```

Reset the category codes and `\input` the result of the conversion.

```
14478 \c { group_end: }
14479 \c { group_begin: }
14480 \c { @@_setup:n }
14481 \cB { contentLevel = inline \cE }
14482 \c { markdownInput }
14483 \cB {
```

```

14484 \c { markdownOptionOutputDir } /
14485 \c { markdownOptionInputTempFileName }
14486 \cE }
14487 \c { group_end: }
14488 \c { tl_use:N }
14489 \c { g_@@_after_markinline_tl }
14490 }
14491 {
14492 \msg_error:nn
14493 { markdown }
14494 { markinline-peek-failure }
14495 \group_end:
14496 \tl_use:N
14497 \g_@@_after_markinline_tl
14498 }
14499 }
14500 \msg_new:nnn
14501 { markdown }
14502 { buffering-markinline }
14503 { Buffering~inline~markdown~input~into~
14504 the~temporary~input~file~"#1". }
14505 \msg_new:nnnn
14506 { markdown }
14507 { markinline-peek-failure }
14508 { Use~of~\iow_char:N \\ markinline~doesn't~match~its~definition }
14509 { The~macro~should~be~followed~by~inline~
14510 markdown~text~in~curly~braces }
14511 \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.

```

14512 \ExplSyntaxOn
14513 \cs_new:Npn
14514 \markdownInput
14515 #1
14516 {
14517 \@@_if_option:nTF
14518 { frozenCache }
14519 {
14520 \markdownInputRaw
14521 { #1 }
14522 }
14523 {

```

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  $\text{\LaTeX}$ , this also includes the directories specified in `\input@path`.

```

14524 \tl_set:Nx
14525 \l_tmpa_tl
14526 { #1 }
14527 \file_get_full_name:VNTF
14528 \l_tmpa_tl
14529 \l_tmpb_tl
14530 {
14531 \exp_args:NV
14532 \markdownInputRaw
14533 \l_tmpb_tl
14534 }
14535 {
14536 \msg_error:nnV
14537 { markdown }
14538 { markdown-file-does-not-exist }
14539 \l_tmpa_tl
14540 }
14541 }
14542 }
14543 \msg_new:nnn
14544 { markdown }
14545 { markdown-file-does-not-exist }
14546 {
14547 Markdown~file~#1~does~not~exist
14548 }
14549 \ExplSyntaxOff
14550 \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.

```

14551 \catcode`|=0%
14552 \catcode`\|=12%
14553 \catcode`|&=6%
14554 |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.

```

14555 |begingroup
14556 |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.

```
14557 |catcode`|#=12
```

If we are reading from the frozen cache, input it, expand the corresponding `\markdownFrozenCache⟨number⟩` macro, and increment `frozenCacheCounter`.

```
14558 |markdownIfOption{frozenCache}{%
14559 |ifnum|markdownOptionFrozenCacheCounter=0|relax
14560 |markdownInfo{Reading frozen cache from
14561 "|markdownOptionFrozenCacheFileName"}%
14562 |input|markdownOptionFrozenCacheFileName|relax
14563 |fi
14564 |markdownInfo{Including markdown document number
14565 "|the|markdownOptionFrozenCacheCounter" from frozen cache}%
14566 |csname markdownFrozenCache%
14567 |the|markdownOptionFrozenCacheCounter|endcsname
14568 |global|advance|markdownOptionFrozenCacheCounter by 1|relax
14569 }{%
14570 |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.

```
14571 |openin|markdownInputFileStream&1
14572 |closein|markdownInputFileStream
14573 |markdownPrepareLuaOptions
14574 |markdownPrepareInputFilename{&1}%
14575 |markdownLuaExecute{%
14576 |markdownPrepare
14577 |markdownConvert
14578 |markdownCleanup}%
```

If we are finalizing the frozen cache, increment `frozenCacheCounter`.

```
14579 |markdownIfOption{finalizeCache}{%
14580 |global|advance|markdownOptionFrozenCacheCounter by 1|relax}{}%
14581 }%
14582 |endgroup
14583 }%
14584 |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.

```
14585 \gdef\markdownEscape#1{%
14586 \catcode`\%=14\relax
```

```

14587 \catcode\#=6\relax
14588 \input #1\relax
14589 \catcode\%=12\relax
14590 \catcode\#=12\relax
14591 }%

```

### 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.

```

14592 \def\markdownVersionSpace{ }%
14593 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
14594 \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).

```

14595 \ExplSyntaxOn
14596 \cs_gset_eq:NN
14597 \markinlinePlainTeX
14598 \markinline
14599 \cs_gset:Npn
14600 \markinline
14601 {
14602 \peek_regex_replace_once:nn
14603 { (\[(.*) \]) ? }
14604 {

```

Apply the options locally.

```

14605 \c { group_begin: }
14606 \c { @@_setup:n }
14607 \cB { \2 \cE }
14608 \c { tl_put_right:Nn }
14609 \c { g_@@_after_markinline_tl }
14610 \cB { \c { group_end: } \cE }
14611 \c { markinlinePlainTeX }
14612 }
14613 }
14614 \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  $\LaTeX$  interface (see Section 2.3.3).

```

14615 \let\markdownInputPlainTeX\markdownInput
14616 \renewcommand\markdownInput[2] []{%
14617 \begingroup
14618 \markdownSetup{#1}%
14619 \markdownInputPlainTeX{#2}%
14620 \endgroup}%
14621 \renewcommand\yamlInput[2] []{%
14622 \begingroup
14623 \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData, #1}%
14624 \markdownInputPlainTeX{#2}%
14625 \endgroup}%

```

The `markdown`, `markdown*`, and `yaml`  $\LaTeX$  environments are implemented using the `\markdownReadAndConvert` macro.

```

14626 \ExplSyntaxOn
14627 \renewenvironment
14628 { markdown }
14629 {

```

In our implementation of the `markdown`  $\LaTeX$  environment, we want to distinguish between the following two cases:

<code>\begin{markdown} [smartEllipses]</code>	<code>\begin{markdown}</code>
<code>% This is an optional argument ^</code>	<code>[smartEllipses]</code>
<code>% ...</code>	<code>% ^ This is link</code>
<code>\end{markdown}</code>	<code>\end{markdown}</code>

Therefore, we cannot use the built-in  $\LaTeX$  support for environments with optional arguments or packages such as `xparse`. Instead, we must read the optional argument manually and prevent reading past the end of a line.

To prevent reading past the end of a line when looking for the optional argument of the `markdown`  $\LaTeX$  environment and accidentally tokenizing markdown text, we change the category code of carriage return (`\r`, ASCII character 13 in decimal) from 5 (end of line).

While any category code other than 5 (end of line) would work, we switch to the category 13 (active), which is also used by the `\markdownReadAndConvert` macro. This is necessary if we read until the end of a line, because then the carriage return character will be produced by  $\TeX$  via the `\endlinechar` plain  $\TeX$  macro and it needs to have the correct category code, so that `\markdownReadAndConvert` processes it correctly.

```

14630 \group_begin:
14631 \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).

```
14632 \char_set_catcode_letter:n { 35 }
```

After we have matched the opening [ that begins the optional argument, we accept carriage returns as well.

```
14633 \peek_regex_replace_once:nnF
14634 { \ *\[r*([^\r]*)\[^\r]* }
14635 {
```

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.

```
14636 \c { group_end: }
14637 \c { tl_set_rescan:Nnn } \c { l_tmpa_tl } { } { \1 }
```

Then, we pass the retokenized content to the `\markdownSetup` macro.

```
14638 \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  $\LaTeX$  environment.

We also make provision for using the `\markdown` command as a part of a different  $\LaTeX$  environment as follows:

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

```
14639 \c { exp_args:NV }
14640 \c { markdownReadAndConvert@ }
14641 \c { @currenvir }
14642 }
14643 {
14644 \group_end:
14645 \exp_args:NV
14646 \markdownReadAndConvert@
14647 \@currenvir
14648 }
14649 }
14650 { \markdownEnd }
14651 \renewenvironment
14652 { markdown* }
14653 [1]
```

```

14654 {
14655 \@@_if_option:nTF
14656 { experimental }
14657 {
14658 \msg_error:nnn
14659 { markdown }
14660 { latex-markdown-star-deprecated }
14661 { #1 }
14662 }
14663 {
14664 \msg_warning:nnn
14665 { markdown }
14666 { latex-markdown-star-deprecated }
14667 { #1 }
14668 }
14669 \@@_setup:n
14670 { #1 }
14671 \markdownReadAndConvert@
14672 { markdown* }
14673 }
14674 { \markdownEnd }
14675 \renewenvironment
14676 { yaml }
14677 {
14678 \group_begin:
14679 \yamlSetup
14680 { jekyllData, expectJekyllData, ensureJekyllData }
14681 \markdown
14682 }
14683 { \yamlEnd }
14684 \msg_new:nnn
14685 { markdown }
14686 { latex-markdown-star-deprecated }
14687 {
14688 The-markdown*-LaTeX-environment-has-been-deprecated-and-will-
14689 be-removed-in-the-next-major-version-of-the-Markdown-package.
14690 }
14691 \cs_generate_variant:Nn
14692 \@@_setup:n
14693 { V }
14694 \ExplSyntaxOff
14695 \beginingroup

```

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*.

```

14696 \catcode`\|=0\catcode`\<=1\catcode`\>=2%
14697 \catcode`\|=12\catcode`\{=12\catcode`\}=12%
14698 |gdef|markdownReadAndConvert@#1<%
14699 |markdownReadAndConvert<\end{#1}>%
14700 <|end<#1>>>%
14701 |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.

```

14702 \ExplSyntaxOn
14703 \prop_new:N \g_@@_latex_loaded_themes_linenos_prop
14704 \prop_new:N \g_@@_latex_loaded_themes_versions_prop
14705 \cs_gset:Nn
14706 \@@_load_theme:nnn
14707 {

```

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.

```

14708 \ifmarkdownLaTeXLoaded
14709 \ifx\@onlypreamble\@notprerr

```

If both conditions are true, end with an error, since we cannot load  $\LaTeX$  themes after the preamble.

```

14710 \bool_if:nTF
14711 {
14712 \bool_lazy_or_p:nn
14713 {
14714 \prop_if_in_p:Nn
14715 \g_@@_latex_built_in_themes_prop
14716 { #1 }
14717 }
14718 {
14719 \file_if_exist_p:n
14720 { markdown theme #3.sty }
14721 }
14722 }
14723 {
14724 \msg_error:nnn
14725 { markdown }
14726 { latex-theme-after-preamble }
14727 { #1 }
14728 }

```

Otherwise, try loading a plain T<sub>E</sub>X theme instead.

```
14729 {
14730 \@@_plain_tex_load_theme:nnn
14731 { #1 }
14732 { #2 }
14733 { #3 }
14734 }
14735 \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.

```
14736 \bool_if:nTF
14737 {
14738 \bool_lazy_or_p:nn
14739 {
14740 \prop_if_in_p:Nn
14741 \g_@@_latex_built_in_themes_prop
14742 { #1 }
14743 }
14744 {
14745 \file_if_exist_p:n
14746 { markdown theme #3.sty }
14747 }
14748 }
14749 {
14750 \prop_get:NnNTF
14751 \g_@@_latex_loaded_themes_linenos_prop
14752 { #1 }
14753 \l_tmpa_tl
14754 {
14755 \prop_get:NnN
14756 \g_@@_latex_loaded_themes_versions_prop
14757 { #1 }
14758 \l_tmpb_tl
14759 \str_if_eq:nVTF
14760 { #2 }
14761 \l_tmpb_tl
14762 {
14763 \msg_warning:nnnVn
14764 { markdown }
14765 { repeatedly-loaded-latex-theme }
14766 { #1 }
14767 \l_tmpa_tl
14768 { #2 }
14769 }
14770 }
14771 \msg_error:nnnnVV
```

```

14772 { markdown }
14773 { different-versions-of-latex-theme }
14774 { #1 }
14775 { #2 }
14776 \l_tmpb_tl
14777 \l_tmpa_tl
14778 }
14779 }
14780 {
14781 \prop_gput:Nnx
14782 \g_@@_latex_loaded_themes_linenos_prop
14783 { #1 }
14784 { \tex_the:D \tex_inputlineno:D } % noqa: W200
14785 \prop_gput:Nnn
14786 \g_@@_latex_loaded_themes_versions_prop
14787 { #1 }
14788 { #2 }

```

Load built-in plain TeX themes from the prop `\g_@@_latex_built_in_themes_prop` and from the filesystem otherwise.

```

14789 \prop_if_in:NnTF
14790 \g_@@_latex_built_in_themes_prop
14791 { #1 }
14792 {
14793 \msg_info:nnnn
14794 { markdown }
14795 { loading-built-in-latex-theme }
14796 { #1 }
14797 { #2 }
14798 \prop_item:Nn
14799 \g_@@_latex_built_in_themes_prop
14800 { #1 }
14801 }
14802 {
14803 \msg_info:nnnn
14804 { markdown }
14805 { loading-latex-theme }
14806 { #1 }
14807 { #2 }
14808 \RequirePackage
14809 { markdown theme #3 }
14810 }
14811 }
14812 }
14813 {
14814 \@@_plain_tex_load_theme:nnn
14815 { #1 }

```

```

14816 { #2 }
14817 { #3 }
14818 }
14819 \fi
14820 \else

```

If the Markdown package has not yet been loaded, postpone the loading until the Markdown package has finished loading.

```

14821 \msg_info:nnnn
14822 { markdown }
14823 { theme-loading-postponed }
14824 { #1 }
14825 { #2 }
14826 \AtEndOfPackage
14827 {
14828 \@@_set_theme:n
14829 { #1 @ #2 }
14830 }
14831 \fi
14832 }
14833 \msg_new:nnn
14834 { markdown }
14835 { theme-loading-postponed }
14836 {
14837 Postponing~loading~version~#2~of~Markdown~theme~#1~until~
14838 Markdown~package~has~finished~loading
14839 }
14840 \msg_new:nnn
14841 { markdown }
14842 { loading-built-in-latex-theme }
14843 { Loading~version~#2~of~built-in~LaTeX~Markdown~theme~#1 }
14844 \msg_new:nnn
14845 { markdown }
14846 { loading-latex-theme }
14847 { Loading~version~#2~of~LaTeX~Markdown~theme~#1 }
14848 \msg_new:nnn
14849 { markdown }
14850 { repeatedly-loaded-latex-theme }
14851 {
14852 Version~#3~of~LaTeX~Markdown~theme~#1~was~previously~
14853 loaded~on~line~#2,~not~loading~it~again
14854 }
14855 \msg_new:nnn
14856 { markdown }
14857 { different-versions-of-latex-theme }
14858 {
14859 Tried~to~load~version~#2~of~LaTeX~Markdown~theme~#1~

```

```

14860 but~version~#3~has~already~been~loaded~on~line~#4
14861 }
14862 \cs_generate_variant:Nn
14863 \msg_new:nnnn
14864 { nnVV }
14865 \tl_set:Nn
14866 \l_tmpa_tl
14867 { Cannot~load~LaTeX~Markdown~theme~#1~after~ }
14868 \tl_put_right:NV
14869 \l_tmpa_tl
14870 \c_backslash_str
14871 \tl_put_right:Nn
14872 \l_tmpa_tl
14873 { begin { document } }
14874 \tl_set:Nn
14875 \l_tmpb_tl
14876 { Load~Markdown~theme~#1~before~ }
14877 \tl_put_right:NV
14878 \l_tmpb_tl
14879 \c_backslash_str
14880 \tl_put_right:Nn
14881 \l_tmpb_tl
14882 { begin { document } }
14883 \msg_new:nnVV
14884 { markdown }
14885 { latex-theme-after-preamble }
14886 \l_tmpa_tl
14887 \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.

```

14888 \tl_set:Nn
14889 \l_tmpa_tl
14890 {
14891 \RequirePackage
14892 { graphicx }
14893 \markdownLoadPlainTeXTheme
14894 }
14895 \prop_gput:NnV
14896 \g_@@_latex_built_in_themes_prop
14897 { witiko / dot }
14898 \l_tmpa_tl
14899 \prop_gput:NnV
14900 \g_@@_latex_built_in_themes_prop
14901 { witiko / graphicx / http }
14902 \l_tmpa_tl
14903 \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.

```
14904 \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.

```
14905 \DeclareOption*{%
14906 \expandafter\markdownSetup\expandafter{\CurrentOption}}%
14907 \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.

```
14908 \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`.

```
14909 \ExplSyntaxOn
14910 \bool_new:N
14911 \g_@@_tight_or_fancy_lists_bool
14912 \bool_gset_false:N
14913 \g_@@_tight_or_fancy_lists_bool
14914 \@@_if_option:nTF
14915 { tightLists }
14916 {
14917 \bool_gset_true:N
14918 \g_@@_tight_or_fancy_lists_bool
14919 }
14920 {
14921 \@@_if_option:nT
14922 { fancyLists }
14923 {
14924 \bool_gset_true:N
14925 \g_@@_tight_or_fancy_lists_bool
14926 }
}
```



```

14927 }
14928 \bool_new:N
14929 \g_@@_beamer_paralist_or_enumitem_bool
14930 \bool_gset_true:N
14931 \g_@@_beamer_paralist_or_enumitem_bool
14932 \@ifclassloaded
14933 { beamer }
14934 { }
14935 {
14936 \@ifpackageloaded
14937 { paralist }
14938 { }
14939 {
14940 \@ifpackageloaded
14941 { enumitem }
14942 { }
14943 {
14944 \bool_gset_false:N
14945 \g_@@_beamer_paralist_or_enumitem_bool
14946 }
14947 }
14948 }
14949 \bool_if:nT
14950 {
14951 \g_@@_tight_or_fancy_lists_bool &&
14952 ! \g_@@_beamer_paralist_or_enumitem_bool
14953 }
14954 {
14955 \bool_if:nTF
14956 {
14957 \bool_lazy_or_p:nn
14958 {
14959 \str_if_eq_p:en
14960 { \markdownThemeVersion }
14961 { experimental }
14962 }
14963 {
14964 \bool_lazy_and_p:nn
14965 {
14966 \prop_if_exist_p:N
14967 \g__pdfmanagement_documentproperties_prop
14968 }
14969 {
14970 \bool_lazy_any_p:n
14971 {
14972 {
14973 \prop_if_in_p:Nn

```

```

14974 \g__pdfmanagement_documentproperties_prop
14975 { document / testphase / phase-I }
14976 }
14977 {
14978 \prop_if_in_p:Nn
14979 \g__pdfmanagement_documentproperties_prop
14980 { document / testphase / phase-II }
14981 }
14982 {
14983 \prop_if_in_p:Nn
14984 \g__pdfmanagement_documentproperties_prop
14985 { document / testphase / phase-III }
14986 }
14987 {
14988 \prop_if_in_p:Nn
14989 \g__pdfmanagement_documentproperties_prop
14990 { document / testphase / phase-IV }
14991 }
14992 {
14993 \prop_if_in_p:Nn
14994 \g__pdfmanagement_documentproperties_prop
14995 { document / testphase / phase-V }
14996 }
14997 {
14998 \prop_if_in_p:Nn
14999 \g__pdfmanagement_documentproperties_prop
15000 { document / testphase / phase-VI }
15001 }
15002 }
15003 }
15004 }
15005 }
15006 {
15007 \RequirePackage
15008 { enumitem }
15009 }
15010 {
15011 \RequirePackage
15012 { paralist }
15013 }
15014 }
15015 \ExplSyntaxOff

```

If we loaded the enumitem package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```

15016 \ExplSyntaxOn
15017 \cs_new:Nn

```

```

15018 \@@_latex_fancy_list_item_label_number:nn
15019 {
15020 \str_case:nn
15021 { #1 }
15022 {
15023 { Decimal } { #2 }
15024 { LowerRoman } { \int_to_roman:n { #2 } }
15025 { UpperRoman } { \int_to_Roman:n { #2 } }
15026 { LowerAlpha } { \int_to_alph:n { #2 } }
15027 { UpperAlpha } { \int_to_Alph:n { #2 } }
15028 }
15029 }
15030 \cs_new:Nn
15031 \@@_latex_fancy_list_item_label_delimiter:n
15032 {
15033 \str_case:nn
15034 { #1 }
15035 {
15036 { Default } { . }
15037 { OneParen } {) }
15038 { Period } { . }
15039 }
15040 }
15041 \cs_new:Nn
15042 \@@_latex_fancy_list_item_label:nnn
15043 {
15044 \@@_latex_fancy_list_item_label_number:nn
15045 { #1 }
15046 { #3 }
15047 \@@_latex_fancy_list_item_label_delimiter:n
15048 { #2 }
15049 }
15050 \cs_generate_variant:Nn
15051 \@@_latex_fancy_list_item_label:nnn
15052 { VVn }
15053 \tl_new:N
15054 \l_@@_latex_fancy_list_item_label_number_style_tl
15055 \tl_new:N
15056 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15057 \ifpackageloaded { enumitem } {
15058 \markdownSetup { rendererPrototypes = {
First, let's define the tight list item renderer prototypes.
15059 ulBeginTight = {
15060 \begin
15061 { itemize }
15062 [noitemsep]
15063 },

```

```

15064 ulEndTight = {
15065 \end
15066 { itemize }
15067 },
15068 olBeginTight = {
15069 \begin
15070 { enumerate }
15071 [noitemsep]
15072 },
15073 olEndTight = {
15074 \end
15075 { enumerate }
15076 },
15077 dlBeginTight = {
15078 \begin
15079 { description }
15080 [noitemsep]
15081 },
15082 dlEndTight = {
15083 \end
15084 { description }
15085 },

```

Second, let's define the fancy list item renderer prototypes.

```

15086 fancyOlBegin = {
15087 \group_begin:
15088 \tl_set:Nn
15089 \l_@@_latex_fancy_list_item_label_number_style_tl
15090 { #1 }
15091 \tl_set:Nn
15092 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15093 { #2 }
15094 \begin
15095 { enumerate }
15096 },
15097 fancyOlBeginTight = {
15098 \group_begin:
15099 \tl_set:Nn
15100 \l_@@_latex_fancy_list_item_label_number_style_tl
15101 { #1 }
15102 \tl_set:Nn
15103 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15104 { #2 }
15105 \begin
15106 { enumerate }
15107 [noitemsep]
15108 },
15109 fancyOlEnd(|Tight) = {

```

```

15110 \end { enumerate }
15111 \group_end:
15112 },
15113 fancyOItemWithNumber = {
15114 \item
15115 [
15116 \l_@@_latex_fancy_list_item_label:VVn
15117 \l_@@_latex_fancy_list_item_label_number_style_tl
15118 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15119 { #1 }
15120]
15121 },
15122 } }

```

Otherwise, if we loaded the paralist package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```

15123 }
15124 { \ifpackageloaded { paralist } {
15125 \markdownSetup { rendererPrototypes = {

```

Make tight bullet lists a little less compact by adding extra vertical space above and below them.

```

15126 ulBeginTight = {
15127 \group_begin:
15128 \pltopsep=\topsep
15129 \plpartopsep=\partopsep
15130 \begin { compactitem }
15131 },
15132 ulEndTight = {
15133 \end { compactitem }
15134 \group_end:
15135 },
15136 fancyOBegin = {
15137 \group_begin:
15138 \tl_set:Nn
15139 \l_@@_latex_fancy_list_item_label_number_style_tl
15140 { #1 }
15141 \tl_set:Nn
15142 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15143 { #2 }
15144 \begin { enumerate }
15145 },
15146 fancyOEnd = {
15147 \end { enumerate }
15148 \group_end:
15149 },

```

Make tight ordered lists a little less compact by adding extra vertical space above and below them.

```

15150 olBeginTight = {
15151 \group_begin:
15152 \plpartopsep=\partopsep
15153 \pltopsep=\topsep
15154 \begin { compactenum }
15155 },
15156 olEndTight = {
15157 \end { compactenum }
15158 \group_end:
15159 },
15160 fancyOlBeginTight = {
15161 \group_begin:
15162 \tl_set:Nn
15163 \l_@@_latex_fancy_list_item_label_number_style_tl
15164 { #1 }
15165 \tl_set:Nn
15166 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15167 { #2 }
15168 \plpartopsep=\partopsep
15169 \pltopsep=\topsep
15170 \begin { compactenum }
15171 },
15172 fancyOlEndTight = {
15173 \end { compactenum }
15174 \group_end:
15175 },
15176 fancyOlItemWithNumber = {
15177 \item
15178 [
15179 \@@_latex_fancy_list_item_label:VVn
15180 \l_@@_latex_fancy_list_item_label_number_style_tl
15181 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15182 { #1 }
15183]
15184 },

```

Make tight definition lists a little less compact by adding extra vertical space above and below them.

```

15185 dlBeginTight = {
15186 \group_begin:
15187 \plpartopsep=\partopsep
15188 \pltopsep=\topsep
15189 \begin { compactdesc }
15190 },
15191 dlEndTight = {

```

```

15192 \end { compactdesc }
15193 \group_end:
15194 }
15195 } }
15196 }
15197 {

```

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.

```

15198 \markdownSetup
15199 {
15200 rendererPrototypes = {
15201 ulBeginTight = \markdownRendererUlBegin,
15202 ulEndTight = \markdownRendererUlEnd,
15203 fancyOlBegin = \markdownRendererOlBegin,
15204 fancyOlEnd = \markdownRendererOlEnd,
15205 olBeginTight = \markdownRendererOlBegin,
15206 olEndTight = \markdownRendererOlEnd,
15207 fancyOlBeginTight = \markdownRendererOlBegin,
15208 fancyOlEndTight = \markdownRendererOlEnd,
15209 dlBeginTight = \markdownRendererDlBegin,
15210 dlEndTight = \markdownRendererDlEnd,
15211 },
15212 }
15213 } }
15214 \ExplSyntaxOff
15215 \RequirePackage{amsmath}

```

Unless the unicode-math package has been loaded, load the amssymb package with symbols to be used for tickboxes.

```

15216 \@ifpackageloaded{unicode-math}{
15217 \markdownSetup{rendererPrototypes={
15218 untickedBox = {\mdlgwhtsquare},
15219 }}
15220 }{
15221 \RequirePackage{amssymb}
15222 \markdownSetup{rendererPrototypes={
15223 untickedBox = {\square},
15224 }}
15225 }
15226 \RequirePackage{csvsimple}
15227 \RequirePackage{fancyvrb}
15228 \RequirePackage{graphicx}
15229 \markdownSetup{rendererPrototypes={
15230 hardLineBreak = {\},
15231 leftBrace = {\textbraceleft},
15232 rightBrace = {\textbraceright},

```

```

15233 dollarSign = {\textdollar},
15234 underscore = {\textunderscore},
15235 circumflex = {\textasciicircum},
15236 backslash = {\textbackslash},
15237 tilde = {\textasciitilde},
15238 pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by T<sub>E</sub>X during the typesetting. Therefore, even if we don't know whether a span of text is part of math formula or not when we are parsing markdown,<sup>34</sup> we can reliably detect math mode inside the renderer.

Here, we will redefine the code span renderer prototype to typeset upright text in math formulae and typewriter text outside math formulae.

```

15239 codeSpan = {%
15240 \ifmmode
15241 \text{#1}%
15242 \else
15243 \texttt{#1}%
15244 \fi
15245 }%

```

### 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.

```

15246 \ExplSyntaxOn
15247 \markdownSetup{
15248 rendererPrototypes = {
15249 contentBlock = {
15250 \str_case:nnF
15251 { #1 }
15252 {
15253 { csv }
15254 {
15255 \begin { table }
15256 \begin { center }
15257 \csvautotabular { #3 }
15258 \end{ center }
15259 \tl_if_empty:nF
15260 { #4 }

```

---

<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.



```

15261 { \caption { #4 } }
15262 \end { table }
15263 }
15264 { html }
15265 {

```

If we are using  $\text{T}_{\text{E}}\text{X}4\text{ht}$ <sup>35</sup>, we will pass HTML elements to the output HTML document unchanged.

```

15266 \cs_if_exist:NTF
15267 \HCode
15268 {
15269 \if_mode_vertical:
15270 \IgnorePar
15271 \fi:
15272 \EndP
15273 \special
15274 { t4ht* < #3 }
15275 \par
15276 \ShowPar
15277 }
15278 {
15279 \@_luaxml_print_html:n
15280 { #3 }
15281 }
15282 }
15283 { tex }
15284 {
15285 \markdownEscape
15286 { #3 }
15287 }
15288 }
15289 {
15290 \markdownInput
15291 { #3 }
15292 }
15293 },
15294 },
15295 }
15296 \ExplSyntaxOff
15297 \markdownSetup{rendererPrototypes={
15298 ulBegin = {\begin{itemize}},
15299 ulEnd = {\end{itemize}},
15300 olBegin = {\begin{enumerate}},
15301 olItem = {\item{}},
15302 olItemWithNumber = {\item[#1.]},
15303 olEnd = {\end{enumerate}},

```

<sup>35</sup>See <https://tug.org/tex4ht/>.

```

15304 dlBegin = {\begin{description}},
15305 dlItem = {\item[#1]},
15306 dlEnd = {\end{description}},
15307 emphasis = {\emph{#1}},
15308 tickedBox = {\boxtimes$},
15309 halfTickedBox = {\boxdot$}}

```

If HTML identifiers appear after a heading, we make them produce `\label` macros.

```

15310 \ExplSyntaxOn
15311 \seq_new:N
15312 \g_@@_header_identifiers_seq
15313 \markdownSetup
15314 {
15315 rendererPrototypes = {
15316 headerAttributeContextBegin = {
15317 \markdownSetup
15318 {
15319 rendererPrototypes = {
15320 attributeIdentifier = {
15321 \seq_gput_right:Nn
15322 \g_@@_header_identifiers_seq
15323 { ##1 }
15324 },
15325 },
15326 }
15327 },
15328 headerAttributeContextEnd = {
15329 \seq_map_inline:Nn
15330 \g_@@_header_identifiers_seq
15331 { \label { ##1 } }
15332 \seq_gclear:N
15333 \g_@@_header_identifiers_seq
15334 },
15335 },
15336 }

```

If the `unnumbered` HTML class (or the `{-}` shorthand) appears after a heading the heading and all its subheadings will be unnumbered.

```

15337 \bool_new:N
15338 \l_@@_header_unnumbered_bool
15339 \markdownSetup
15340 {
15341 rendererPrototypes = {
15342 headerAttributeContextBegin += {
15343 \markdownSetup
15344 {
15345 rendererPrototypes = {
15346 attributeClassName = {

```

```

15347 \bool_if:nT
15348 {
15349 \str_if_eq_p:nn
15350 { ##1 }
15351 { unnumbered } &&
15352 ! \l_@@_header_unnumbered_bool
15353 }
15354 {
15355 \group_begin:
15356 \bool_set_true:N
15357 \l_@@_header_unnumbered_bool
15358 \c@secnumdepth = 0
15359 \markdownSetup
15360 {
15361 rendererPrototypes = {
15362 sectionBegin = {
15363 \group_begin:
15364 },
15365 sectionEnd = {
15366 \group_end:
15367 },
15368 },
15369 }
15370 }
15371 },
15372 },
15373 }
15374 },
15375 },
15376 }
15377 \ExplSyntaxOff
15378 \markdownSetup{rendererPrototypes={
15379 superscript = {#1},
15380 subscript = {\textsubscript{#1}},
15381 blockQuoteBegin = {\begin{quotation}},
15382 blockQuoteEnd = {\end{quotation}},
15383 inputVerbatim = {\VerbatimInput{#1}},
15384 thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
15385 note = {\footnote{#1}}}}

```

### 3.3.4.3 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

15386 \RequirePackage{ltxcmds}
15387 \ExplSyntaxOn
15388 \cs_gset_protected:Npn
15389 \markdownRendererInputFencedCodePrototype#1#2#3

```

```

15390 {
15391 \tl_if_empty:nTF
15392 { #2 }
15393 { \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.

```

15394 {
15395 \regex_extract_once:nnN
15396 { \w* }
15397 { #2 }
15398 \l_tmpa_seq
15399 \seq_pop_left:NN
15400 \l_tmpa_seq
15401 \l_tmpa_tl

```

When the minted package is loaded, use it for syntax highlighting.

```

15402 \ltx@ifpackageloaded
15403 { minted }
15404 {
15405 \catcode`\%=14\relax
15406 \catcode`\#=6\relax
15407 \exp_args:NV
15408 \inputminted
15409 \l_tmpa_tl
15410 { #1 }
15411 \catcode`\%=12\relax
15412 \catcode`\#=12\relax
15413 }
15414 {

```

When the listings package is loaded, use it for syntax highlighting.

```

15415 \ltx@ifpackageloaded
15416 { listings }
15417 { \lstinputlisting [language = \l_tmpa_tl] { #1 } }

```

When neither the listings package nor the minted package is loaded, act as though no infostring were given.

```

15418 { \markdownRendererInputFencedCode { #1 } { } { } }
15419 }
15420 }
15421 }
15422 \ExplSyntaxOff

```

Support the nesting of strong emphasis.

```

15423 \ExplSyntaxOn
15424 \def\markdownLATEXStrongEmphasis#1{
15425 \str_if_in:NnTF
15426 \f@series

```

```

15427 { b }
15428 { \textnormal{#1} }
15429 { \textbf{#1} }
15430 }
15431 \ExplSyntaxOff
15432 \markdownSetup{rendererPrototypes={strongEmphasis={%
15433 \protect\markdownLATEXStrongEmphasis{#1}}}}

```

Support L<sup>A</sup>T<sub>E</sub>X document classes that do not provide chapters.

```

15434 \@ifundefined{chapter}{%
15435 \markdownSetup{rendererPrototypes = {
15436 headingOne = {\section{#1}},
15437 headingTwo = {\subsection{#1}},
15438 headingThree = {\subsubsection{#1}},
15439 headingFour = {\paragraph{#1}},
15440 headingFive = {\subparagraph{#1}}}}
15441 }{%
15442 \markdownSetup{rendererPrototypes = {
15443 headingOne = {\chapter{#1}},
15444 headingTwo = {\section{#1}},
15445 headingThree = {\subsection{#1}},
15446 headingFour = {\subsubsection{#1}},
15447 headingFive = {\paragraph{#1}},
15448 headingSix = {\subparagraph{#1}}}}
15449 }%

```

### 3.3.4.4 Tickboxes

If the `taskLists` option is enabled, we will hide bullets in unordered list items with tickboxes.

```

15450 \markdownSetup{
15451 rendererPrototypes = {
15452 ulItem = {%
15453 \futurelet\markdownLaTeXCheckbox\markdownLaTeXUItem
15454 },
15455 },
15456 }
15457 \def\markdownLaTeXUItem{%
15458 \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
15459 \item[\markdownLaTeXCheckbox]%
15460 \expandafter\@gobble
15461 \else
15462 \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
15463 \item[\markdownLaTeXCheckbox]%
15464 \expandafter\expandafter\expandafter\@gobble
15465 \else
15466 \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
15467 \item[\markdownLaTeXCheckbox]%

```

```

15468 \expandafter\expandafter\expandafter\expandafter
15469 \expandafter\expandafter\expandafter\@gobble
15470 \else
15471 \item{}%
15472 \fi
15473 \fi
15474 \fi
15475 }

```

### 3.3.4.5 HTML elements

If the `html` option is enabled and we are using `TeX4ht`<sup>36</sup>, we will pass HTML elements to the output HTML document unchanged.

```

15476 \@ifundefined{HCode}{}{
15477 \markdownSetup{
15478 rendererPrototypes = {
15479 inlineHtmlTag = {%
15480 \ifvmode
15481 \IgnorePar
15482 \EndP
15483 \fi
15484 \HCode{#1}%
15485 },
15486 inputBlockHtmlElement = {%
15487 \ifvmode
15488 \IgnorePar
15489 \fi
15490 \EndP
15491 \special{t4ht*<#1}%
15492 \par
15493 \ShowPar
15494 },
15495 },
15496 }
15497 }

```

### 3.3.4.6 Citations

Here is a basic implementation for citations that uses the `LATEX` `\cite` macro. There are also implementations that use the `natbib` `\citep`, and `\citet` macros, and the `BibLATEX` `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```

15498 \newcount\markdownLaTeXCitationsCounter
15499
15500 % Basic implementation

```

---

<sup>36</sup>See <https://tug.org/tex4ht/>.

```

15501 \long\def@gobblethree#1#2#3{%
15502 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
15503 \advance\markdownLaTeXCitationsCounter by 1\relax
15504 \ifx\relax#4\relax
15505 \ifx\relax#5\relax
15506 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15507 \relax
15508 \cite{#1#2#6}% No prenotes/postnotes, just accumulate cites
15509 \expandafter\expandafter\expandafter
15510 \expandafter\expandafter\expandafter\expandafter
15511 \@gobblethree
15512 \fi
15513 \else% Before a postnote (#5), dump the accumulator
15514 \ifx\relax#1\relax\else
15515 \cite{#1}%
15516 \fi
15517 \cite[#5]{#6}%
15518 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15519 \relax
15520 \else
15521 \expandafter\expandafter\expandafter
15522 \expandafter\expandafter\expandafter\expandafter
15523 \expandafter\expandafter\expandafter
15524 \expandafter\expandafter\expandafter\expandafter
15525 \markdownLaTeXBasicCitations
15526 \fi
15527 \expandafter\expandafter\expandafter
15528 \expandafter\expandafter\expandafter\expandafter{%
15529 \expandafter\expandafter\expandafter
15530 \expandafter\expandafter\expandafter\expandafter}%
15531 \expandafter\expandafter\expandafter
15532 \expandafter\expandafter\expandafter\expandafter{%
15533 \expandafter\expandafter\expandafter
15534 \expandafter\expandafter\expandafter\expandafter}%
15535 \expandafter\expandafter\expandafter
15536 \@gobblethree
15537 \fi
15538 \else% Before a prenote (#4), dump the accumulator
15539 \ifx\relax#1\relax\else
15540 \cite{#1}%
15541 \fi
15542 \ifnum\markdownLaTeXCitationsCounter>1\relax
15543 \space % Insert a space before the prenote in later citations
15544 \fi
15545 #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
15546 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15547 \relax

```

```

15548 \else
15549 \expandafter\expandafter\expandafter
15550 \expandafter\expandafter\expandafter\expandafter
15551 \markdownLaTeXBasicCitations
15552 \fi
15553 \expandafter\expandafter\expandafter{%
15554 \expandafter\expandafter\expandafter}%
15555 \expandafter\expandafter\expandafter{%
15556 \expandafter\expandafter\expandafter}%
15557 \expandafter
15558 \@gobblethree
15559 \fi\markdownLaTeXBasicCitations{#1#2#6},}
15560 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
15561
15562 % Natbib implementation
15563 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
15564 \advance\markdownLaTeXCitationsCounter by 1\relax
15565 \ifx\relax#3\relax
15566 \ifx\relax#4\relax
15567 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15568 \relax
15569 \citep{#1,#5}% No prenotes/postnotes, just accumulate cites
15570 \expandafter\expandafter\expandafter
15571 \expandafter\expandafter\expandafter\expandafter
15572 \@gobbletwo
15573 \fi
15574 \else% Before a postnote (#4), dump the accumulator
15575 \ifx\relax#1\relax\else
15576 \citep{#1}%
15577 \fi
15578 \citep[] [#4]{#5}%
15579 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15580 \relax
15581 \else
15582 \expandafter\expandafter\expandafter
15583 \expandafter\expandafter\expandafter\expandafter
15584 \expandafter\expandafter\expandafter
15585 \expandafter\expandafter\expandafter\expandafter
15586 \markdownLaTeXNatbibCitations
15587 \fi
15588 \expandafter\expandafter\expandafter
15589 \expandafter\expandafter\expandafter\expandafter{%
15590 \expandafter\expandafter\expandafter
15591 \expandafter\expandafter\expandafter\expandafter}%
15592 \expandafter\expandafter\expandafter
15593 \@gobbletwo
15594 \fi

```



```

15595 \else% Before a prenote (#3), dump the accumulator
15596 \ifx\relax#1\relax\relax\else
15597 \citep{#1}%
15598 \fi
15599 \citep[#3][#4]{#5}%
15600 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15601 \relax
15602 \else
15603 \expandafter\expandafter\expandafter
15604 \expandafter\expandafter\expandafter\expandafter
15605 \markdownLaTeXNatbibCitations
15606 \fi
15607 \expandafter\expandafter\expandafter{%
15608 \expandafter\expandafter\expandafter}%
15609 \expandafter
15610 \@gobbletwo
15611 \fi\markdownLaTeXNatbibCitations{#1,#5}}
15612 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
15613 \advance\markdownLaTeXCitationsCounter by 1\relax
15614 \ifx\relax#3\relax
15615 \ifx\relax#4\relax
15616 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15617 \relax
15618 \citet{#1,#5}% No prenotes/postnotes, just accumulate cites
15619 \expandafter\expandafter\expandafter
15620 \expandafter\expandafter\expandafter\expandafter
15621 \@gobbletwo
15622 \fi
15623 \else% After a prenote or a postnote, dump the accumulator
15624 \ifx\relax#1\relax\else
15625 \citet{#1}%
15626 \fi
15627 , \citet[#3][#4]{#5}%
15628 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
15629 \relax
15630 ,
15631 \else
15632 \ifnum
15633 \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
15634 \relax
15635 ,
15636 \fi
15637 \fi
15638 \expandafter\expandafter\expandafter
15639 \expandafter\expandafter\expandafter\expandafter
15640 \markdownLaTeXNatbibTextCitations
15641 \expandafter\expandafter\expandafter

```

```

15642 \expandafter\expandafter\expandafter\expandafter{%
15643 \expandafter\expandafter\expandafter
15644 \expandafter\expandafter\expandafter\expandafter}%
15645 \expandafter\expandafter\expandafter
15646 \@gobbletwo
15647 \fi
15648 \else% After a prenote or a postnote, dump the accumulator
15649 \ifx\relax#1\relax\relax\else
15650 \citet{#1}%
15651 \fi
15652 , \citet[#3][#4]{#5}%
15653 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
15654 \relax
15655 ,
15656 \else
15657 \ifnum
15658 \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
15659 \relax
15660 ,
15661 \fi
15662 \fi
15663 \expandafter\expandafter\expandafter
15664 \markdownLaTeXNatbibTextCitations
15665 \expandafter\expandafter\expandafter{%
15666 \expandafter\expandafter\expandafter}%
15667 \expandafter
15668 \@gobbletwo
15669 \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
15670
15671 % BibLaTeX implementation
15672 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
15673 \advance\markdownLaTeXCitationsCounter by 1\relax
15674 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15675 \relax
15676 \autocites#1[#3][#4]{#5}%
15677 \expandafter\@gobbletwo
15678 \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
15679 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
15680 \advance\markdownLaTeXCitationsCounter by 1\relax
15681 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15682 \relax
15683 \textcites#1[#3][#4]{#5}%
15684 \expandafter\@gobbletwo
15685 \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}}
15686
15687 \markdownSetup{rendererPrototypes = {
15688 cite = {}

```

```

15689 \markdownLaTeXCitationsCounter=1%
15690 \def\markdownLaTeXCitationsTotal{#1}%
15691 \@ifundefined{autocites}{%
15692 \@ifundefined{citep}{%
15693 \expandafter\expandafter\expandafter
15694 \markdownLaTeXBasicCitations
15695 \expandafter\expandafter\expandafter{%
15696 \expandafter\expandafter\expandafter}%
15697 \expandafter\expandafter\expandafter{%
15698 \expandafter\expandafter\expandafter}%
15699 }{%
15700 \expandafter\expandafter\expandafter
15701 \markdownLaTeXNatbibCitations
15702 \expandafter\expandafter\expandafter{%
15703 \expandafter\expandafter\expandafter}%
15704 }%
15705 }{%
15706 \expandafter\expandafter\expandafter
15707 \markdownLaTeXBibLaTeXCitations
15708 \expandafter{\expandafter}%
15709 }},
15710 textCite = {%
15711 \markdownLaTeXCitationsCounter=1%
15712 \def\markdownLaTeXCitationsTotal{#1}%
15713 \@ifundefined{autocites}{%
15714 \@ifundefined{citep}{%
15715 \expandafter\expandafter\expandafter
15716 \markdownLaTeXBasicTextCitations
15717 \expandafter\expandafter\expandafter{%
15718 \expandafter\expandafter\expandafter}%
15719 \expandafter\expandafter\expandafter{%
15720 \expandafter\expandafter\expandafter}%
15721 }{%
15722 \expandafter\expandafter\expandafter
15723 \markdownLaTeXNatbibTextCitations
15724 \expandafter\expandafter\expandafter{%
15725 \expandafter\expandafter\expandafter}%
15726 }%
15727 }{%
15728 \expandafter\expandafter\expandafter
15729 \markdownLaTeXBibLaTeXTextCitations
15730 \expandafter{\expandafter}%
15731 }}}

```

### 3.3.4.7 Links

Here is an implementation for hypertext links and relative references.

```

15732 \RequirePackage{url}
15733 \RequirePackage{expl3}
15734 \ExplSyntaxOn
15735 \cs_gset_protected:Npn
15736 \markdownRendererLinkPrototype
15737 #1#2#3#4
15738 {
15739 \tl_set:Nn \l_tmpa_tl { #1 }
15740 \tl_set:Nn \l_tmpb_tl { #2 }
15741 \bool_set:Nn
15742 \l_tmpa_bool
15743 {
15744 \tl_if_eq_p:NN
15745 \l_tmpa_tl
15746 \l_tmpb_tl
15747 }
15748 \tl_set:Nn \l_tmpa_tl { #4 }
15749 \bool_set:Nn
15750 \l_tmpb_bool
15751 {
15752 \tl_if_empty_p:N
15753 \l_tmpa_tl
15754 }

```

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.

```

15755 \bool_if:nTF
15756 {
15757 \l_tmpa_bool && \l_tmpb_bool
15758 }
15759 {
15760 \markdownLaTeXRendererAutolink { #2 } { #3 }
15761 }
15762 {
15763 \markdownLaTeXRendererDirectOrIndirectLink
15764 { #1 } { #2 } { #3 } { #4 }
15765 }
15766 }
15767 \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.

```

15768 \tl_set:Nn
15769 \l_tmpa_tl
15770 { #2 }
15771 \tl_trim_spaces:N
15772 \l_tmpa_tl

```

```

15773 \tl_set:Nx
15774 \l_tmpb_tl
15775 {
15776 \tl_range:Nnn
15777 \l_tmpa_tl
15778 { 1 }
15779 { 1 }
15780 }
15781 \str_if_eq:NNTF
15782 \l_tmpb_tl
15783 \c_hash_str
15784 {
15785 \tl_set:Nx
15786 \l_tmpb_tl
15787 {
15788 \tl_range:Nnn
15789 \l_tmpa_tl
15790 { 2 }
15791 { -1 }
15792 }
15793 \exp_args:NV
15794 \ref
15795 \l_tmpb_tl
15796 }
15797 {
15798 \url { #2 }
15799 }
15800 }
15801 \ExplSyntaxOff
15802 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
15803 #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.

```

15804 \newcount\markdownLaTeXRowCount
15805 \newcount\markdownLaTeXRowTotal
15806 \newcount\markdownLaTeXColumnCounter
15807 \newcount\markdownLaTeXColumnTotal
15808 \newtoks\markdownLaTeXTable
15809 \newtoks\markdownLaTeXTableAlignment
15810 \newtoks\markdownLaTeXTableEnd
15811 \AtBeginDocument{%
15812 \ifpackageloaded{booktabs}{%
15813 \def\markdownLaTeXTopRule{\toprule}%
15814 \def\markdownLaTeXMidRule{\midrule}%

```

```

15815 \def\markdownLaTeXBottomRule{\bottomrule}%
15816 }{%
15817 \def\markdownLaTeXTopRule{\hline}%
15818 \def\markdownLaTeXMidRule{\hline}%
15819 \def\markdownLaTeXBottomRule{\hline}%
15820 }%
15821 }
15822 \markdownSetup{rendererPrototypes={
15823 table = {%
15824 \markdownLaTeXTable={}%
15825 \markdownLaTeXTableAlignment={}%
15826 \markdownLaTeXTableEnd={%
15827 \markdownLaTeXBottomRule
15828 \end{tabular}}%
15829 \ifx\empty#1\empty\else
15830 \addto@hook\markdownLaTeXTable{%
15831 \begin{table}
15832 \centering}%
15833 \addto@hook\markdownLaTeXTableEnd{%
15834 \caption{#1}}%
15835 \fi
15836 }
15837 }}

```

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.

```

15838 \ExplSyntaxOn
15839 \seq_new:N
15840 \l_@@_table_identifiers_seq
15841 \markdownSetup {
15842 rendererPrototypes = {
15843 table += {
15844 \seq_map_inline:Nn
15845 \l_@@_table_identifiers_seq
15846 {
15847 \addto@hook
15848 \markdownLaTeXTableEnd
15849 { \label { ##1 } }
15850 }
15851 },
15852 }
15853 }
15854 \markdownSetup {
15855 rendererPrototypes = {
15856 tableAttributeContextBegin = {
15857 \group_begin:
15858 \markdownSetup {

```

```

15859 rendererPrototypes = {
15860 attributeIdentifier = {
15861 \seq_put_right:Nn
15862 \l_@@_table_identifiers_seq
15863 { ##1 }
15864 },
15865 },
15866 }
15867 },
15868 tableAttributeContextEnd = {
15869 \group_end:
15870 },
15871 },
15872 }
15873 \ExplSyntaxOff
15874 \markdownSetup{rendererPrototypes={
15875 table += {%
15876 \ifx\empty#1\empty\else
15877 \addto@hook\markdownLaTeXTableEnd{%
15878 \end{table}}%
15879 \fi
15880 \addto@hook\markdownLaTeXTable{\begin{tabular}}%
15881 \markdownLaTeXRowCounter=0%
15882 \markdownLaTeXRowTotal=#2%
15883 \markdownLaTeXColumnTotal=#3%
15884 \markdownLaTeXRenderTableRow
15885 }
15886 }}
15887 \def\markdownLaTeXRenderTableRow#1{%
15888 \markdownLaTeXColumnCounter=0%
15889 \ifnum\markdownLaTeXRowCounter=0\relax
15890 \markdownLaTeXReadAlignments#1%
15891 \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
15892 \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
15893 \the\markdownLaTeXTableAlignment}}%
15894 \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
15895 \else
15896 \markdownLaTeXRenderTableCell#1%
15897 \fi
15898 \ifnum\markdownLaTeXRowCounter=1\relax
15899 \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
15900 \fi
15901 \advance\markdownLaTeXRowCounter by 1\relax
15902 \ifnum\markdownLaTeXRowCounter>\markdownLaTeXRowTotal\relax
15903 \the\markdownLaTeXTable
15904 \the\markdownLaTeXTableEnd
15905 \expandafter\@gobble

```

```

15906 \fi\markdownLaTeXRenderTableRow}
15907 \def\markdownLaTeXReadAlignments#1{%
15908 \advance\markdownLaTeXColumnCounter by 1\relax
15909 \if#1d%
15910 \addto@hook\markdownLaTeXTableAlignment{1}%
15911 \else
15912 \addto@hook\markdownLaTeXTableAlignment{#1}%
15913 \fi
15914 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
15915 \expandafter\@gobble
15916 \fi\markdownLaTeXReadAlignments}
15917 \def\markdownLaTeXRenderTableCell#1{%
15918 \advance\markdownLaTeXColumnCounter by 1\relax
15919 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
15920 \addto@hook\markdownLaTeXTable{#1&}%
15921 \else
15922 \addto@hook\markdownLaTeXTable{#1\\}%
15923 \expandafter\@gobble
15924 \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.

```

15925
15926 \markdownIfOption{lineBlocks}{%
15927 \RequirePackage{verse}
15928 \markdownSetup{rendererPrototypes={
15929 lineBlockBegin = {%
15930 \begingroup
15931 \def\markdownRendererHardLineBreak{\\}%
15932 \begin{verse}%
15933 },
15934 lineBlockEnd = {%
15935 \end{verse}%
15936 \endgroup
15937 },
15938 }}
15939 }{}
15940

```

### 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.

```

15941 \ExplSyntaxOn

```



```

15942 \keys_define:nn
15943 { markdown / jekyllData }
15944 {
15945 author .code:n = {
15946 \author
15947 { #1 }
15948 },
15949 date .code:n = {
15950 \date
15951 { #1 }
15952 },
15953 title .code:n = {
15954 \title
15955 { #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.

```

15956 \AddToHook
15957 { begindocument / end }
15958 { \maketitle }
15959 },
15960 }

```

### 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.

```

15961 \@@_if_option:nT
15962 { mark }
15963 {
15964 \sys_if_engine_luatex:TF
15965 {
15966 \RequirePackage
15967 { luacolor }
15968 \RequirePackage
15969 { lua-ul }
15970 \markdownSetup
15971 {
15972 rendererPrototypes = {
15973 mark = {
15974 \highLight
15975 { #1 }
15976 },
15977 }
15978 }

```

```

15979 }
15980 {
15981 \RequirePackage
15982 { xcolor }
15983 \RequirePackage
15984 { soul }
15985 \markdownSetup
15986 {
15987 rendererPrototypes = {
15988 mark = {
15989 \hl
15990 { #1 }
15991 },
15992 }
15993 }
15994 }
15995 }

```

### 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.

```

15996 \@@_if_option:nT
15997 { strikeThrough }
15998 {
15999 \sys_if_engine luatex:TF
16000 {
16001 \RequirePackage
16002 { lua-ul }
16003 \markdownSetup
16004 {
16005 rendererPrototypes = {
16006 strikeThrough = {
16007 \strikeThrough
16008 { #1 }
16009 },
16010 }
16011 }
16012 }
16013 {
16014 \RequirePackage
16015 { soul }
16016 \markdownSetup
16017 {
16018 rendererPrototypes = {
16019 strikeThrough = {
16020 \st

```

```

16021 { #1 }
16022 },
16023 }
16024 }
16025 }
16026 }

```

### 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 `<key>=<value>` 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  $\LaTeX$  labels for referencing figures.

```

16027 \seq_new:N
16028 \l_@@_image_identifiers_seq
16029 \markdownSetup {
16030 rendererPrototypes = {
16031 image = {
16032 \tl_if_empty:nTF
16033 { #4 }
16034 {
16035 \begin { center }
16036 \includegraphics
16037 [alt = { #1 }]
16038 { #3 }
16039 \end { center }
16040 }
16041 {
16042 \begin { figure }
16043 \begin { center }
16044 \includegraphics
16045 [alt = { #1 }]
16046 { #3 }
16047 \caption { #4 }
16048 \seq_map_inline:Nn
16049 \l_@@_image_identifiers_seq
16050 { \label { ##1 } }
16051 \end { center }
16052 \end { figure }
16053 }
16054 },
16055 }
16056 }
16057 \@@_if_option:nT

```

```

16058 { linkAttributes }
16059 {
16060 \RequirePackage { graphicx }
16061 }
16062 \markdownSetup {
16063 rendererPrototypes = {
16064 imageAttributeContextBegin = {
16065 \group_begin:
16066 \markdownSetup {
16067 rendererPrototypes = {
16068 attributeIdentifier = {
16069 \seq_put_right:Nn
16070 \l_@@_image_identifiers_seq
16071 { ##1 }
16072 },
16073 attributeKeyValue = {
16074 \setkeys
16075 { Gin }
16076 { { ##1 } = { ##2 } }
16077 },
16078 },
16079 }
16080 },
16081 imageAttributeContextEnd = {
16082 \group_end:
16083 },
16084 },
16085 }
16086 \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`.

```

16087 \ExplSyntaxOn
16088 \cs_new:Nn
16089 \@@_luaxml_print_html:n
16090 {
16091 \luabridge_now:n
16092 {
16093 local~input_file = assert(io.open(" #1 ", "r"))
16094 local~input = assert(input_file:read("*a"))
16095 assert(input_file:close())
16096 input = "<body>" .. input .. "</body>"
16097 local~dom = require("luaxml-domobject").html_parse(input)

```

```

16098 local~output = require("luaxml-htmltemplates"):process_dom(dom)
16099 print(output)
16100 }
16101 }
16102 \cs_gset_protected:Npn
16103 \markdownRendererInputRawInlinePrototype#1#2
16104 {
16105 \str_case:nnF
16106 { #2 }
16107 {
16108 { latex }
16109 {
16110 \@@_plain_tex_default_input_raw_inline:nn
16111 { #1 }
16112 { tex }
16113 }
16114 { html }
16115 {

```

If we are using T<sub>E</sub>X4ht<sup>37</sup>, we will pass HTML elements to the output HTML document unchanged.

```

16116 \cs_if_exist:NTF
16117 \HCode
16118 {
16119 \if_mode_vertical:
16120 \IgnorePar
16121 \EndP
16122 \fi:
16123 \special
16124 { t4ht* < #1 }
16125 }
16126 {
16127 \@@_luaxml_print_html:n
16128 { #1 }
16129 }
16130 }
16131 }
16132 {
16133 \@@_plain_tex_default_input_raw_inline:nn
16134 { #1 }
16135 { #2 }
16136 }
16137 }
16138 \cs_gset_protected:Npn
16139 \markdownRendererInputRawBlockPrototype#1#2
16140 {

```

---

<sup>37</sup>See <https://tug.org/tex4ht/>.

```

16141 \str_case:nnF
16142 { #2 }
16143 {
16144 { latex }
16145 {
16146 \@@_plain_tex_default_input_raw_block:nn
16147 { #1 }
16148 { tex }
16149 }
16150 { html }
16151 {

```

If we are using  $\text{T}\text{E}\text{X}4\text{ht}$ <sup>38</sup>, we will pass HTML elements to the output HTML document unchanged.

```

16152 \cs_if_exist:NTF
16153 \HCode
16154 {
16155 \if_mode_vertical:
16156 \IgnorePar
16157 \fi:
16158 \EndP
16159 \special
16160 { t4ht* < #1 }
16161 \par
16162 \ShowPar
16163 }
16164 {
16165 \@@_luaxml_print_html:n
16166 { #1 }
16167 }
16168 }
16169 }
16170 {
16171 \@@_plain_tex_default_input_raw_block:nn
16172 { #1 }
16173 { #2 }
16174 }
16175 }

```

### 3.3.4.15 Bracketed spans

If the `bracketedSpans` option is enabled, we will register any identifiers, so that they can be used as  $\text{L}\text{A}\text{T}\text{E}\text{X}$  labels for referencing the last  $\text{L}\text{A}\text{T}\text{E}\text{X}$  counter that has been incremented in e.g. ordered lists.

```

16176 \seq_new:N
16177 \l_@@_bracketed_span_identifiers_seq

```

---

<sup>38</sup>See <https://tug.org/tex4ht/>.

```

16178 \markdownSetup {
16179 rendererPrototypes = {
16180 bracketedSpanAttributeContextBegin = {
16181 \group_begin:
16182 \markdownSetup {
16183 rendererPrototypes = {
16184 attributeIdentifier = {
16185 \seq_put_right:Nn
16186 \l_@@_bracketed_span_identifiers_seq
16187 { ##1 }
16188 },
16189 },
16190 }
16191 },
16192 bracketedSpanAttributeContextEnd = {
16193 \seq_map_inline:Nn
16194 \l_@@_bracketed_span_identifiers_seq
16195 { \label { ##1 } }
16196 \group_end:
16197 },
16198 },
16199 }
16200 \ExplSyntaxOff
16201 \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.

```

16202 \newcommand\markdownMakeOther{%
16203 \count0=128\relax
16204 \loop
16205 \catcode\count0=11\relax
16206 \advance\count0 by 1\relax
16207 \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.

```
16208 \def\markdownMakeOther{%
16209 \count0=128\relax
16210 \loop
16211 \catcode\count0=11\relax
16212 \advance\count0 by 1\relax
16213 \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.

```
16214 \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).

```
16215 \long\def\inputmarkdown{%
16216 \dosingleempty
16217 \doinputmarkdown}%
16218 \long\def\doinputmarkdown[#1]#2{%
16219 \begingroup
16220 \iffirstargument
16221 \setupmarkdown[#1]%
16222 \fi
16223 \markdownInput{#2}%
16224 \endgroup}%
16225 \long\def\inputyaml{%
16226 \dosingleempty
16227 \doinputyaml}%
16228 \long\def\doinputyaml[#1]#2{%
16229 \doinputmarkdown
16230 [jekyllData, expectJekyllData, ensureJekyllData, #1]{#2}}%
```

The `\startmarkdown`, `\stopmarkdown`, `\startyaml`, and `\stopyaml` macros are implemented using the `\markdownReadAndConvert` macro.

In Knuth’s T<sub>E</sub>X, trailing spaces are removed very early on when a line is being put to the input buffer. [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)T<sub>E</sub>X, but ConT<sub>E</sub>Xt MkIV funnels all input through its own input handler. This makes it possible to suppress the removal of trailing spaces in ConT<sub>E</sub>Xt MkIV and therefore to insert hard line breaks into markdown text.

```
16231 \startluacode
16232 document.markdown_buffering = false
```



```

16233 local function preserve_trailing_spaces(line)
16234 if document.markdown_buffering then
16235 line = line:gsub("[\t][\t]$", "\t\t")
16236 end
16237 return line
16238 end
16239 resolvers.installinputlinehandler(preserve_trailing_spaces)
16240 \stoptluacode
16241 \begingroup
16242 \catcode\|=0%
16243 \catcode\|=12%
16244 |gdef|startmarkdown{%
16245 |ctxlua{document.markdown_buffering = true}%
16246 |markdownReadAndConvert{\stopmarkdown}%
16247 {|stopmarkdown}}%
16248 |gdef|stopmarkdown{%
16249 |ctxlua{document.markdown_buffering = false}%
16250 |markdownEnd}%
16251 |gdef|startyaml{%
16252 |begingroup
16253 |ctxlua{document.markdown_buffering = true}%
16254 |setupyaml[jekyllData, expectJekyllData, ensureJekyllData]%
16255 |markdownReadAndConvert{\stopyaml}%
16256 {|stopyaml}}%
16257 |gdef|stopyaml{%
16258 |ctxlua{document.markdown_buffering = false}%
16259 |yamlEnd}%
16260 |endgroup

```

### 3.4.2 Themes

This section overrides the plain  $\text{T}_{\text{E}}\text{X}$  implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in  $\text{ConT}_{\text{E}}\text{Xt}$  themes provided with the Markdown package.

```

16261 \ExplSyntaxOn
16262 \prop_new:N \g_@@_context_loaded_themes_linenos_prop
16263 \prop_new:N \g_@@_context_loaded_themes_versions_prop
16264 \cs_gset:Nn
16265 \@@_load_theme:nnn
16266 {

```

Determine whether either this is a built-in theme according to the prop `\g_@@_context_built_in_themes_prop` or a file named `t-markdowntheme<munged theme name>.tex` exists. If it does, load it. Otherwise, try loading a plain  $\text{T}_{\text{E}}\text{X}$  theme instead.

```

16267 \bool_if:nTF

```

```

16268 {
16269 \bool_lazy_or_p:nn
16270 {
16271 \prop_if_in_p:Nn
16272 \g_@@_context_built_in_themes_prop
16273 { #1 }
16274 }
16275 {
16276 \file_if_exist_p:n
16277 { t - markdown theme #3.tex }
16278 }
16279 }
16280 {
16281 \prop_get:NnNTF
16282 \g_@@_context_loaded_themes_linenos_prop
16283 { #1 }
16284 \l_tmpa_tl
16285 {
16286 \prop_get:NnN
16287 \g_@@_context_loaded_themes_versions_prop
16288 { #1 }
16289 \l_tmpb_tl
16290 \str_if_eq:nVTF
16291 { #2 }
16292 \l_tmpb_tl
16293 {
16294 \msg_warning:nnnVn
16295 { markdown }
16296 { repeatedly-loaded-context-theme }
16297 { #1 }
16298 \l_tmpa_tl
16299 { #2 }
16300 }
16301 {
16302 \msg_error:nnnnVV
16303 { markdown }
16304 { different-versions-of-context-theme }
16305 { #1 }
16306 { #2 }
16307 \l_tmpb_tl
16308 \l_tmpa_tl
16309 }
16310 }
16311 }
16312 \prop_gput:Nnx
16313 \g_@@_context_loaded_themes_linenos_prop
16314 { #1 }

```

```

16315 { \tex_the:D \tex_inputlineno:D } % noqa: W200
16316 \prop_gput:Nnn
16317 \g_@@_context_loaded_themes_versions_prop
16318 { #1 }
16319 { #2 }

```

Load built-in plain TeX themes from the prop `\g_@@_context_built_in_themes_prop` and from the filesystem otherwise.

```

16320 \prop_if_in:NnTF
16321 \g_@@_context_built_in_themes_prop
16322 { #1 }
16323 {
16324 \msg_info:nxxx
16325 { markdown }
16326 { loading-built-in-context-theme }
16327 { #1 }
16328 { #2 }
16329 \prop_item:Nn
16330 \g_@@_context_built_in_themes_prop
16331 { #1 }
16332 }
16333 {
16334 \msg_info:nxxx
16335 { markdown }
16336 { loading-context-theme }
16337 { #1 }
16338 { #2 }
16339 \usemodule
16340 [t]
16341 [markdown theme #3]
16342 }
16343 }
16344 }
16345 {
16346 \@@_plain_tex_load_theme:nnn
16347 { #1 }
16348 { #2 }
16349 { #3 }
16350 }
16351 }
16352 \msg_new:nnn
16353 { markdown }
16354 { loading-built-in-context-theme }
16355 { Loading-version-#2-of-built-in-ConTeXt-Markdown-theme-#1 }
16356 \msg_new:nnn
16357 { markdown }
16358 { loading-context-theme }

```

```

16359 { Loading~version~#2~of~ConTeXt~Markdown~theme~#1 }
16360 \msg_new:nnn
16361 { markdown }
16362 { repeatedly-loaded-context-theme }
16363 {
16364 Version~#3~of~ConTeXt~Markdown~theme~#1~was~previously~
16365 loaded~on~line~#2,~not~loading~it~again
16366 }
16367 \msg_new:nnn
16368 { markdown }
16369 { different-versions-of-context-theme }
16370 {
16371 Tried~to~load~version~#2~of~ConTeXt~Markdown~theme~#1~
16372 but~version~#3~has~already~been~loaded~on~line~#4
16373 }
16374 \ExplSyntaxOff

```

The `witiko/markdown/defaults` ConTeXt theme provides default definitions for token renderer prototypes. First, the ConTeXt theme loads the plain TeX theme with the default definitions for plain TeX:

```
16375 \markdownLoadPlainTeXTheme
```

Next, the ConTeXt theme overrides some of the plain TeX definitions. See Section 3.4.3 for the actual definitions.

### 3.4.3 Token Renderer Prototypes

The following configuration should be considered placeholder. If the option `plain` has been enabled (see Section 2.2.2.3), none of the definitions will take effect.

```

16376 \markdownIfOption{plain}{\iffalse}{\iftrue}
16377 \def\markdownRendererHardLineBreakPrototype{\blank}%
16378 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
16379 \def\markdownRendererRightBracePrototype{\textbraceright}%
16380 \def\markdownRendererDollarSignPrototype{\textdollar}%
16381 \def\markdownRendererPercentSignPrototype{\percent}%
16382 \def\markdownRendererUnderscorePrototype{\textunderscore}%
16383 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
16384 \def\markdownRendererBackslashPrototype{\textbackslash}%
16385 \def\markdownRendererTildePrototype{\textasciitilde}%
16386 \def\markdownRendererPipePrototype{\char`|}%
16387 \def\markdownRendererLinkPrototype#1#2#3#4{%
16388 \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
16389 \fi\texttt<\hyphenatedurl{#3}>}}%
16390 \usemodule[database]
16391 \defineseparatedlist
16392 [MarkdownConTeXtCSV]
16393 [separator={,},

```

```

16394 before=\bTABLE,after=\eTABLE,
16395 first=\bTR,last=\eTR,
16396 left=\bTD,right=\eTD]
16397 \def\markdownConTeXtCSV{csv}
16398 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
16399 \def\markdownConTeXtCSV@arg{#1}%
16400 \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
16401 \placetable [] [tab:#1]{#4}{%
16402 \processeparatedfile [MarkdownConTeXtCSV] [#3]}%
16403 \else
16404 \markdownInput{#3}%
16405 \fi}%
16406 \def\markdownRendererImagePrototype#1#2#3#4{%
16407 \placefigure [] []{#4}{\externalfigure[#3]}%
16408 \def\markdownRendererUlBeginPrototype{\startitemize}%
16409 \def\markdownRendererUlBeginTightPrototype{\startitemize[packed]}%
16410 \def\markdownRendererUlItemPrototype{\item}%
16411 \def\markdownRendererUlEndPrototype{\stopitemize}%
16412 \def\markdownRendererUlEndTightPrototype{\stopitemize}%
16413 \def\markdownRendererOlBeginPrototype{\startitemize[n]}%
16414 \def\markdownRendererOlBeginTightPrototype{\startitemize[packed,n]}%
16415 \def\markdownRendererOlItemPrototype{\item}%
16416 \def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
16417 \def\markdownRendererOlEndPrototype{\stopitemize}%
16418 \def\markdownRendererOlEndTightPrototype{\stopitemize}%
16419 \definedescription
16420 [MarkdownConTeXtDlItemPrototype]
16421 [location=hanging,
16422 margin=standard,
16423 headstyle=bold]%
16424 \definestartstop
16425 [MarkdownConTeXtDlPrototype]
16426 [before=\blank,
16427 after=\blank]%
16428 \definestartstop
16429 [MarkdownConTeXtDlTightPrototype]
16430 [before=\blank\startpacked,
16431 after=\stoppacked\blank]%
16432 \def\markdownRendererDlBeginPrototype{%
16433 \startMarkdownConTeXtDlPrototype}%
16434 \def\markdownRendererDlBeginTightPrototype{%
16435 \startMarkdownConTeXtDlTightPrototype}%
16436 \def\markdownRendererDlItemPrototype#1{%
16437 \startMarkdownConTeXtDlItemPrototype{#1}}%
16438 \def\markdownRendererDlItemEndPrototype{%
16439 \stopMarkdownConTeXtDlItemPrototype}%
16440 \def\markdownRendererDlEndPrototype{%

```

```

16441 \stopMarkdownConTeXtDlPrototype}%
16442 \def\markdownRendererDlEndTightPrototype{%
16443 \stopMarkdownConTeXtDlTightPrototype}%
16444 \def\markdownRendererEmphasisPrototype#1{\em#1}%
16445 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
16446 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
16447 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
16448 \def\markdownRendererLineBlockBeginPrototype{%
16449 \begingroup
16450 \def\markdownRendererHardLineBreak{
16451 }%
16452 \startlines
16453 }%
16454 \def\markdownRendererLineBlockEndPrototype{%
16455 \stoplines
16456 \endgroup
16457 }%
16458 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%

```

### 3.4.3.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

16459 \ExplSyntaxOn
16460 \cs_gset:Npn
16461 \markdownRendererInputFencedCodePrototype#1#2#3
16462 {
16463 \tl_if_empty:nTF
16464 { #2 }
16465 { \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

```

```

16466 {
16467 \regex_extract_once:nnN
16468 { \w* }
16469 { #2 }
16470 \l_tmpa_seq
16471 \seq_pop_left:NN
16472 \l_tmpa_seq
16473 \l_tmpa_tl
16474 \typefile[\l_tmpa_tl][] {#1}
16475 }
16476 }
16477 \ExplSyntaxOff
16478 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
16479 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
16480 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
16481 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
16482 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
16483 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
16484 \def\markdownRendererThematicBreakPrototype{%
16485 \blackrule[height=1pt, width=\hsize]}%
16486 \def\markdownRendererNotePrototype#1{\footnote{#1}}%
16487 \def\markdownRendererTickedBoxPrototype{${\boxtimes}$}
16488 \def\markdownRendererHalfTickedBoxPrototype{${\boxdot}$}
16489 \def\markdownRendererUntickedBoxPrototype{${\square}$}
16490 \def\markdownRendererStrikeThroughPrototype#1{\overstrikes{#1}}
16491 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
16492 \def\markdownRendererSubscriptPrototype#1{\low{#1}}
16493 \def\markdownRendererDisplayMathPrototype#1{%
16494 \startformula#1\stopformula}%

```

### 3.4.3.2 Tables

There is a basic implementation of tables.

```

16495 \newcount\markdownConTeXtRowCounter
16496 \newcount\markdownConTeXtRowTotal
16497 \newcount\markdownConTeXtColumnCounter
16498 \newcount\markdownConTeXtColumnTotal
16499 \newtoks\markdownConTeXtTable
16500 \newtoks\markdownConTeXtTableFloat
16501 \def\markdownRendererTablePrototype#1#2#3{%
16502 \markdownConTeXtTable={}%
16503 \ifx\empty#1\empty
16504 \markdownConTeXtTableFloat={%
16505 \the\markdownConTeXtTable}%

```

```

16506 \else
16507 \markdownConTeXtTableFloat={%
16508 \placetable{#1}{\the\markdownConTeXtTable}}%
16509 \fi
16510 \beginingroup
16511 \setupTABLE[r][each][topframe=off, bottomframe=off,
16512 leftframe=off, rightframe=off]
16513 \setupTABLE[c][each][topframe=off, bottomframe=off,
16514 leftframe=off, rightframe=off]
16515 \setupTABLE[r][1][topframe=on, bottomframe=on]
16516 \setupTABLE[r][#1][bottomframe=on]
16517 \markdownConTeXtRowCounter=0%
16518 \markdownConTeXtRowTotal=#2%
16519 \markdownConTeXtColumnTotal=#3%
16520 \markdownConTeXtRenderTableRow}
16521 \def\markdownConTeXtRenderTableRow#1{%
16522 \markdownConTeXtColumnCounter=0%
16523 \ifnum\markdownConTeXtRowCounter=0\relax
16524 \markdownConTeXtReadAlignments#1%
16525 \markdownConTeXtTable={\bTABLE}%
16526 \else
16527 \markdownConTeXtTable=\expandafter{%
16528 \the\markdownConTeXtTable\bTR}%
16529 \markdownConTeXtRenderTableCell#1%
16530 \markdownConTeXtTable=\expandafter{%
16531 \the\markdownConTeXtTable\eTR}%
16532 \fi
16533 \advance\markdownConTeXtRowCounter by 1\relax
16534 \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
16535 \markdownConTeXtTable=\expandafter{%
16536 \the\markdownConTeXtTable\eTABLE}%
16537 \the\markdownConTeXtTableFloat
16538 \endgroup
16539 \expandafter\gobbleoneargument
16540 \fi\markdownConTeXtRenderTableRow}
16541 \def\markdownConTeXtReadAlignments#1{%
16542 \advance\markdownConTeXtColumnCounter by 1\relax
16543 \if#1d%
16544 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
16545 \fi\if#1l%
16546 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
16547 \fi\if#1c%
16548 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
16549 \fi\if#1r%
16550 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
16551 \fi
16552 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax

```



```

16553 \else
16554 \expandafter\gobbleoneargument
16555 \fi\markdownConTeXtReadAlignments}
16556 \def\markdownConTeXtRenderTableCell#1{%
16557 \advance\markdownConTeXtColumnCounter by 1\relax
16558 \markdownConTeXtTable=\expandafter{%
16559 \the\markdownConTeXtTable\bTD#1\eTD}%
16560 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
16561 \else
16562 \expandafter\gobbleoneargument
16563 \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`.

```

16564 \ExplSyntaxOn
16565 \cs_gset:Npn
16566 \markdownRendererInputRawInlinePrototype#1#2
16567 {
16568 \str_case:nnF
16569 { #2 }
16570 {
16571 { latex }
16572 {
16573 \@@_plain_tex_default_input_raw_inline:nn
16574 { #1 }
16575 { context }
16576 }
16577 }
16578 {
16579 \@@_plain_tex_default_input_raw_inline:nn
16580 { #1 }
16581 { #2 }
16582 }
16583 }
16584 \cs_gset:Npn
16585 \markdownRendererInputRawBlockPrototype#1#2
16586 {
16587 \str_case:nnF
16588 { #2 }
16589 {
16590 { context }
16591 {
16592 \@@_plain_tex_default_input_raw_block:nn
16593 { #1 }
16594 { tex }

```

```

16595 }
16596 }
16597 {
16598 \@@_plain_tex_default_input_raw_block:nn
16599 { #1 }
16600 { #2 }
16601 }
16602 }
16603 \cs_gset_eq:NN
16604 \markdownRendererInputRawBlockPrototype
16605 \markdownRendererInputRawInlinePrototype
16606 \fi % Closes \markdownIfOption{plain}{\iffalse}{\iftrue}
16607 \ExplSyntaxOff
16608 \stopmodule
16609 \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).

```

16610 \ExplSyntaxOn
16611 \str_if_eq:VVT
16612 \c_@@_top_layer_tl
16613 \c_@@_option_layer_context_tl
16614 {
16615 \use:c
16616 { ExplSyntaxOff }
16617 \@@_if_option:nF
16618 { noDefaults }
16619 {
16620 \@@_if_option:nTF
16621 { experimental }
16622 {
16623 \@@_setup:n
16624 { theme = witiko/markdown/defaults@experimental }
16625 }
16626 }
16627 \@@_setup:n
16628 { theme = witiko/markdown/defaults }
16629 }
16630 }
16631 \use:c
16632 { ExplSyntaxOn }
16633 }
16634 \ExplSyntaxOff
16635 \stopmodule
16636 \protect

```

## References

- [1] LuaTeX development team. *LuaTeX reference manual*. Version 1.10 (stable). July 23, 2021. URL: <https://www.pragma-ade.com/general/manuals/luatex.pdf> (visited on 09/30/2022).
- [2] 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 shortverb 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 Miletić. *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 Ierusalimschy. *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 $\epsilon$</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	21
blankBeforeCodeFence	22
blankBeforeDivFence	22
blankBeforeHeading	22
blankBeforeList	22
bracketedSpans	23, 88, 462
breakableBlockquotes	23
cacheDir	4, 17, 19, 59, 60, 159, 172, 374, 394, 414
citationNbsps	23
citations	24, 91
codeSpans	24
contentBlocks	20, 25, 34
contentBlocksLanguageMap	20
contentLevel	25
debugExtensions	9, 20, 26, 318
debugExtensionsFileName	20, 26
defaultOptions	10, 52, 372, 374
definitionLists	26, 95

eagerCache	17, 372
ensureJekyllData	27
entities.char_entity	218
entities.dec_entity	218
entities.hex_entity	218
entities.hex_entity_with_x_char	218
escape_minimal	222
escape_programmatic_text	222
escape_typographic_text	222
expandtabs	279
expectJekyllData	27, 27
experimental	5, 17, 432
extensions	28, 167, 323
extensions.bracketed_spans	323
extensions.citations	324
extensions.content_blocks	329
extensions.definition_lists	332
extensions.fancy_lists	334
extensions.fenced_code	340
extensions.fenced_divs	345
extensions.header_attributes	350
extensions.inline_code_attributes	351
extensions.jekyll_data	368
extensions.line_blocks	352
extensions.link_attributes	353
extensions.mark	353
extensions.notes	355
extensions.pipe_table	357
extensions.raw_inline	362
extensions.strike_through	362
extensions.subscripts	363
extensions.superscripts	364
extensions.tex_math	364
fancyLists	29, 112–117, 432
fencedCode	30, 39, 92, 100, 118, 386, 389
fencedCodeAttributes	31, 86, 100, 389
fencedDiv	100
fencedDivs	31, 41
finalizeCache	17, 20, 32, 32, 59, 60, 158, 372, 373
frozenCache	20, 32, 60, 78, 158, 386, 394
frozenCacheCounter	32, 373, 422

frozenCacheFileName	20, 32, 59, 373
\g_markdown_diagrams_infostrings_prop	390
gfmAutoIdentifiers	21, 32, 86, 101
hashEnumerators	33
headerAttributes	33, 41, 86, 101
html	34, 104, 105, 446
hybrid	34, 34, 40, 46, 48, 62, 79, 120, 159, 223, 280, 421
inlineCodeAttributes	35, 86, 93
inlineNotes	36
\input	56, 374
\inputmarkdown	162, 163, 165, 464
inputTempFileName	60, 62, 415, 416, 419
\inputyaml	162, 164, 464
iterlines	279
jeekyllData	3, 27, 28, 36, 128–132, 134
\l_file_search_path_seq	421
languages_json	329, 329
lineBlocks	37, 107
linkAttributes	37, 86, 106, 109, 299, 459
mark	38, 110, 457
\markdown	154–156, 425
markdown	153, 153, 154, 424
markdown*	153, 153, 154, 158, 424
\markdownBegin	54, 54–56, 152–155, 162, 163
\markdownCleanup	414
\markdownConvert	414
\markdownEnd	54, 54–56, 152–156, 162, 163
\markdownError	151, 151
\markdownEscape	54, 57, 422
\markdownIfOption	58
\markdownIfSnippetExists	80
\markdownInfo	151, 151
\markdownInput	54, 56, 153, 156, 158, 163, 420, 423
\markdownInputFilename	413
\markdownInputFileStream	415
\markdownInputPlainTeX	423
\markdownLoadPlainTeXTheme	160, 166, 385

<code>\markdownLuaExecute</code>	417, 420
<code>\markdownLuaOptions</code>	410, 414
<code>\markdownMakeOther</code>	152, 463, 464
<code>\markdownOptionFinalizeCache</code>	59
<code>\markdownOptionFrozenCache</code>	59
<code>\markdownOptionHybrid</code>	62
<code>\markdownOptionInputTempFileName</code>	60
<code>\markdownOptionNoDefaults</code>	61
<code>\markdownOptionOutputDir</code>	60, 60, 63, 64
<code>\markdownOptionPlain</code>	61
<code>\markdownOptionStripPercentSigns</code>	62
<code>\markdownOutputFileStream</code>	415
<code>\markdownPrepare</code>	413
<code>\markdownPrepareInputFilename</code>	413
<code>\markdownPrepareLuaOptions</code>	410
<code>\markdownReadAndConvert</code>	152, 415, 424, 425, 464
<code>\markdownReadAndConvertProcessLine</code>	416, 417
<code>\markdownReadAndConvertStripPercentSigns</code>	416
<code>\markdownReadAndConvertTab</code>	415
<code>\markdownRendererAttributeClassName</code>	86
<code>\markdownRendererAttributeIdentifier</code>	86
<code>\markdownRendererAttributeKeyValue</code>	86
<code>\markdownRendererBlockQuoteBegin</code>	87
<code>\markdownRendererBlockQuoteEnd</code>	87
<code>\markdownRendererBracketedSpanAttributeContextBegin</code>	88
<code>\markdownRendererBracketedSpanAttributeContextEnd</code>	88
<code>\markdownRendererCite</code>	91, 91
<code>\markdownRendererCodeSpan</code>	93
<code>\markdownRendererCodeSpanAttributeContextBegin</code>	93
<code>\markdownRendererCodeSpanAttributeContextEnd</code>	93
<code>\markdownRendererContentBlock</code>	94, 94
<code>\markdownRendererContentBlockCode</code>	94
<code>\markdownRendererContentBlockOnlineImage</code>	94
<code>\markdownRendererDisplayMath</code>	125
<code>\markdownRendererDlBegin</code>	95
<code>\markdownRendererDlBeginTight</code>	96
<code>\markdownRendererDlDefinitionBegin</code>	97
<code>\markdownRendererDlDefinitionEnd</code>	97
<code>\markdownRendererDlEnd</code>	98
<code>\markdownRendererDlEndTight</code>	98
<code>\markdownRendererDlItem</code>	96
<code>\markdownRendererDlItemEnd</code>	96

<code>\markdownRendererDocumentBegin</code>	110
<code>\markdownRendererDocumentEnd</code>	110
<code>\markdownRendererEllipsis</code>	41, 98
<code>\markdownRendererEmphasis</code>	99, 138
<code>\markdownRendererError</code>	127
<code>\markdownRendererFancyOlBegin</code>	113, 113
<code>\markdownRendererFancyOlBeginTight</code>	113
<code>\markdownRendererFancyOlEnd</code>	117
<code>\markdownRendererFancyOlEndTight</code>	117
<code>\markdownRendererFancyOlItem</code>	115
<code>\markdownRendererFancyOlItemEnd</code>	115
<code>\markdownRendererFancyOlItemWithNumber</code>	116
<code>\markdownRendererFencedCodeAttributeContextBegin</code>	100
<code>\markdownRendererFencedCodeAttributeContextEnd</code>	100
<code>\markdownRendererFencedDivAttributeContextBegin</code>	100
<code>\markdownRendererFencedDivAttributeContextEnd</code>	100
<code>\markdownRendererHalfTickedBox</code>	126
<code>\markdownRendererHardLineBreak</code>	108
<code>\markdownRendererHeaderAttributeContextBegin</code>	101
<code>\markdownRendererHeaderAttributeContextEnd</code>	101
<code>\markdownRendererHeadingFive</code>	103
<code>\markdownRendererHeadingFour</code>	103
<code>\markdownRendererHeadingOne</code>	102
<code>\markdownRendererHeadingSix</code>	104
<code>\markdownRendererHeadingThree</code>	103
<code>\markdownRendererHeadingTwo</code>	102
<code>\markdownRendererImage</code>	105
<code>\markdownRendererImageAttributeContextBegin</code>	106
<code>\markdownRendererImageAttributeContextEnd</code>	106
<code>\markdownRendererInlineHtmlComment</code>	104
<code>\markdownRendererInlineHtmlTag</code>	104
<code>\markdownRendererInlineMath</code>	125
<code>\markdownRendererInputBlockHtmlElement</code>	105
<code>\markdownRendererInputFencedCode</code>	92
<code>\markdownRendererInputRawBlock</code>	118
<code>\markdownRendererInputRawInline</code>	118
<code>\markdownRendererInputVerbatim</code>	92
<code>\markdownRendererInterblockSeparator</code>	106
<code>\markdownRendererJekyllDataBegin</code>	128
<code>\markdownRendererJekyllDataBoolean</code>	131
<code>\markdownRendererJekyllDataEmpty</code>	134
<code>\markdownRendererJekyllDataEnd</code>	129



<code>\markdownRendererJekyllDataMappingBegin</code>	129
<code>\markdownRendererJekyllDataMappingEnd</code>	130
<code>\markdownRendererJekyllDataNumber</code>	131
<code>\markdownRendererJekyllDataProgrammaticString</code>	132, 132, 133
<code>\markdownRendererJekyllDataSequenceBegin</code>	130
<code>\markdownRendererJekyllDataSequenceEnd</code>	130
<code>\markdownRendererJekyllDataString</code>	133, 137
<code>\markdownRendererJekyllDataStringPrototype</code>	147
<code>\markdownRendererJekyllDataTypographicString</code>	132, 132, 133, 368
<code>\markdownRendererLineBlockBegin</code>	107
<code>\markdownRendererLineBlockEnd</code>	107
<code>\markdownRendererLink</code>	109, 138
<code>\markdownRendererLinkAttributeContextBegin</code>	109
<code>\markdownRendererLinkAttributeContextEnd</code>	109
<code>\markdownRendererMark</code>	110
<code>\markdownRendererNbsp</code>	111
<code>\markdownRendererNote</code>	112
<code>\markdownRendererOlBegin</code>	112
<code>\markdownRendererOlBeginTight</code>	112
<code>\markdownRendererOlEnd</code>	116
<code>\markdownRendererOlEndTight</code>	116
<code>\markdownRendererOlItem</code>	42, 114
<code>\markdownRendererOlItemEnd</code>	114
<code>\markdownRendererOlItemWithNumber</code>	42, 114
<code>\markdownRendererParagraphSeparator</code>	107
<code>\markdownRendererReplacementCharacter</code>	119
<code>\markdownRendererSectionBegin</code>	119
<code>\markdownRendererSectionEnd</code>	119
<code>\markdownRendererSoftLineBreak</code>	108
<code>\markdownRendererStrikeThrough</code>	123
<code>\markdownRendererStrongEmphasis</code>	99
<code>\markdownRendererSubscript</code>	123
<code>\markdownRendererSuperscript</code>	124
<code>\markdownRendererTable</code>	125
<code>\markdownRendererTableAttributeContextBegin</code>	124
<code>\markdownRendererTableAttributeContextEnd</code>	124
<code>\markdownRendererTextCite</code>	91
<code>\markdownRendererThematicBreak</code>	126
<code>\markdownRendererTickedBox</code>	126
<code>\markdownRendererUlBegin</code>	88
<code>\markdownRendererUlBeginTight</code>	89
<code>\markdownRendererUlEnd</code>	90

<code>\markdownRendererUlEndTight</code>	<i>90</i>
<code>\markdownRendererUlItem</code>	<i>89</i>
<code>\markdownRendererUlItemEnd</code>	<i>90</i>
<code>\markdownRendererUntickedBox</code>	<i>126</i>
<code>\markdownRendererWarning</code>	<i>127</i>
<code>\markdownSetup</code>	<i>58, 58, 62, 157, 158, 165, 425, 432</i>
<code>\markdownSetupSnippet</code>	<i>79, 79</i>
<code>\markdownThemeVersion</code>	<i>69, 69, 70</i>
<code>\markdownWarning</code>	<i>151, 151</i>
<code>\markinline</code>	<i>54, 55, 56, 153, 156, 418, 423</i>
<code>\markinlinePlainTeX</code>	<i>423</i>
<code>new</code>	<i>7, 18, 372, 374</i>
<code>notes</code>	<i>38, 112</i>
<code>parsers</code>	<i>239, 279</i>
<code>parsers.commented_line</code>	<i>259</i>
<code>parsers.unicode_data</code>	<i>240</i>
<code>pipeTables</code>	<i>7, 39, 45, 125</i>
<code>preserveTabs</code>	<i>39, 43, 279</i>
<code>rawAttribute</code>	<i>35, 39, 40, 118</i>
<code>reader</code>	<i>8, 29, 167, 239, 278, 323</i>
<code>reader-&gt;add_special_character</code>	<i>8, 9, 29, 317</i>
<code>reader-&gt;auto_link_email</code>	<i>307</i>
<code>reader-&gt;auto_link_url</code>	<i>307</i>
<code>reader-&gt;create_parser</code>	<i>280</i>
<code>reader-&gt;finalize_grammar</code>	<i>313, 379</i>
<code>reader-&gt;initialize_named_group</code>	<i>318</i>
<code>reader-&gt;insert_pattern</code>	<i>8, 9, 29, 313, 319</i>
<code>reader-&gt;lookup_note_reference</code>	<i>292</i>
<code>reader-&gt;lookup_reference</code>	<i>292</i>
<code>reader-&gt;normalize_tag</code>	<i>279</i>
<code>reader-&gt;options</code>	<i>279</i>
<code>reader-&gt;parser_functions</code>	<i>280</i>
<code>reader-&gt;parser_functions.name</code>	<i>280</i>
<code>reader-&gt;parsers</code>	<i>279, 279</i>
<code>reader-&gt;register_link</code>	<i>291</i>
<code>reader-&gt;update_rule</code>	<i>313, 316, 319</i>
<code>reader-&gt;writer</code>	<i>279</i>
<code>reader.new</code>	<i>278, 278, 379</i>
<code>relativeReferences</code>	<i>40</i>

<code>\setupmarkdown</code>	165, 165
<code>\setupyaml</code>	165
<code>shiftHeadings</code>	7, 41
<code>singletonCache</code>	18
<code>slice</code>	7, 41, 219, 231, 232
<code>smartEllipses</code>	41, 98, 159
<code>\startmarkdown</code>	162, 162, 464
<code>startNumber</code>	42, 114–116
<code>\startyaml</code>	162, 163, 464
<code>\stopmarkdown</code>	162, 162, 464
<code>\stopyaml</code>	162, 163, 464
<code>strikeThrough</code>	42, 123, 458
<code>stripIndent</code>	43, 280
<code>stripPercentSigns</code>	415, 416
<code>subscripts</code>	43, 123
<code>superscripts</code>	44, 124
<code>syntax</code>	315, 319
<code>tableAttributes</code>	44, 124, 454
<code>tableCaptions</code>	7, 44, 45, 124
<code>taskLists</code>	45, 126, 445
<code>texComments</code>	46, 280
<code>texMathDollars</code>	35, 46, 125
<code>texMathDoubleBackslash</code>	35, 47, 125
<code>texMathSingleBackslash</code>	35, 47, 125
<code>tightLists</code>	47, 89, 90, 96, 98, 112, 113, 117, 432
<code>underscores</code>	48
<code>unicodeNormalization</code>	18, 19
<code>unicodeNormalizationForm</code>	18, 19
<code>util.cache</code>	167, 168
<code>util.cache_verbatim</code>	168
<code>util.encode_json_string</code>	168
<code>util.err</code>	167
<code>util.escaper</code>	171
<code>util.expand_tabs_in_line</code>	168
<code>util.flatten</code>	169
<code>util.intersperse</code>	170
<code>util.map</code>	170
<code>util.pathname</code>	171
<code>util.rope_last</code>	170
<code>util.rope_to_string</code>	169

util.salt	172
util.table_copy	168
util.walk	169, 169, 170
util.warning	172
walkable_syntax	8, 20, 26, 313, 316–319
writer	167, 167, 219, 323
writer->active_attributes	230, 230–232
writer->attribute_type_levels	231
writer->attributes	228
writer->block_html_element	227
writer->blockquote	227
writer->bulletitem	225
writer->bulletlist	225
writer->citations	325
writer->code	223
writer->contentblock	330
writer->defer_call	238, 238
writer->definitionlist	332
writer->display_math	364
writer->div_begin	345
writer->div_end	346
writer->document	228
writer->ellipsis	221
writer->emphasis	227
writer->error	223
writer->escape	223
writer->escaped_chars	222, 222
writer->escaped_minimal_strings	221, 222
writer->escaped_strings	222
writer->escaped_uri_chars	221, 222
writer->fancyitem	335
writer->fancylist	334
writer->fencedCode	340
writer->flatten_inlines	219, 219
writer->get_state	238
writer->hard_line_break	221
writer->heading	236
writer->identifier	223
writer->image	224
writer->infostring	223
writer->inline_html_comment	226

writer->inline_html_tag	226
writer->inline_math	365
writer->interblocksep	220
writer->is_writing	219, 219
writer->jekyllData	368
writer->lineblock	352
writer->link	224
writer->mark	353
writer->math	223
writer->nbsp	220
writer->note	355
writer->options	219
writer->ordereditem	226
writer->orderedlist	225
writer->paragraph	220
writer->paragraphsep	221
writer->plain	220
writer->pop_attributes	231, 231, 232
writer->push_attributes	231, 231, 232
writer->rawBlock	341
writer->rawInline	362
writer->set_state	238
writer->slice_begin	219
writer->slice_end	219
writer->soft_line_break	221
writer->space	220
writer->span	324
writer->strike_through	362
writer->string	223
writer->strong	227
writer->subscript	363
writer->superscript	364
writer->table	359
writer->thematic_break	221
writer->checkbox	227
writer->undosep	221, 322
writer->uri	223
writer->verbatim	227
writer->warning	172, 223
writer.new	219, 219, 379
\yaml	156

yaml	153, 155, 156, 424
\yamlBegin	54, 55, 152, 155, 163
\yamlEnd	54, 55, 152, 155, 156, 163
\yamlInput	54, 56, 153, 156, 164, 423
\yamlSetup	58